Faster Iterative Algorithm via Efficient, Robust and Low-Dimensional Inner Product Search Data Structure

Zhao Song* Zhaozhuo Xu† Lichen Zhang‡

Abstract

We present a framework that utilizes efficient data structures to improve various problems involving an iterative process. Specifically, we develop a new approach to solve spectral sparsification [BSS12], one-sided Kadison-Singer-typed discrepancy problem [Wea13] and experimental design problem [AZLSW20]. In the heart of our work is the design of inner product search data structures that are efficient, compatible to dimensionality reduction and robust against adversary. To facilitate such a process, we also craft new types of sparse Johnson-Lindenstrauss transform that can be applied quickly on tensor product of vectors.

By leveraging our inner product search data structures with robust and efficient dimensionality reduction, we obtain an algorithm for dynamic vector sparsification problem: given a set of vectors $\{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$ with $\sum_{i=1}^m v_i v_i^\top = I$, find a weight vector $s \in \mathbb{R}^m$ such that $(1-\varepsilon)I \preceq \sum_{i=1}^m s_i v_i v_i^\top \preceq (1+\varepsilon)I$. Our algorithm achieves the optimal support size of s: $|\{s_i : s_i \neq 0\}| = \Theta(d/\varepsilon^2)$, with an update time (insert/delete vector v_j) of $\widetilde{O}((m^\rho + \mathrm{nnz}(v_j))d^2)$ and a query time (output a sparsifier on the new set) $\widetilde{O}(\varepsilon^{-2}d(m^\rho + d^\omega))$, where $\rho \approx 0.05$. Our algorithm is also robust against adaptive adversary. Prior to our results, all dynamic algorithms would either yield a sub-optimal size ($\Theta((d\log d)/\varepsilon^2)$) or have a linear runtime dependence on m, which is prohibitive when m is large ($m \gg d^2$). By using this framework, we also exhibit runtime improvement for the one-sided Kadison-Singer-typed discrepancy problem and experimental design problem.

*zsong@adobe.com. Adobe Research.
†zx22@rice.edu. Rice University.
‡lichenz@andrew.cmu.edu. Carnegie Mellon University.

1 Introduction

In recent years, there is a growing trend of using efficient data structures in efforts to improve the running time of various optimization problems [CLS19, LSZ19, Bra20, BLSS20, JSWZ21, Bra21]. In this paper, we deploy this idea to a new class of problem including graph sparsification or more general sum-of-rank-1-matrices sparsification [SS11, BSS12, AZLO15, LS15, LS17], a one-sided "packing" version of a Kadison-Singer type discrepancy problem [Sri10, Wea13] and an experimental design problem [AZLSW20]. All these problems can be formulated as follows: given a set of vector $V := \{v_1, \ldots, v_m\}$. At each iteration, one needs to compute a query vector q and find the vector $v_i \in V$ that maximizes or minimizes the inner product $\langle v_i, q \rangle$ respectively. Previous developments on efficient algorithms for these kinds of problems typically take the following form: at each iteration, search through all vectors in V and find the desired vector v_i to proceed, incurring a cost linear on m. Clearly, such a process is sub-optimal suppose one is allowed to *preprocess* the set V, since one would hope to obtain a sublinear cost per iteration in terms of m by exploiting the inner product search structure of these problems.

For the sake of illustration, let's focus the problem of computing an optimal-sized sum-of-rank-1-matrices sparsifier, which can be viewed as a generalization of the graph spectral sparsifier [SS11, BSS12]. Consider the following dynamic scenario: we start with the set V and we are asked to construct a sparsifier for V. At each timestamp, a constant number of v_i's have been removed from V and a constant number of new v_j's have been added and we are tasked with outputting a sparsifier. One approach is to consider the use of dynamic data structures for maintaining such sparisfier (see, e.g., [CGH+20, BBG+20]), however, all these dynamic data structures are sub-optimal in the size of sparsifier ($\Theta(n \log n)$) in the graph setting, and it is not clear how to generalize these results the general rank-1 sparsification task since they heavily exploited graph-specific structures, such as having access to trees and expanders. This is in contrast of the much more general result obtained by Batson, Spielman and Srivastava [BSS12]. On the flip side, one can simply run an efficient implementation of BSS sparsifier [BSS12, AZLO15, LS15, LS17] at each timestamp, which would require to *re-run* their algorithms on the new set V', incur a running time at least linear in m.

In this work, we show how to satisfy the above two goals simultaneously via a novel combination of efficient and robust data structures and the iterative-construction of sparsifiers as in [BSS12]: as long as the number of insertion/deletion is sublinear in m, then we can construct a sparsifier of the optimal size in time sublinear in m. Our method is also *robust against adaptive adversary*, which, to the best of our knowledge, is only obtained in [BBG+20] in the case of graph. Interestingly, the robustness and dynamic maintenance follows naturally from the data structure we designed for the iterative process.

The heart of our work is to craft data structure that supports the inner product-typed queries as we demonstrated. It must satisfy the following guarantees simultaneously: it has moderate preprocessing time and supports efficient query, insertion and deletion. In the meantime, it must be robust against adaptive query to ensure the correctness of the method itself. Additionally, it should be compatible with fast dimensionality reduction techniques. As we will see later, if not dealt carefully, the efficiency of the data structure will be diminished by the large input dimension.

The data structure we will be implementing is the so-called Max-IP/Min-IP data structures [ACW16, CW19]: given a set of vectors V that can be preprocessed, such data structures can answer the query of the following type: given a query point q, return a vector $v \in V$ such that $\langle q, v \rangle$ is maximized or minimized, in a time efficient manner (the query time is sublinear in m). Moreover, these data structures support insertion and deletion. With such data structure in hand, a natural idea for solving these tasks is to first preprocess all vectors in V, then at each iteration use the data structure to answer the query efficiently. However, a simple combination won't work, due to the

following issues: 1). The input vectors might be of high dimension. Although the input vectors $V = \{v_i, \ldots, v_m\} \in (\mathbb{R}^d)^m$ are of dimension d, the input dimension to these data structures are in fact d^2. This significantly reduces the efficiency of both preprocessing and querying with these data structures. 2). If we are to use randomized data structures, we have to deal with adaptive adversary — in standard setting, it is natural to assume the adversary is *oblivious* with respect to the randomness of the data structure and all queries are *independent* of each other. However, this is no longer the case in our application, since a query is *dependent* on all previous query answers. To handle this problem, it is instructive to design *robust* data structures against *adpative adversary*.

We show how to realize the efficiency, robustness and dimensionality reduction requirements all at once for the Max-IP/Min-IP data structures. For the Max-IP data structure, we show that it is equivalent to design a *nearest neighbor search* (NNS) data structures, which is well-studied [Cla83, AM93, Cla97, IM98, DIIM04, C$^+$06, AINR14, AIL$^+$15, AR15, ALRW17, IW18, AIR18, DIRW19, CCD$^+$20, LMWY20]. On the flip side, the Min-IP data structure corresponds to the *furthest neighbor search* (FNS) data structure [Ind03]. However, as we have argued before, it is not enough to use these data structures in a straightforward manner — to some extent, they only tackle the efficiency issue, while leaving the dimension and robustness problems untouched.

1.1 Related Work

Graph Sparsification and General Sum of Rank-1 Matrices Sparsification. Given a potentially dense graph (G, V, E) with many edges, one of the fundamental questions is to compute a *sparsifier* of the graph, i.e., a new graph (H, V, E') with fewer weighted edges but certain properties are still preserved, e.g., all cuts in the graph [BK96] or the spectral property of the Laplacian matrix [SS11, BSS12]. To preserve the spectral structure of the Laplacian matrix, Batson, Spielman and Srivastava [BSS12] showed a deterministic construction with an optimal $\Theta(n/\varepsilon^2)$ edges which can also be extended to sparsify a general sum-of-rank-1-matrices. Following their work, Allen-Zhu, Liao and Orecchia [AZLO15] used a regret minimization framework to tackle the general sum-of-rank-1-matrices or even the sum of PSD matrices. To improve the runtime efficiency of generating a graph or sum-of-rank-1-matrices sparsifier, Lee and Sun [LS15] combined the idea of regret minimization [AZLO15] and importance sampling [SS11] to obtain a running time of $\widetilde{O}(\varepsilon^{-2} q m n^{\omega-1+q/3})$ for some positive even integer q. Subsequently, they improved the running time to $\widetilde{O}(\varepsilon^{-O(1)} m)$ via implementing a one-sided oracle that can be solved via semi-definite programming (SDP) efficiently [LS17]. We highlight that in the general sum-of-rank-1-matrices sparsification setting, all these results [BSS12, AZLO15, LS15, LS17] are essentially optimal in terms of the size of sparsifier ($\Theta(n/\varepsilon^2)$). However, these algorithms are inherently static: when there are vector insertions/deletions, it is not clear how to avoid the linear dependence on m in the running time.

Apart from graph sparsfication, the BSS construction also finds applications in constrained linear regression, multi-response regression [BDMi13], matrix CUR decomposition [BW14] and tensor CURT decomposition [SWZ19].

There is also a long line of research concerning maintaining a graph sparsifier under dynamic edge insertions and deletions [CGH$^+$20, BBG$^+$20] and in a dynamic data stream [KW14, KLM$^+$, KNST19, KMM$^+$20]. In particular, they can achieve an update time of $\Theta(\mathrm{poly}(\log n, \varepsilon^{-1}))$. However, one clear downside of their approaches is they can only get a sparsifier of size $\Theta(\varepsilon^{-2} n \log n)$, which is sub-optimal. Moreover, the machinery developed in their works is highly reliant on the structure of graphs, and hence it is not clear how to extend them for general rank-1 or PSD matrices sparsification.

Robust Data Structures Handling Adaptive Adversary. Often times, when designing efficient randomized data structures, it is not enough to assume the query sequence is from an *oblivious adversary*, i.e., the query sequence is fixed before the randomness of the data structure has been determined. Rather, queries can be from a *adaptive adversary*, i.e., each query is dependent on the outputs of all previous answers from data structure. This is particularly the case when one wants to combine data structure with an iterative process. One typical handling is to require the data structure to be deterministic, however, this would potentially result in far worse running time. Another idea is to require the data structure to work for *all possible inputs*. Specifically, for the task of preserving pair-wise distances, this is the so-called *terminal embedding* [EFN17]. Recently, there are more efforts have been put to develop optimal-dimension embedding [MMMR18, NN19] and efficient algorithm for this problem [MMMR18, NN19, CN21]. Narayanan and Nelson [NN19] showed that one can achieve an optimal dimension of embedding ($\Theta(\varepsilon^{-2}\log m)$) while Cherapanamjeri and Nelson [CN21] designed data structures that support sublinear query time in terms of the size of V. However, their methods suffer from high preprocessing time and are inherently static. It is not clear how to adapt such data structure for dynamic tasks.

Sparse Johnson-Lindenstrauss Transform. Johnson-Lindenstrauss transforms [JL84] are powerful primitives to reduce the dimension of a set of vectors while preserving their pair-wise distances. However, the original Johnson-Lindenstrauss matrix is a dense matrix with entries sampled from a Gaussian distribution, hence computing the low-dimension projection is expensive. Designing new distribution of matrices that satisfy the Johnson-Lindenstrauss lemma while can be applied efficiently to vectors or matrices has been an active line of research [AC06, DKS10, FL19]. Dasgupta, Kumar and Sarlos have shown that there exists a distribution of matrices such that each column only has $\Theta(\log^2(m/\delta))$ non-zero entries that satisfy the Johnson-Lindenstrauss lemma, which means one can apply the JL matrix to a vector in time proportional to the *number of non-zero entries* (nnz) of the vector. Subsequently, Kane and Nelson improved the sparsity per column to $\Theta(\log(m/\delta))$ [KN10, KN14]. Nelson and Nguyên [NN13] further showed that such construction yields an *oblivious subspace embedding* [Sar06]. The proof of this sparse construction is simplified by Cohen, Jayram and Nelson [CJN18].

Sketching Techniques for Tensor-Typed Inputs. Given two vectors $u, v \in \mathbb{R}^d$, computing their tensor product $u \otimes v$ would naively take $O(d^2)$ time. However, if one only cares about preserving the ℓ_2 norm of tensor product, then various sketching techniques can be exploited to 1). facilitate efficient construction of an approximate tensor product, 2). reduce the target dimension from d^2 to a much smaller dimension. Common approaches including TensorSketch [Pag13, ANW14], which can be applied to two vectors in input sparsity time, and TensorSRHT [AKK+20], which gives a high probability guarantee and supports Fast Fourier Transform (FFT) to compute the tensor product. These prototypes also find applications to compute relative error tensor low rank approximation [SWZ19], Kronecker product regression [DSSW18, DJS+19] and approximate a wide range of kernels [AKK+20, WZ20, SWYZ21].

2 Our Results

2.1 Max-IP and Min-IP: Efficient, Dimensionality Reduction and Robust

Our first result regards a pair of data structures that are efficient, support dimensionality reduction and robust against adaptive adversary. More concretely, we realize the Max-IP data structure via efficient locality-sentive hashing (LSH)-based NNS data structures [AR15, ALRW17] and the

Min-IP data structure via a random projection-based data structure that implements FNS. We start by describing the random projection-based FNS data structure.

The idea is to first design a data structure that solves the decision version of the FNS problem: given a query point q and a distance estimate r, output a point $v \in V$ such that $\|v - q\|_2 \geq r$. To do this efficiently, we compute $\ell = \Theta(m^{1/c^2})$ random directions for some $c > 1$, then project all points in V onto each of the random direction. We then sort each direction corresponds to the value of projection. Given a query point q, we project it onto all ℓ directions and use binary search to find at least 2ℓ points (v, i) such that the projection of v on direction i is at most $\widetilde{O}(r/c)$ away from the projection of q on direction i. By the concentration and anti-concentration of Gaussian distribution, one can conclude that with good probability, $\|p - v\|_2 \geq r/c$.

One can then perform binary search using this decision data structure. We remark that this construction resembles of that introduced by Indyk [Ind03]. However, this is not sufficient for our application, since it is only efficient, not robust against adaptive adversary.

The next is to tackle the dimensionality reduction problem. A standard practice is to apply the Johnson-Lindenstrauss transform [JL84] to all vectors in V and all subsequent queries q. However, the JL transform must also be robust against adaptive adversary, since the JL matrix cannot be chosen independently at each iteration. A natural handle is to union bound over all possible query points, which blows up the dimension and essentially renders the JL transform useless. To circumvent this problem, we take an alternative perspective of the union bound idea: instead of blowing up the dimension, we instantiate $\widetilde{O}(d^2)$ independent JL transforms, each of dimension only $O(\log(m/\delta))$, then with high probability, a constant fraction of them will preserve the distances between all queries and vectors in V. Hence, it suffices for us to uniformly pick $\Omega(\log m)$ of them at query time. In fact, we generalize this argument to a much larger range family of JL transforms that can be efficiently applied to both input vectors and query vectors.

Now we have dealt with the dimension problem, the next question is: how to further augment the search data structures so that they are *inherently robust*? Our strategy is to use a quantization process, which is essentially a γ-net argument: for each query q, pick its nearest neighbor $\widehat{q}$ in the γ-net and use $\widehat{q}$ as the new query. By doing so, we remove the dependence between queries, and it suffices to union bound over all points in the γ-net. The key observation here is the net argument is done in a much *lower-dimension*, since we have already used robust JL transform to reduce the dimension. Although we need to use extra independent data structures to boost the success probability, the number is proportion to the *dimension input to the data structure*, which is logarithmic in m.

We remark that these two-stages of robustifying the data structure will incur a small additive error in the final estimation, but this error can be controlled and it also suffices for our applications.

We present our two main data structure results in the following two theorems:

Theorem 2.1 (Informal version of Theorem 5.11)**.** *Let $\mathcal{T}_S(x)$ to denote the time of applying a JL transform matrix $S \in \mathbb{R}^{s \times d}$ to a vector $x \in \mathbb{R}^d$. Let $c \in (0,1), \tau \in (0,1)$ and $\rho \in (0.05, 0.06)$.*

Given a set of m-points $V \subset \mathbb{S}^{d-1}$ on the sphere, one can build a data structure with preprocessing time $\mathcal{T}_{\text{init}} = \widetilde{O}(dm^{1+\rho} + d \cdot \mathcal{T}_S(V))$ so that for every query $q \in \mathbb{S}^{d-1}$ in an adaptive sequence $Q = \{q_1, \ldots, q_T\}$, the query time is $\widetilde{O}(m^\rho + \mathcal{T}_S(q))$, with the following guarantee:

- *Let $v^* \in V$ be the vector such that $\langle v^*, q \rangle$ is maximized among all $v \in V$ and $\langle v^*, q \rangle \geq \tau$, then we output a vector $\widehat{v} \in V$ such that $\langle \widehat{v}, q \rangle \geq c \cdot \tau - O(\frac{1}{m^{O(1)}})$.*
- *Otherwise, we output* fail.

Theorem 2.2 (Informal version of Theorem 6.14)**.** *Let $\mathcal{T}_S(x)$ to denote the time of applying a JL transform matrix $S \in \mathbb{R}^{s \times d}$ to a vector $x \in \mathbb{R}^d$. Let $c \in (0,1), \tau \in (0,1)$.*

Given a set of m-points $V \subset \mathbb{S}^{d-1}$ on the sphere, one can build a data structure with preprocessing time $\mathcal{T}_{\mathrm{init}} = \widetilde{O}(dm^{1.01} + d \cdot \mathcal{T}_S(V))$ so that for every query $q \in \mathbb{S}^{d-1}$ in an adaptive sequence $Q = \{q_1, \ldots, q_T\}$, the query time is $\widetilde{O}(m^{0.01} + \mathcal{T}_S(q))$, with the following guarantee:

- *Let $v^* \in V$ be the vector such that $\langle v^*, q \rangle$ is minimized among all $v \in V$ and $\langle v^*, q \rangle \geq \tau$, then we output a vector $\widehat{v} \in V$ such that $\langle \widehat{v}, q \rangle \leq \frac{1}{c} \cdot \tau + O(\frac{1}{m^{O(1)}})$.*
- *Otherwise, we output* fail.

2.2 Sparse Johnson-Lindenstrauss Transforms for Tensor-Typed Input

A key step in our framework is to apply Johnson-Lindenstrauss transform fast to a certain type of inputs, specifically, input in the form of tensor product: $v_i \otimes v_i$. One might consider to use two of the popular sketching matrices for tensors: TensorSRHT and TensorSketch. However, both of them are not enough for our purpose: while TensorSketch is sparse and can be applied to vectors in time proportional to the number of non-zero entries of input, it has a sub-optimal dependence on target dimension, render it useless if one wishes to union bound a set of m points. On the other hand, TensorSRHT can achieve a relatively small target dimension, but it cannot be applied to vectors in input-sparsity time.

To achieve both a nearly input-sparsity running time and a good target dimension, we generalize the sparse embedding matrix [DKS10, KN10, NN13, KN14, CJN18] so that it can be applied to the input vector of the form $u \otimes v$ quickly, where $\otimes$ denote the tensor product of two vectors: $u \otimes v = \mathrm{vec}(uv^\top)$.

To understand our construction better, we first recall the TensorSketch [Pag13, ANW14] matrix, which is an extension of the well-known CountSketch matrix [CCFC02]. We can define the CountSketch matrix $S \in \mathbb{R}^{k \times d}$ as follows: let $h : [d] \to [k]$ be a 3-wise independent hash function $\sigma : [d] \to \{\pm 1\}$ be 4-wise independent then

$$S_{r,i} = \sigma(i) \cdot \mathbf{1}[h(i) = r].$$

By definition, CountSketch has only one non-zero entry per column. We can make use of this idea to construct a sparse embedding matrix as in [KN14]: we partition each column into s blocks so that each block contains k/s consecutive entries. Then, we perform the CountSketch construction for each block. Formally, let $h : [d] \times [s] \to [k/s]$ be $O(s)$-wise independent as well as $\sigma : [d] \times [s] \to \{\pm 1\}$, further suppose index r_t is in the t-th block, then

$$S_{r_t,i} = \sigma(i,t) \cdot \mathbf{1}[h(i,t) = r_t].$$

By design, S will have exactly s non-zero entries at each column.

The definition of TensorSketch is similar to that of CountSketch, except it makes use of two independent hash functions and sign functions: let $h_1, h_2 : [d] \to [k]$ and $\sigma_1, \sigma_2 : [d] \to \{\pm 1\}$. The TensorSketch matrix $S \in \mathbb{R}^{k \times d^2}$ is defined as follows: suppose we use (i,j) to denote the (i,j)-th column of S, then

$$S_{r,(i,j)} = \sigma_1(i) \cdot \sigma_2(j) \cdot \mathbf{1}[h_1(i) + h_2(j) = r \bmod k].$$

One advantage of this construction is it can be applied to tensor of vectors quickly: given $u, v \in \mathbb{R}^d$, one can compute $S(u \otimes v)$ in time $O(\mathrm{nnz}(u) + \mathrm{nnz}(v) + k \log k)$.

Inspired by this generalization,

we define a new type of sparse Johnson-Lindenstrauss transform for tensors: the TensorSparse *matrix.*

More specifically, let $h_1, h_2 : [d] \times [s] \to [k]$ and $\sigma_1, \sigma_2 : [d] \times [s] \to \{\pm 1\}$, and suppose we are considering row index r_t in the t-th block, then

$$S_{r_t,(i,j)} = \sigma_1(i,t) \cdot \sigma_2(j,t) \cdot \mathbf{1}[h_1(i,t) + h_2(j,t) = r_t \bmod k/s].$$

Essentially, this is the TensorSketch for s blocks. Hence, compute $S(u \otimes v)$ takes $O(s \cdot (\mathrm{nnz}(u) + \mathrm{nnz}(v)) + k \log(k/s))$. Next, we would like to prove the TensorSparse is a sparse Johnson-Lindenstrauss transform, hence, it suffices to set k as $\Theta(\varepsilon^{-2} \log(m/\delta))$ and s as $\Theta(\varepsilon^{-1} \log(m/\delta))$, which is optimal in terms of target dimension [LN17]. One idea is to use the second moment analysis for the TensorSketch matrix, i.e., define $Z = \|Sx\|_2^2 - 1$, one can analyze the quantity $\mathbb{E}[Z^2]$. However, this would lead to a similar conclusion as in [ANW14], which does not support a union bound over all m points. Therefore, it is necessary to prove for a higher moment.

Our proof is by a crucial observation: if we define $H : [d] \times [d] \times [s] \to [k/s]$ with $H(i,j,t) := h_1(i,t) + h_2(j,t) \bmod k/s$, then H is also $O(s)$-wise independent. The same holds if we set $\sigma : [d] \times [d] \times [s] \to \{\pm 1\}$ as $\sigma(i,j,t) := \sigma_1(i,t) \cdot \sigma_2(j,t)$. Hence, we can apply a Hanson-Wright inequality [HW71] type argument to conclude that our TensorSparse is a Johnson-Lindenstrauss transform. Such argument provides an entry to analyze higher moment behavior with only $O(s)$-wise independence.

One interesting and crucial aspect of our argument is it does not blow up the target dimension, which makes it highly applicable when combined with our Max-IP/Min-IP data structures. We summarize the property of our novel TensorSparse matrix as follows:

Theorem 2.3 (Informal version of Theorem 4.10)**.** *Let $V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$, then the* TensorSparse *matrix S with $k = \Theta(\varepsilon^{-2} \log(m/\delta))$ rows and each column has $s = \Theta(\varepsilon^{-1} \log(m/\delta))$ sparsity has the property that, for any $v_i, v_j \in V$,*

$$(1-\varepsilon)\|v_i \otimes v_i - v_j \otimes v_j\|_2 \leq \|S(v_i \otimes v_i) - S(v_j \otimes v_j)\|_2 \leq (1+\varepsilon)\|v_i \otimes v_i - v_j \otimes v_j\|_2$$

with probability at least $1 - \delta$.

We remark the TensorSparse is especially valuable when preprocessing the set V, since we would like to feed V^2 to Max-IP/Min-IP data structures, which consists of the tensor product of vectors in V. Using a naive sparse embedding matrix, one might have to first compute the tensor product which takes $O(md^2)$ time, then apply the sparse embedding matrix in time $\widetilde{O}(s \cdot \mathrm{nnz}(V^2))$. On the other hand, using TensorSparse will give us a running time of $\widetilde{O}(s \cdot \mathrm{nnz}(V))$. Also, note that $\mathrm{nnz}(V) = \sum_{i=1}^m \mathrm{nnz}(v_i)$, while $\mathrm{nnz}(V^2) = \sum_{i=1}^m \mathrm{nnz}(v_i)^2$. When the input does not have tensor structure, using TensorSparse would still yield a nearly input sparsity time to apply.

Though we develop and utilize TensorSparse mainly for the purpose of our data structure task, the family of matrices itself might be of independent interest, e.g., in designing subspace embedding for polynomial kernels [AKK+20, WZ20, SWYZ21] and improving various downstream tasks, such as sketching Gaussian kernels, p-convergent kernels and neural tangent kernels [SWYZ21].

2.3 Combining Robust Data Structure with Iterative Process

Now that we have designed the desired data structure, we shall utilize them to develop efficient iterative algorithms for various of problems we mentioned above.

Sum-of-Rank-1-Matrices Sparsifier. We start by studying the size $\Theta(d/\varepsilon^2)$ sparsifier introduced by Baston, Spielman and Srivastava [BSS12] in a more general setting[1]. We are given a set of vectors $V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$ satisfying $\sum_{i=1}^m v_i v_i^\top = I$, the goal is to construct a sparsifier such that $(1-\varepsilon)I \preceq \sum_{i=1}^m s_i v_i v_i^\top \preceq (1+\varepsilon)I$ with the property that $|\{s_i : s_i \neq 0\}| = \Theta(d/\varepsilon^2)$.

To better describe the iterative process invented in [BSS12], we define the following two barrier functions:

Definition 2.4. Let $A \in \mathbb{R}^{d\times d}$ by a symmetric matrix and $u, \ell \in \mathbb{R}$ be two parameters, we define the two barrier potential functions:

$$\Phi^u(A) := \mathrm{tr}[(uI - A)^{-1}] = \sum_{i=1}^d \frac{1}{u - \lambda_i},$$
$$\Phi_\ell(A) := \mathrm{tr}[(A - \ell I)^{-1}] = \sum_{i=1}^d \frac{1}{\lambda_i - \ell}.$$

The BSS sparsifier maintains two initial barriers $u_0 = d/\varepsilon$ and $\ell_0 = -d/\varepsilon$ and an initial matrix $A_0 = \mathbf{0}_{d\times d}$, then at each iteration $t \in [T]$, the two barriers are incremented respectively: $u_t = u_{t-1} + \delta_U, \ell_t = \ell_{t-1} + \delta_L$, and define the following quantities:

$$L_t = \frac{(A_{t-1} - \ell_t I)^{-2}}{\Phi_{\ell_t}(A_{t-1}) - \Phi_{\ell_{t-1}}(A_{t-1})} - (A_{t-1} - \ell_t I)^{-1}$$
$$U_t = \frac{(u_t I - A_{t-1})^{-2}}{\Phi^{u_{t-1}}(A_{t-1}) - \Phi^{u_t}(A_{t-1})} + (u_t I - A_{t-1})^{-1},$$

the algorithm proceeds by finding an index $j \in [m]$ that *witnesses the gap between lower and upper barriers*, i.e., $v_j^\top L_t v_j \geq v_j^\top U_t v_j$, which is equivalent to $v_j^\top (L_t - U_t) v_j \geq 0$. The core result proved in [BSS12] is that if we set δ_U, δ_L properly, then such a condition can always be satisfied.

After finding such v_j, one then uses $c \cdot v_j v_j^\top$ to update A, where $c = \frac{1}{2} v_j^\top (L_t + U_t) v_j$. After $T = \Theta(d/\varepsilon^2)$, the resulted matrix A_T/d satisfies the desired property.

We identify a key step of this iterative process is to perform a *search*: finding an index j such that $v_j^\top (L_t - U_t) v_j \geq 0$. Note that if one instead looks for the index that *maximizes* the quantity $v_j^\top (L_t - U_t) v_j$, then it is guaranteed by [BSS12] that it is non-negative.

This can be recast into the following maximum inner product search problem of two matrices:

$$\langle v_j v_j^\top, L_t - U_t \rangle,$$

where the inner product between two matrices A and B is $\mathrm{tr}[A^\top B]$. If one further vectorizes these two matrices, then this is nothing more than a standard maximum inner product search problem between two vectors:

$$\langle v_j \otimes v_j, \mathrm{vec}(L_t - U_t) \rangle.$$

Hence, our strategy will be using a Max-IP data structure to preprocess all vectors in the form of $v_i \otimes v_i$, and at each iteration, we form the matrix $L_t - U_t$ accordingly, and then use it as a query point to find the desired v_j. However, several challenges remain for this framework.

Reducing the d^2 Dimension. Note that the vectors passing into the Max-IP data structure are in the form of $v_i \otimes v_i$, and therefore, are of dimension d^2. Using any NNS-based Max-IP data

[1] We will later refer to this sparsifier as BSS sparsifier.

structure, one has to pay at least $\Theta(d^2)$ for *both* preprocessing and querying, which is not optimal. To address this problem, we make use of the TensorSparse Johnson-Lindenstrauss transform that can be applied to preprocess these tensor-typed inputs quickly and to general queries in nearly-input sparsity time. Moreover, we can augment TensorSparse to handle adaptive adversary by using $\widetilde{\Theta}(d^2)$ independent sketches at initialization, and sample $\Theta(\log m)$ of them in the query time. Moreover, this enables the Max-IP data structure to operate in a lower-dimension, and avoids the quadratic dependence on d in the query time.

Robust Max-IP for Dependent Queries. In the iterative process, the queries are dependent of each other: the query at time t is dependent on the matrix A_{t-1}, which in turn depends on the query at time $t-1$. Fortunately, we can deal with adaptive query using the Max-IP we obtained in Theorem 2.1. By utilizing it, we not only guarantee the correctness of our algorithm, but augment the framework itself to be robust against adaptive adversary as well: consider the dynamic setting we've described in the introduction section, the queries we get might be from an adaptive adversary. The strong guarantee given by a robust Max-IP data structure with a robust Johnson-Lindenstrauss transform automatically provides robustness for our framework against adaptive adversary.

Obtain Efficient Data Structure via Approximation. One caveat of using LSH-based method to implement Max-IP data structure is it only solves an *approximate* version of the problem in the following sense: given a query q, suppose $v^* \in V$ maximizes $\langle q, v^* \rangle$, then our Max-IP data structure will output a vector $\widehat{v}$ such that $\langle q, \widehat{v} \rangle \geq c \cdot \langle q, v^* \rangle - \lambda$, for some $c \in (0,1)$ and small additive error λ. Luckily, for BSS we are not required to report the optimal estimate v^*, rather, it is enough to find a $v \in V$ such that $\langle q, v \rangle \geq 0$. To make sure the vector $\widehat{v}$ is good enough, we slightly change the parameters δ_U and δ_L to guarantee that $\langle q, v^* \rangle \geq \frac{\varepsilon}{m}$, and further observe that λ is very small, one can set $c = \Theta(\frac{1}{\varepsilon m^{O(1)}})$ so that $\langle q, \widehat{v} \rangle \geq 0$ holds with probability 1.

We remark that, while it is natural to use Max-IP search for an approximate *optimal* solution, it is quite non-standard to adapt it for searching non-negative inner product. Such an adaptation has the distinctive advantage of near-optimal preprocessing and query time: we can set the parameter c small for a very crude estimation, which can be utilized to optimize over the runtime dependence on m^ρ. The fundamental reason is that we are using Max-IP data structure to solve a task where the optimality condition is not actually required. We hope such use of Max-IP data structure can find more applications outside the context of BSS sparsifier.

We summarize our results as follows:

Theorem 2.5 (Informal version of Theorem 7.9 and Theorem 7.11)**.** *Let $V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$ such that $\sum_{i=1}^m v_i v_i^\top = I$. Let $\rho \in (0.05, 0.06)$. There exists a randomized algorithm to find a set of weights $\{s_i\}_{i=1}^m$ (success with high probability) such that*

$$(1-\varepsilon) I \preceq \sum_{i=1}^m s_i v_i v_i^\top \preceq (1+\varepsilon) I$$

and $|\{s_i : s_i \neq 0\}| = \Theta(d/\varepsilon^2)$. Moreover, the running time of this algorithm is

$$\mathcal{T}_{\mathrm{init}} + \varepsilon^{-2} d \cdot \mathcal{T}_{\mathrm{iter}},$$

where [2]

$$\mathcal{T}_{\mathrm{init}} = \widetilde{O}(m^{1+\rho} d^2 + \mathrm{nnz}(V) \cdot d^2),$$
$$\mathcal{T}_{\mathrm{iter}} = \widetilde{O}(m^\rho d^2 + d^\omega).$$

[2] ω is the exponent of matrix multiplication, and currently $\omega \approx 2.37$ [Wil12, LG14, AW21].

Extension to Dynamic Setting. We now show how to extend the above result to the dynamic setting. In the dynamic scenario, for each timestamp, a constant number of v_i's have been removed from V and a constant number of v_j's have been added to V. Then we are asked to output a sparsifier for the new set V'.

To solve this problem using our method, we only need to implement insertion and deletion efficiently for our Max-IP data structure, this can be done in the same time as query, as shown in [OvL81, AR15, ALRW17]. However, it is worth noticing that we need to perform insertion and deletion for all $\widetilde{\Theta}(d^2)$ data structures, hence we will pay $\widetilde{O}(m^\rho d^2)$ for an update. Then, we can simply perform the iterative algorithm on the new dataset V', which takes $\widetilde{O}(\varepsilon^{-2}d\cdot(m^\rho+d^\omega))$ time.

We remark that though this result seems less impressive for small m regime, such as in graph setting, where $m \leq O(d^2)$. In the more general sum-of-rank-1-matrices sparsification task we consider, it is possible that $m \gg d^2$, hence removing the linear dependence on m is critical. Such application arises when one uses BSS sparsifier for large-scale numerical linear algebra task, e.g., constrained linear regression and multiple-response regression [BDMi13], matrix CUR decomposition [BW14] and tensor CURT decomposition [SWZ19]. To the best of our knowledge, our method is the only that can handle adaptive queries, output an optimal-sized sparsifier and remove the linear dependence on m all at once.

One-Sided Kadison-Singer Problem. In the "one-sided Kadison-Singer problem" posed by Weaver [Wea13], one is given a set of vectors $V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$ with $\sum_{i=1}^m v_i v_i^\top$ and $\|v_i\|_2 = \frac{1}{\sqrt{N}}$ for some positive value N, the task is to find a subset $S \subseteq [m]$ with $|S| = n$ such that $\|\sum_{i\in S} v_i v_i^\top\| \leq \frac{n}{m} + O(\frac{1}{\sqrt{N}})$. We note that this is a much restricted one-sided version of the Kadison-Singer problem, in the sense that Kadison-Singer asks us to also obtain a *lower bound of eigenvalues* on set S. It hence resembles a similarity of the restricted invertibility problem [Sri10], where one only worries to find a set S with lower bound on eigenvalues.

To solve this problem, we adapt a similar approach as that of [Sri10], i.e., using only one barrier functions instead of two as in [BSS12]. Specifically, we use the upper barrier function $\Phi^u(A) = \mathrm{tr}[(uI - A)^{-1}]$ to progress. At each iteration, the algorithm looks for an index $j \in [m]$ such that $v_j^\top U_t v_j \leq 1$, where U_t is defined as $\frac{(u_t I - A_{t-1})^{-2}}{\Phi_{u_{t-1}}(A_{t-1}) - \Phi_{u_t}(A_{t-1})} + (u_t I - A_{t-1})^{-1}$. By using a one-sided argument as in [Sri10, Wea13], one can guarantee that such an index always exists.

We can formulate this problem as a *minimum inner product search* problem, where we first preprocess all vectors $v_i \otimes v_i$, then at each iteration we form the matrix U_t and use it as a query to the Min-IP data structure. Use the TensorSparse transform to preprocess vectors in V and the Min-IP data structure we've developed as in Theorem 2.2 to solve this problem efficiently. We summarize our result below, which significantly improves a running time of $O(nmd^\omega)$ obtained in [Wea13].

Theorem 2.6 (Informal of Theorem 8.5)**.** *Let $\tau, c \in (0,1)$ and $N \in \mathbb{N}_+$, if $V := \{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m v_i v_i^\top = I$. Then for any $n < m$, there exists a randomized algorithm (success with high probability) that takes time $\mathcal{T}$ to find a set S ($|S| = n$) such that*

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

Further, we have that, if $c \in (\tau, \frac{400\tau}{399+\tau})$, then

$$\mathcal{T} = \widetilde{O}((m^{1.01} + \mathrm{nnz}(V))\cdot d^2 + n\cdot(m^{0.01}d^2 + d^\omega)).$$

Moreover, if one picks $\tau = 0.9$ and $c = 0.9$, then $\frac{1}{c} \approx 1.11$.

Experimental Design via Regret Minimization. In the work by Allen-Zhu, Li, Singh and Wang [AZLSW20], they introduced a unified framework to solve the experimental design problem, which concerns the following problem: given $V := \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$, the goal is to select n of them from V so that the *statistical efficiency* is maximized when regressed on the n selected points. To approach this problem, they first use variants of mirror descent algorithm to solve a continuous relaxation, then round the solution via a regret minimization framework. More concretely, let $\pi \in [0,1]^m$ be a fractional solution satisfying $\|\pi\|_1 \leq n$, the goal is to round π into a vector $s \in \{0,1\}^m$ such that $\sum_{i=1}^m s_i \cdot v_i v_i^\top \succeq (1-\varepsilon) \sum_{i=1}^m \pi_i \cdot v_i v_i^\top$ and $\|s\|_1 = n$. To implement the rounding, they developed a swapping algorithm, which involves initiating a set S_0 randomly of cardinality n, then at iteration t, we construct the following matrix: $A_t = (c_t I - \alpha \sum_{i \in S_{t-1}} v_i v_i^\top)^{-2}$ where $c_t \in \mathbb{R}$ is the constant such that $A_t \succ 0$ for $\alpha = \sqrt{d}/\varepsilon$. Then we are to find two indices:

$$i_t = \arg\min_{i \in S_{t-1}, 2\alpha \langle A_t^{1/2}, v_i v_i^\top \rangle < 1} B^-(v_i) := \frac{\langle A_t, v_i v_i^\top \rangle}{1 - 2\alpha \langle A_t^{1/2}, v_i v_i^\top \rangle},$$
$$j_t = \arg\max_{j \in [m] \setminus S_{t-1}} B^+(v_j) := \frac{\langle A_t, v_j v_j^\top \rangle}{1 + 2\alpha \langle A_t^{1/2}, v_j v_j^\top \rangle}.$$

Then we set $S_t = S_{t-1} \setminus \{i_t\} \cup \{j_t\}$.

In order to perform inner product search with this framework, it is necessary to exhibit an upper bound on $B^-(v_i)$ and a lower bound on $B^+(v_j)$, which is also critical for the correctness proof. By an averaging argument, [AZLSW20] showed that one will always have $B^-(v_i) \leq \frac{1-\varepsilon}{n}$ and $B^+(v_j) \geq \frac{1}{n}$. This enables us to reduce this problem into an inner product search: in the set S_{t-1}, we perform Min-IP search, while in set $\overline{S}_{t-1}$, we perform Max-IP search. However, since both Max-IP and Min-IP are *approximate* data structures and the gap between B^- and B^+ is small, we have to require an extra condition on the approximation ratio $c_{\text{Max-IP}} \in (0,1)$ and $c_{\text{Min-IP}} > 1$. We discuss this issue in more details in Section 9.

On the other hand, in the scheme where either $n \ll m - n$ or $n \gg m - n$, a better approach is to use use Min-IP/Max-IP separately on the larger set and simply perform linear scan on the smaller set. This would circumvent the restriction on the relationship between approximation factors and ε.

We present an "unbalanced" version of result here, where $n \gg m - n$. Using our technique, we significantly improve the running time obtained by [AZLSW20][3], which is $\widetilde{O}(\varepsilon^{-1} nmd^2)$. Note that in the running time we obtain, we do not have a dependence on product mn, which is sub-optimal when n is large.

Theorem 2.7 (Informal version of Theorem 9.16)**.** *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 $\tau \in (0,1)$ and $c \in (\frac{1}{\gamma - 1}, 1)$. If $n \geq \frac{6d/\varepsilon^2}{\gamma - 1 - 1/c}$, then there exists a randomized algorithm (success with high probability) that takes time $\mathcal{T}$ to find such S. Furthermore,if $c \in (\tau, \frac{400\tau}{399+\tau})$, then

$$\mathcal{T} = \widetilde{O}((m-n)d^2 + n^{1.01} d^2 + \varepsilon^{-2} n (n^{0.01} d^2 + d^\omega)).$$

[3]Note that their result is interesting only when $\varepsilon \sim O(1)$, since if $\varepsilon \sim \frac{1}{\log d}$, one could use importance sampling to achieve a faster running time.

Roadmap. In Section 3, we give a preliminary on notations, definitions, some useful facts and probabilistic tools used in this paper. In Section 4, we introduce the efficient sketchings for tensors that is robust to adaptive adversary. In Section 5, we present the our efficient data structure to solve the Max-IP problem with robustness to adaptive adversary. In Section 6, we provide the efficient Min-IP data structure. In Section 7, we introduce our algorithm for generalized sum of rank-1 matrices sparsification. In Section 8, we present our algorithmic result for one-sided Kadison-Singer problem with approximate guarantee. In Section 9, we utilize our algorithmic framework on the rounding up of experimental design problem.

3 Preliminaries

This section gives some preliminary background definitions and facts.

- In Section 3.1, we introduce notations used across this paper.
- In Section 3.2, we recall the definition of Johnson-Lindenstrauss transform.
- In Section 3.3, we record some useful facts for our later proof.
- In Section 3.4, we introduce the probability tools used in the paper.

3.1 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 $\mathcal{T}_{\mathrm{mat}}(a, b, c) = O(\mathcal{T}_{\mathrm{mat}}(a, c, b)) = O(\mathcal{T}_{\mathrm{mat}}(b, a, c))$.

For real symmetric matrices A and B of the same size, we use $A \approx_\varepsilon B$ if $(1-\varepsilon)B \preceq A \preceq (1+\varepsilon)B$.

3.2 Johnson-Lindenstrauss Transform

We consider the well-known Johnson-Lindenstrauss transform [JL84], throughout this paper, we will make use of various sketching matrices that satisfy the Johnson-Lindenstrauss lemma. We introduce the following definition.

Definition 3.1 (Johnson-Lindenstrauss transform (JLT))**.** Let $\{x_1, \ldots, x_m\} \in (\mathbb{R}^d)^m$, we say a distribution Π over $s \times d$ matrices is a (m, ε, δ)-JLT if for any $S \sim \Pi$, we have

$$\Pr[\|S(x_i - x_j)\|_2^2 \leq (1 \pm \varepsilon)\|x_i - x_j\|_2^2] \geq 1 - \delta, \quad \forall (i, j) \in [m] \times m.$$

We remark that in order to obtain this property for all m^2 pairs of point, it suffices to obtain the following guarantee for any fixed point $x \in \mathbb{R}^d$:

$$\Pr[\|Sx\|_2^2 \leq (1 \pm \varepsilon)\|x\|_2^2] \geq 1 - \delta,$$

then union bound over all m^2 pairs of points, we are done.

It is a common practice to use JLT to reduce the dimension of input points for similarity search data structures, then feed into low dimensional vectors into the data structures. We will later show this idea is very powerful when designing task-specific data structures.

3.3 Useful Facts

We list and prove some useful facts regarding matrices.

Fact 3.2. *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 $Z^{1/2}$. □

Fact 3.3 (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 3.4. *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. □

Fact 3.5 (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*

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

3.4 Probability Tools

In this section, we present some probability tools.

We start with the standard 2-stable Gaussian distribution. We refer the readers to [DIIM04] for more details.

Fact 3.6 (Standard Gaussian is 2-stable)**.** *Let $Z, X_1, X_2, \ldots, X_k \sim \mathcal{N}(0,1)$ and $v \in \mathbb{R}^k$, then $\sum_{i=1}^k v_i X_i$ and $\|v\|_2 \cdot Z$ have the same distribution.*

Next, we present a concentration and anti-concentration bound for Gaussian distribution.

Fact 3.7 (Gaussian concentration bound)**.** *Let $X \sim \mathcal{N}(0,1)$ and $t > 0$, then we have*

Part 1 Concentration. $\Pr[|X| \geq t] \leq 2\exp(-t^2/2)/t$.

Part 2 Anti-Concentration. *There exists a constant $B > 0$ such that*

$$\Pr[|X| \geq t] \geq 2B \cdot \exp(-t^2/2)/\max\{1, t\}.$$

Definition 3.8. Let X be a random variable, we use $\|X\|_{L_q}$ to denote $(\mathbb{E}[|X|^q])^{1/q}$. By Minkowski's inequality, $\|\cdot\|_{L_q}$ is a norm when $q \geq 1$.

Lemma 3.9 (Hanson-Wright inequality [HW87])**.** *For σ_1, σ_n independent Rademachers and $A \in \mathbb{R}^{n \times n}$, for all $q \geq 1$,*

$$\|\sigma^\top A\sigma - \mathbb{E}[\sigma^\top A\sigma]\|_{L_q} \leq O(1) \cdot (\sqrt{q} \cdot \|A\|_F + q \cdot \|A\|).$$

Lemma 3.10. *For Y distributed as $\mathsf{Binomial}(N, \alpha)$ for integer $N \geq 1$ and $\alpha \in (0,1)$, let $1 \leq p \leq N$ and define $B := p/(\alpha N)$. Then*

$$\|Y\|_{L_p} \leq \begin{cases} \frac{p}{\log B}, & \text{if } B \geq e \\ \frac{p}{B}, & \text{if } B < e. \end{cases}$$

4 Efficient and Adaptive Adversary Robust Sketchings for Tensors

In this section, we introduce several primitives that perform Johnson-Lindenstrauss transforms efficiently on outer product of vectors, or equivalently, the tensor product on vectors. We will exploit these primitives to design fast Johnson-Lindenstrauss transforms for the matrices in the form of $vv^\top$, then feed in the sketched vectors into our Max-IP and Min-IP data structures. This yields an improved preprocess and query time of our data structures, which is key to improve the overall running time of several algorithms.

Throughout this section, we will use m to denote the target dimension of sketching, n to denote the number of points we want to preserve their pair-wise distances, d to denote the dimension of original data points, and s to denote the sparsity for each column of a sketching matrix.

This section is organized as below.

- In Section 4.1, we introduce the TensorSRHT transform for efficient tensor product.
- In Section 4.2, we present the TensorSparse transform for tensor product in input-sparsity time.
- In Section 4.3, we provide tools with theoretical gaurantees for the sketchings to handle adaptive adversary.

4.1 TensorSRHT: Compute Tensor Product via FFT

We first introduce the primitive of TensorSRHT transform ([AKK+20, SWYZ21]), which gives high probability guarantee and nearly-linear time in order to evaluate a tensor product.

Definition 4.1. We define the TensorSRHT $S : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^m$ as $S = \frac{1}{\sqrt{m}} P \cdot (HD_1 \times HD_2)$, where each row of $P \in \{0,1\}^{m \times d^2}$ contains only one 1 at a random coordinate, one can view P as a sampling matrix. H is a $d \times d$ Hadamard matrix, and D_1, D_2 are two $d \times d$ independent diagonal matrices with diagonals that are each independently set to be a Rademacher random variable (uniform in $\{-1, 1\}$).

As the name suggests, one can utilize the structure of Hadamard matrix and use Fast Fourier Transform (FFT) to compute the tensor product of two vectors: $S(x \otimes y)$ can be computed in time $O(d \log d + m)$.

Next, we show that given any fixed pair of vectors $x, y \in \mathbb{R}^d$, S preserves the inner product with high probability:

Lemma 4.2 (Theorem 2 of [AKK+20])**.** *Let $x, y \in \mathbb{R}^d$ be any fixed pair of vectors. Let $\varepsilon \in (0,1)$ be precision parameter and $\delta \in (0,1)$ be success probability. Let $S \in \mathbb{R}^{m \times d^2}$ be a* TensorSRHT *transform matrix (Def. 4.1). Suppose $m = \Omega(\frac{\log^3(1/\varepsilon\delta)}{\varepsilon^2})$, then we have S is a $(1, \varepsilon, \delta)$-JLT (Def. 3.1).*

For our purpose, it suffices to preserve such inner products for $\Theta(n^2)$ pairs, hence by a union bound, we shall use a TensorSRHT matrix of size $\Theta(\frac{\log^3(n/\varepsilon\delta)}{\varepsilon^2})$. This leads to the following result:

Lemma 4.3. *Let $\{x_1, \ldots, x_n\} \in (\mathbb{R}^{d^2})^n$. Let $\varepsilon \in (0,1)$ be precision parameter and $\delta \in (0,1)$ be success probability. Let $S \in \mathbb{R}^{m \times d^2}$ be a* TensorSRHT *transform matrix (Def. 4.1). Suppose $m = \Omega(\varepsilon^{-2} \log^3(n/\varepsilon\delta))$, then we have S is an (n, ε, δ)-JLT (Def. 3.1).*

Moreover, if $x = u \otimes v$ for some $u, v \in \mathbb{R}^d$, then Sx can be computed in time $O(d \log d + m)$.

Proof. For the dimension of sketching matrix, it is a consequence of union bounding over all n^2 pairs of vectors. For the running time, it follows directly from the structure of TensorSRHT. □

One important guarantee given by Lemma 4.3 is it preserves the inner product of any pair of vectors with proper dimensions, but facilitate faster computation for tensor-type computation in the form of $S(u \otimes v)$. For a vector $x \in \mathbb{R}^{d^2}$ that does not have such tensor structure, we can still compute Sx use standard matrix-vector product in time $O(sd^2) = \widetilde{O}(d^2)$. This suffices for our application.

Additionally, we prove a result regarding the Frobenius norm of the TensorSRHT matrix.

Lemma 4.4. *Let $S \in \mathbb{R}^{m \times d^2}$ be a* TensorSRHT *matrix (Def. 4.1), then we have*

$$\|S\|_F \leq d.$$

Proof. We note that a Hadamard matrix has orthonormal columns, and since D_i is a diagonal matrix with $\{\pm 1\}$ on its diagonal, we have that HD_i is also has orthonormal columns for $i \in \{1, 2\}$. Hence, we know that $\|HD_i\|_F = \sqrt{d}$. Moreover, we note that $\|HD_1 \times HD_2\|_F \leq \|HD_1\|_F \|HD_2\|_F = d$ since $\times$ is the tensor product of two matrices. Finally, note that P is a sampling matrix, it samples s rows from $HD_1 \times HD_2$ with replacement, hence $\|P(HD_1 \times HD_2)\|_F \leq \|P\|_F \|HD_1 \times HD_2\|_F \leq \sqrt{s}d$. The result follows since we need to scale the matrix $P(HD_1 \times HD_2)$ by $\frac{1}{\sqrt{s}}$. □

4.2 TensorSparse: Efficient Tensor Product in Input-Sparsity Time

We recall the sparse embedding matrix [DKS10, KN10, KN14, CJN18].

Definition 4.5. Let $h : [d] \times [s] \to [m/s]$ be a random $O(\log 1/\delta)$-wise independent hash function and $\sigma : [d] \times [s] \to \{\pm 1\}$ be $O(\log 1/\delta)$-wise independent. Then $R \in \mathbb{R}^{m \times d}$ is a sparse embedding matrix with sparsity parameter s if we set $R_{(j-1)m/s+h(i,j),i} = \sigma(i,j)/\sqrt{s}$ for all $(i,j) \in [d] \times [s]$ and all other entries to 0.

Alternatively, we can define the following:

$$R_{r,i} = \exists k \in [s] : \sigma(i,k)/\sqrt{s} \cdot \mathbf{1}[h(i,k) + (k-1)m/s = r]$$

We extend the construction of sparse embedding to handle tensor product of vectors, specifically, our goal is to design a sparse matrix that is similar to Def. 4.5, so that we can enjoy certain nice properties, such as it is a $(1, \varepsilon, \delta)$-JLT with $m = O(\varepsilon^{-2} \log(1/\delta))$, this again enables us to union bound over n points.

Definition 4.6 (TensorSparse)**.** Let $h_1, h_2 : [d] \times [s] \to [m/s]$ be $O(\log 1/\delta)$-wise independent hash functions and let $\sigma_1, \sigma_2 : [d] \times [s] \to \{\pm 1\}$ be $O(\log 1/\delta)$-wise independent random sign functions. Then, the degree two tensor sparse transform, $R : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^m$ is defined as follows:

$$R_{r,(i,j)} = \exists k \in [s] : \sigma_1(i,k)\sigma_2(j,k)/\sqrt{s} \cdot \mathbf{1}[((h_1(i,k) + h_2(j,k)) \bmod m/s) + (k-1)m/s = r]$$

We will show that for any fixed unit vector $x \in \mathbb{R}^{d^2}$, Rx preserves the length of x with good probability. To do so, we first exhibit some properties of our sketch.

Lemma 4.7. *The degree two* TensorSparse *transform (Def. 4.6) has the following property. We define $\delta_{r,(i,j)}$ as the Bernoulli random variable on whether the entry $R_{r,(i,j)}$ is non-zero or not. Then*

1. *Each column has exactly s non-zero entries.*
2. *For all $r \in [m]$ and $(i,j) \in [d] \times [d]$, $\mathbb{E}[\delta_{r,(i,j)}] = s/m$.*
3. *The $\delta_{r,(i,j)}$'s are negatively-correlated:*

$$\forall T \subset [m] \times [d] \times [d] \text{ and } |T| \leq \Theta(\log(1/\delta)), \quad \mathbb{E}\Big[\prod_{r,(i,j) \in T} \delta_{r,(i,j)}\Big] \leq \prod_{r,(i,j) \in T} \mathbb{E}[\delta_{r,(i,j)}] = (\frac{s}{m})^{|T|}.$$

Proof. We prove three parts separately.

Part 1. To see each column has exactly s non-zero entries, we partition each column into s blocks, where each block contains m/s entries and then show that each block has exactly 1 non-zero entry. Fix the block to be the k-th block and consider the (i,j)-th column, then we are looking at the values of hash functions $(h_1(i,k) + h_2(j,k)) \bmod m/s$, since both h_1 and h_2 have their ranges being $[m/s]$, this means $(h_1(i,k) + h_2(j,k)) \bmod m/s$ must have its value being in the range of $[m/s]$, and its value corresponding to the entry that is non-zero.

Part 2. We will again use the block-partition view and consider the k-th block of (i,j)-th column. For each index r, the probability that it is non-zero is equal to the probability that $(h_1(i,k) + h_2(j,k)) \bmod m/s = r - (k-1)m/s$. We first observe that if we are using a single 3-wise independent hashing function, then this probability is naturally s/m. Here, the hashing function we are considering is $H(i,j,k) := h_1(i,k) + h_2(j,k) \bmod m/s$, it is well-known that H is also $\Theta(\log(1/\delta))$-wise independent ([CW79, PT12]). We hence conclude that $\Pr[\delta_{r,(i,j)} = 1] = \frac{s}{m}$ and therefore, $\mathbb{E}[\delta_{r,(i,j)}] = \frac{s}{m}$.

Part 3. To see the negative correlation, we let $t = |T|$, and we denote the elements in T as $(r_1, l_1), \ldots, (r_t, l_t)$. We define the following indicator random variable: $\mathbf{1}[\exists (r_i, l_i), (r_j, l_j) \in T$ s.t. $r_i \neq r_j$ belong to the same block and $l_i = l_j]$.

Note that if such event happens, then $\mathbb{E}[\prod_{(r,l)\in T} \delta_{r,l}] = 0$ since we can write it as

$$\begin{aligned}\mathbb{E}[\prod_{(r,l)\in T} \delta_{r,l}] &= \Pr[\bigwedge_{r,l\in T} \delta_{r,l} = 1] \\ &= \Pr[\delta_{r_1,l_1} = 1 \wedge \delta_{r_2,l_2} = 1] \cdot \Pr[\bigwedge_{(r,l)\in T, r\neq r_1,r_2, l\neq l_1,l_2} \delta_{r,l} = 1 \mid \delta_{r_1,l_1} = 1 \wedge \delta_{r_2,l_2} = 1].\end{aligned}$$

When the above event happens, then we are considering the case that $r_1 \neq r_2$ but they belong to the same block, and the column is the same. By construction, for each column, there is exactly one non-zero entry. Hence, $\Pr[\delta_{r_1,l_1} = 1 \wedge \delta_{r_2,l_2} = 1] = 0$, and we conclude the expectation is 0.

Suppose the above event does not happen, then we will make use the fact that our hashing function H is $\Theta(\log(1/\delta))$-wise independent, and $\delta_{r,l} = 1$ is equivalent to for some $k \in [s]$, we have $H(l, k) = r$. The above event does not happen is equivalent to

$$\begin{aligned}\Pr[\bigwedge_{(r,l)\in T} \exists k \in [s], H(l,k) = r] &= \prod_{(r,l)\in T} \Pr[\exists k \in [s], H(l,k) = r] \\ &= \prod_{(r,l)\in T} \Pr[\delta_{r,l} = 1] \\ &= \prod_{(r,l)\in T} \mathbb{E}[\delta_{r,l}],\end{aligned}$$

where the first step is due to H is $\Theta(\log(1/\delta))$-wise independence. Therefore, we conclude that the random variables $\delta_{r,l}$'s are negatively correlated. □

Remark 4.8. We note that we only require our hashing function H and sign function σ to be $\Theta(\log(1/\delta))$-wise independent, since in our later proofs, we will only consider the q-th power of an expression Z which involves the term $\prod_{(r,l)\in T} \delta_{r,l}$ for $|T| \leq q$. Thus, the expectation of Z are term-by-term dominated by the case that all $\delta_{r,l}$ are i.i.d. Bernoulli with expectation s/m. This justifies our later use of Lemma 3.10 and Hanson-Wright inequality.

We will adapt an analysis from [CJN18] to conclude that TensorSparse is a JLT:

Lemma 4.9. *If R is a* TensorSparse *matrix as defined in Def. 4.6, with target dimension $m \geq \Omega(\log(1/\delta)/\varepsilon^2)$ and sparsity parameter $s = \varepsilon m$, then*

$$\Pr[|\|Rx\|_2^2 - 1| > \varepsilon] \leq \delta.$$

Proof. We first observe that

$$\begin{aligned}\|Rx\|_2^2 &= \frac{1}{s}\sum_{r=1}^{m}\sum_{i,j=1}^{d^2} \delta_{r,i}\delta_{r,j}\sigma_{r,i}\sigma_{r,j}x_i x_j \\ &= \frac{1}{s}\sum_{r=1}^{m}\sum_{i=1}^{d^2} \delta_{r,i}x_i^2 + \frac{1}{s}\sum_{r=1}^{m}\sum_{i\neq j}^{d^2} \delta_{r,i}\delta_{r,j}\sigma_{r,i}\sigma_{r,j}x_i x_j,\end{aligned}$$

for the first term (diagonal term), we have

$$\begin{aligned}\frac{1}{s}\sum_{r=1}^{m}\sum_{i=1}^{d^2}\delta_{r,i}x_i^2 &= \sum_{i=1}^{d^2}x_i^2(\frac{1}{s}\sum_{r=1}^{m}\delta_{r,i})\\ &= \|x\|_2^2\\ &= 1,\end{aligned}$$

where the second step follows from the fact that each column of R has exactly s non-zero entries. We define the intermediate variable $Z := \|Rx\|_2^2 - 1$, which as shown by proceeding calculations, captures the off-diagonal term. Consider the following terms: we first define $A_{x,\delta}$ which is a block diagonal matrix with m blocks, where the k-th block is defined as $\frac{1}{s}x^{(k)}(x^{(k)})^\top$ but with the diagonal zeroed out, with $(x^{(k)})_i = \delta_{k,i}x_i$. Note that by construction, $A_{x,\delta} \in \mathbb{R}^{md^2\times md^2}$. We further define the following length md^2 vector $\sigma \in \mathbb{R}^{md^2}$, where $\sigma_{r,i}$ is the sign generated for the entry (r,i) of R.

It is not hard to see that $Z = \frac{1}{s}\sum_{r=1}^{m}\sum_{i\neq j}^{d^2}\delta_{r,i}\delta_{r,j}\sigma_{r,i}\sigma_{r,j}x_ix_j = \sigma^\top A_{x,\delta}\sigma$. Let $\|X\|_{L_q} := (\mathbb{E}[|X|^q])^{1/q}$. Since σ is a vector with each entry being independent Rademacher random variable, by Hanson-Wright inequality, we have

$$\begin{aligned}\|\sigma^\top A_{x,\delta}\sigma\|_{L_q} &\le \|\sqrt{q}\cdot\|A_{x,\delta}\|_F + q\cdot\|A_{x,\delta}\|\|_{L_q}\\ &\le \sqrt{q}\cdot\|\|A_{x,\delta}\|_F\|_{L_q} + q\cdot\|\|A_{x,\delta}\|\|_{L_q},\end{aligned}$$

since $A_{x,\delta}$ is block diagonal, its spectral norm is the largest spectral norm of any block. Note that the spectral norm of k-th block is

$$\begin{aligned}\|\frac{1}{s}\cdot x^{(k)}(x^{(k)})^\top\| &\le \frac{1}{s}\cdot\|x^{(k)}\|_2^2\\ &\le \frac{1}{s},\end{aligned}$$

where the first step is the sub-multiplicativity of spectral norm and the spectral norm of a vector is its ℓ_2 norm, and the second line follows from $\|x^{(k)}\|_2 \le \|x\|_2 = 1$.

Next, we define $Q_{i,j} = \sum_{r=1}^{m}\delta_{r,i}\delta_{r,j}$, so

$$\begin{aligned}\|A_{x,\delta}\|_F^2 &= \frac{1}{s^2}\sum_{r=1}^{m}\sum_{i\neq j}^{d^2}\delta_{r,i}\delta_{r,j}x_i^2x_j^2\\ &= \frac{1}{s^2}\sum_{i\neq j}^{d^2}Q_{i,j}x_i^2x_j^2.\end{aligned}$$

Recall that for any column i of R, there exists exactly s non-zero entries, so we suppose $\delta_{r_1,i} = \ldots = \delta_{r_s,i} = 1$ for distinct r_t and write $Q_{i,j} = \sum_{t=1}^{s}Y_t$, where Y_t denotes the indicator random variable for the event that $\delta_{r_t,j} = 1$. By the conclusion of Lemma 4.7, we may well assume Y_t's are independent, in which case $Q_{i,j}$ is distributed as $\mathsf{Binomial}(s, s/m)$, by Lemma 3.10, we know that $\|Q_{i,j}\|_{L_{q/2}} \le q/2$. Thus,

$$\begin{aligned}\|\|A_{x,\delta}\|_F\|_{L_q} &= \|\|A_{x,\delta}\|_F^2\|_{L_{q/2}}^{1/2}\\ &= \|\frac{1}{s^2}\sum_{i\neq j}x_i^2x_j^2Q_{i,j}\|_{L_{q/2}}^{1/2}\end{aligned}$$

$$\leq \frac{1}{s}(\sum_{i\neq j} x_i^2 x_j^2 \|Q_{i,j}\|_{L_{q/2}})^{1/2}$$
$$\leq O(\frac{\sqrt{q}}{s}).$$

Put things together, we have

$$\|\sigma^\top A_{x,\delta}\sigma\|_{L_q} \leq O(\frac{q}{s}). \tag{1}$$

Set $q = \Theta(\log(1/\delta)) = \Theta(s^2/m)$, we have $\|Z\|_{L_q} \leq O(\frac{s}{m})$, then by Markov inequality, we have

$$\Pr[|\|Rx\|_2^2 - 1| > \varepsilon] = \Pr[|\sigma^\top A_{x,\delta}\sigma| > \varepsilon] < \varepsilon^{-q} \cdot C^q(m^{-q/2} + s^{-q}) < \delta,$$

as desired. □

Note that our construction resembles the TensorSketch matrix [Pag13, ANW14], more specifically, we can view our tensor sparse embedding as s distinct TensorSketch matrices, each with dimension $m/s \times d^2$. Hence, to compute the tensor product between two vectors, we can run the TensorSketch algorithm for s blocks, yielding an overall running time of $O(s \cdot (\mathrm{nnz}(x) + \mathrm{nnz}(y)) + m\log(m/s))$ for computing $S(x \otimes y)$.

We summarize the JLT result and efficient computation of tensor in the following theorem:

Theorem 4.10 (Formal version of Theorem 2.3)**.** *Let $\{x_1, \ldots, x_n\} \in (\mathbb{R}^{d^2})^n$. Let $\varepsilon \in (0,1)$ be precision parameter and $\delta \in (0,1)$ be success probability. Let $R \in \mathbb{R}^{m \times d^2}$ be a* TensorSparse *matrix (Def. 4.6). Suppose $m = \Omega(\varepsilon^{-2}\log(n/\delta))$ and $s = \varepsilon m$ be the sparsity parameter, then we have R is an (n, ε, δ)-JLT (Def. 3.1).*

Moreover, if $x = u \otimes v$ for some $u, v \in \mathbb{R}^d$, then Rx can be computed in time $O(s \cdot (\mathrm{nnz}(u) + \mathrm{nnz}(v)) + m\log(m/s))$.

Proof. The JLT result is by apply union bound over all n^2 pairs of points using Lemma 4.9. The running time is by using the TensorSketch algorithm for s blocks. □

For further applications, we prove a simple result regarding the Frobenius norm of R.

Lemma 4.11. *Let $R \in \mathbb{R}^{m \times d^2}$ be a* TensorSparse *matrix (Def. 4.6), then we have*

$$\|R\|_F = d.$$

Proof. We observe that each column of R has exactly s non-zero entries, each has magnitude $\frac{1}{\sqrt{s}}$, hence each column is a unit length vector. There are d^2 columns in total, yielding a Frobenius norm of d. □

4.3 Robust Sketches Against Adaptive Adversary

We note that the above discussion only applies when we consider an *independent* set of points, i.e., all points we want to preserve using TensorSRHT or TensorSparse are picked oblivious with respect to the randomness of the sketch. However, this is no longer the case for our application — specifically, the query we send for iteration $t+1$ is *dependent* on the answer we receive at iteration t.

One idea is to require a sketching matrix that preserves the length of *all* vectors in a subspace. Unfortunately, this will result in a sketching dimension of roughly $\Theta(d^2/\varepsilon^2)$, which essentially diminishes the necessity of using sketching. To address this problem, we exploit the following idea:

we use a number of independent sketches of small dimension, and we show that with high probability, a good fraction of them will do well on a (potentially) adversary query. We will show that the dimension-saving by using lower-dimensional sketching matrices will have to be paid back by the number of sketches required. However, this has one distinctive advantage for our applications: we will then operate our ANN or AFN data structures on much lower dimensions, hence the preprocessing time and query time can be significantly improved.

We prove the following lemma:

Lemma 4.12. *Let $V := \{v_1, \ldots, v_n\} \in (\mathbb{R}^d)^n$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. Furthermore, let $\{S_i\}_{i=1}^k \in \mathbb{R}^{m \times d}$ for $k \geq \Omega((d + \log(1/\delta))\log(nd))$ such that each S_i is an independent $(n+1, \varepsilon, 0.99)$-JLT matrix (Def. 3.1) with $\|S_i\|_F \leq d$. Then we have*

$$\forall q \in \mathbb{S}^{d-1}, \forall v \in V, \sum_{i=1}^{k} \mathbf{1}[\|S_i(q-v)\|_2^2 \leq (1 \pm O(\varepsilon))\|q-v\|_2^2 + \alpha] \geq 0.95k$$

with probability at least $1-\delta$ and $\alpha \leq O(\frac{1}{(nd)^9})$.

Proof. We will prove via a standard γ-net argument. Let N be a γ-net of $\mathbb{S}^{d-1}$ with $\gamma = \frac{c}{(nd)^{10}}$ for some small enough constant c, and it is not hard to see that $|N| \leq (nd)^{O(d)}$. Let $u \in N$, define the following event:

$$W_i(u) = \|S_i u\|_2^2 \leq (1 + O(\varepsilon)) \text{ and } \forall v_i, v_j \in V, |u^\top S_i^\top S_i (v_i - v_j) - u^\top (v_i - v_j)| \leq O(\varepsilon)\|v_i - v_j\|_2,$$

i.e., the length of u is preserved by S_i and for any pair of points in V, the inner product is also preserved by S_i. We note that we only need this property to hold with respect to the set of points $V \cup \{u\}$, since S_i is a $(n+1, \varepsilon, 0.99)$-JLT, we know this event holds with probability at least 0.99.

By an application of Hoeffding's inequality on the random variables $\sum_{i=1}^k W_i(u)$, we have that

$$\Pr[\sum_{i=1}^{k} W_i(u) \leq 0.97k] \leq \exp(-2k),$$

we then union bound over all points in N:

$$\begin{aligned}
\Pr[\forall u \in N, \sum_{i=1}^{k} W_i(u) \leq 0.97k] &\leq \exp(-2k) \cdot (nd)^{O(d)} \\
&= (\frac{1}{nd})^{O(d)} \cdot (nd)^{O(d)} \cdot \exp(-\log(1/\delta)\log(nd)) \\
&\leq \delta/4.
\end{aligned}$$

We will condition on this event happen throughout the rest of the proof. To extend this bound from all points in N to the entire unit sphere, consider any $q \in \mathbb{S}^{d-1}$ and pick a net point $u \in N$ such that $\|q-u\|_2 \leq \gamma$. Let $i \in [m]$ be the index such that $W_i(u)$ happens. We shall bound the term $\|S_i(q-v)\|_2$ for $v \in V$:

$$\begin{aligned}
\|S_i(q-v)\|_2 &\leq \|S_i(q-u)\|_2 + \|S_i(u-v)\|_2 \\
&\leq d \cdot \gamma + (1 \pm O(\varepsilon))\|u-v\|_2 \\
&\leq d \cdot \gamma + (1 \pm O(\varepsilon))(\|q-v\|_2 - \gamma) \\
&= (1 \pm O(\varepsilon))\|q-v\|_2 + (d - (1 \pm O(\varepsilon)))\gamma \\
&\leq (1 \pm O(\varepsilon))\|q-v\|_2 + \alpha.
\end{aligned}$$

The conclusion of the lemma follows. □

Remark 4.13. We note that by using the γ-net argument, we get a weaker conclusion compared to standard Johnson-Lindenstrauss lemma, namely, we preserve the distance with $(1 \pm O(\varepsilon))$ relative error and α additive error. Fortunately, the magnitude of α is small enough so that it won't affect the quality of our downstream task too much.

As an example, consider the following adversary robust ANN: we use k different independent data structures where each one has an independent JLT matrix S_i. At each query point q, we shall sample $\Theta(\log m)$ data structures and output the one with the best quality.

Now, note that

$$\begin{aligned} \|S_i q - S_i \widehat{p}\|_2 &\geq (1 - O(\varepsilon))\|q - \widehat{p}\|_2 - \alpha, \\ \|q - \widehat{p}\|_2 &\leq \frac{\|S_i q - S_i \widehat{p}\|_2 + \alpha}{1 - O(\varepsilon)}, \end{aligned}$$

on the other hand,

$$\begin{aligned} \|S_i q - S_i \widehat{p}\|_2 &\leq \overline{c} \cdot \|S_i q - S_i p\|_2 \\ &\leq \overline{c} \cdot ((1 + O(\varepsilon))\|q - p\|_2 + \alpha) \\ &\leq \overline{c} \cdot (1 + O(\varepsilon))\|q - p\|_2 + \overline{c} \cdot \alpha, \end{aligned}$$

put things together,

$$\|q - \widehat{p}\|_2 \leq \overline{c} \cdot (1 + O(\varepsilon))\|q - p\|_2 + (1 + \overline{c})(1 + O(\varepsilon)) \cdot \alpha.$$

Using this estimate for the Max-IP task, we incur a $(1+\overline{c})(1+O(\varepsilon))\cdot\alpha$ additive error as well. However, later we will see that for the task such as BSS, it suffices to require the max inner product to be positive and by a simple scaling argument, the magnitude of the inner product can be controlled.

As a direct consequence, we have the following result with TensorSRHT and TensorSparse:

Corollary 4.14. *Let $V := \{v_1, \ldots, v_n\} \in (\mathbb{R}^d)^n$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. Furthermore, let $\{S_i\}_{i=1}^k \in \mathbb{R}^{m \times d}$ for $k \geq \Omega((d + \log(1/\delta))\log(nd))$ such that each S_i is an independent* TensorSRHT *matrix with $m = \Theta(\frac{\log^3(n/\varepsilon)}{\varepsilon^2})$ rows and $\|S_i\|_F \leq d$. Then we have*

$$\forall q \in \mathbb{S}^{d-1}, \forall v \in V, \sum_{i=1}^k \mathbf{1}[\|S_i(q - v)\|_2^2 \leq (1 \pm O(\varepsilon))\|q - v\|_2^2 + \alpha] \geq 0.95k$$

with probability at least $1 - \delta$ and $\alpha \leq O(\frac{1}{(nd)^9})$.

Proof. The result follows by combining Lemma 4.3 and Lemma 4.12. □

Corollary 4.15. *Let $V := \{v_1, \ldots, v_n\} \in (\mathbb{R}^d)^n$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. Furthermore, let $\{R_i\}_{i=1}^k \in \mathbb{R}^{m \times d}$ for $k \geq \Omega((d + \log(1/\delta))\log(nd))$ such that each R_i is an independent* TensorSparse *matrix with $m = \Theta(\frac{\log n}{\varepsilon^2})$ rows and $\|R_i\|_F = d$. Then we have*

$$\forall q \in \mathbb{S}^{d-1}, \forall v \in V, \sum_{i=1}^k \mathbf{1}[\|S_i(q - v)\|_2^2 \leq (1 \pm O(\varepsilon))\|q - v\|_2^2 + \alpha] \geq 0.95k$$

with probability at least $1 - \delta$ and $\alpha \leq O(\frac{1}{(nd)^9})$.

Proof. The result follows from Theorem 4.10 and Lemma 4.12. □

5 Max-IP Data Structure: Efficient, Low-Dimensional, and Adaptive Adversary Robust

In this section, we design a novel data structure for Max-IP. Our data structure takes advantage of nearest neighbor search data structure and efficient Johnson-Lindenstrauss transform. This combination of data structure and random embedding significantly reduces the query time complexity. We also show how to augment this data structure to be adaptive adversary robust. This is essential for our applications, since the query point we feed into data structure is dependent on the answer we get from data structure at previous iterations. This section is organized as below:

- In Section 5.1, we formally define the Max-IP problem and state its connection with approximate nearest neighbor search.
- In Section 5.2, we introduce the efficient data structure to solve the Max-IP problem.
- In Section 5.3, we present an efficient and robust Max-IP data structure to adaptive queries.

Throughout this section, we will use n to denote the number of data points, and d to denote the dimension of data.

5.1 Solving Max-IP via Approximate Nearest Neighbor Search

We define the exact and approximate Max-IP problem as follows:

Definition 5.1 (Exact Max-IP)**.** Given a data set $Y \subseteq \mathbb{R}^d$, we define Max-IP for a query point $x \in \mathbb{R}^d$ with respect to Y as follows:

$$\text{Max-IP}(x, Y) := \max_{y \in Y} \langle x, y \rangle.$$

In practice, we relax the exact Max-IP to allow approximation. In this way, more randomized algorithms could be introduced for query time speedup.

Definition 5.2 (Approximate Max-IP)**.** Let $c \in (0, 1)$ and $\tau \in (0, 1)$. Given an n-point dataset $Y \subset \mathbb{S}^{d-1}$, the (c, τ)-Max-IP aims at building a data structure that, given a query $x \in \mathbb{S}^{d-1}$ with the promise that $\text{Max-IP}(x, Y) \geq \tau$, the data structure reports a datapoint $z \in Y$ with similarity $\langle x, z \rangle$ greater than $c \cdot \text{Max-IP}(x, Y)$.

The approximate Max-IP has a dual problem: Approximate Nearest Neighbor (ANN). We could solve approximate Max-IP via solving ANN. The ANN is a well-studied problem [AM93, IM98, DIIM04, AINR14, AIL+15, AR15, ALRW17, IW18, AIR18, DIRW19, CCD+20]. We use the standard ANN definition shown as below:

Definition 5.3 (Approximate Nearest Neighbor (ANN))**.** Let $\overline{c} > 1$. Let $r \in (0, 2)$. Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the $(\overline{c}, r)$-Approximate Near Neighbor Search (ANN) aims at developing a data structure that, given a query $q \in \mathbb{S}^{d-1}$ with the promise that there exists a datapoint $p \in P$ with $\|p - q\|_2 \leq r$, the data structure reports a datapoint $p' \in P$ with distance less than $\overline{c} \cdot r$ from q.

Using efficient data structures such as locality-sensitive hashing [IM98, DIIM04, AR15, ALRW17], the query complexity of ANN could be reduced to sublinear in number of data following Theorem 5.4 and Theorem 5.5. Note that here we write $O(1/\sqrt{\log n})$ as $o(1)$.

Theorem 5.4 (Andoni and Razenshteyn [AR15]). *Let $\overline{c} > 1$ and $r \in (0, 2)$. The $(\overline{c}, r)$-ANN on a unit sphere $\mathbb{S}^{d-1}$ can be solved by a data structure with query time $O(d \cdot n^{\rho})$, space $O(n^{1+\rho} + dn)$ and preprocessing time $O(dn^{1+\rho})$, where $\rho = \frac{1}{2\overline{c}^2 - 1} + o(1)$.*

Theorem 5.5 (Andoni, Laarhoven, Razenshteyn and Waingarten [ALRW17]). *Let $\overline{c} > 1$. Let $r \in (0, 2)$. There exists a data structure that solves $(\overline{c}, r)$-ANN on the unit sphere $\mathbb{S}^{d-1}$ with query time $O(d \cdot n^{\rho})$, space $O(n^{1+o(1)} + dn)$ and preprocessing time $O(dn^{1+o(1)})$, where $\rho = \frac{2}{\overline{c}^2} - \frac{1}{\overline{c}^4} + o(1)$.*

5.2 Efficient Max-IP Data Structure

In this section, we show a standard way of solving Max-IP with ANN data structures [AR15, ALRW17, AIR18].

Theorem 5.6. *Let $c \in (0, 1)$ and $\tau \in (0, 1)$. Given a set of n-points $Y \subset \mathbb{S}^{d-1}$ on the sphere, one can build a data structure with preprocessing time $\mathcal{T}_{\mathsf{init}}$ and space $\mathcal{S}_{\mathsf{space}}$ so that for any query $x \in \mathbb{S}^{d-1}$, we take $O(d \cdot n^{\rho})$ query time:*

- *if $\mathsf{Max\text{-}IP}(x, Y) \geq \tau$, then we output a vector in Y which is a (c, τ)-Max-IP with respect to (x, Y) with probability at least 0.9[4], where $\rho := f(c, \tau) + o(1)$.*
- *otherwise, we output fail.*

Further,

- *If $\mathcal{T}_{\mathsf{init}} = O(dn^{1+\rho})$ and $\mathcal{S}_{\mathsf{space}} = O(n^{1+\rho} + dn)$, then $f(c, \tau) = \frac{1-\tau}{1-2c\tau+\tau}$.*
- *If $\mathcal{T}_{\mathsf{init}} = O(dn^{1+o(1)})$ and $\mathcal{S}_{\mathsf{space}} = O(n^{1+o(1)} + dn)$, then $f(c, \tau) = \frac{2(1-\tau)}{(1-c\tau)} - \frac{(1-\tau)^2}{(1-c\tau)^2}$.*

Proof. We start with showing that 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 $r^2 = 2 - 2\tau$ for a $(\overline{c}, r)$-ANN and a (c, τ)-Max-IP on x, Y.

Further, if we have a data structure for $(\overline{c}, r)$-ANN, it automatically becomes a data structure for (c, τ)-Max-IP with parameters $\tau = 1 - 0.5r^2$ and $c = \frac{1-0.5\overline{c}^2 r^2}{1-0.5r^2}$. This implies that

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

Next, we show how to solve (c, τ)-Max-IP by solving $(\overline{c}, r)$-ANN using two different data structures.

Part 1. If we initialize the data structure following Theorem 5.4, we show that the (c, τ)-Max-IP on a unit sphere $\mathbb{S}^{d-1}$ can be solved by solving $(\overline{c}, r)$-ANN with query time $O(d \cdot n^{\rho})$, space $O(n^{1+\rho} + dn)$ and preprocessing time $O(dn^{1+\rho})$, where

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

Thus, $f(c, \tau) = \frac{1-\tau}{1-2c\tau+\tau}$.

Part 2. If we initialize the data structure following Theorem 5.5, we show that the (c, τ)-Max-IP on a unit sphere $\mathbb{S}^{d-1}$ can be solved by solving $(\overline{c}, r)$-ANN with query time $O(d \cdot n^{\rho})$, space $O(n^{1+o(1)} + dn)$ and preprocessing time $O(dn^{1+o(1)})$, where

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

[4] One can boost success probability from constant to δ by using $\log(1/\delta)$ independent data structures.

Thus, $f(c,\tau) = \frac{2(1-\tau)}{(1-c\tau)} - \frac{(1-\tau)^2}{(1-c\tau)^2}$.

□

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

Definition 5.7. 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 $y \in Y$.

$$\varphi(x) = \left[\frac{x^\top}{D_X} \quad 0 \quad \sqrt{1 - \frac{\|x\|_2^2}{D_X^2}}\right]^\top, \psi(y) = \left[\frac{y^\top}{D_Y} \quad \sqrt{1 - \frac{\|y\|_2^2}{D_Y^2}} \quad 0\right]^\top,$$

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

Remark 5.8. 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. Moreover, D_X and D_Y plays a role in controlling the value of parameter τ in approximate Max-IP (see Definition 5.2) for better efficiency.

5.3 Improved Max-IP Data Structure with Robustness to Adaptive Queries

In an iterative process, the queries to our data structure at each step are not independent to each other. Therefore, we could not union bound the failure probability of this adaptive query sequence. Moreover, the existence of adaptive adversarial queries would also challenge the robustness of our algorithm. We tackle this issue through a standard quantization framework.

Definition 5.9. Let $c \in (0,1)$ and $\tau \in (0,1)$. Let $\lambda \geq 0$. Given an n-point dataset $Y \subset \mathbb{S}^{d-1}$, the goal of the (c,τ,λ)-Max-IP is to build a data structure, given a query $x \in \mathbb{S}^{d-1}$ with the promise that $\text{Max-IP}(x, Y) \geq \tau$, it reports a data point $z \in Y$ such that $\langle x, z\rangle \geq c \cdot \text{Max-IP}(x, Y) - \lambda$.

We show a standard way to do quantization for Max-IP. We denote Q as the convex hull of all queries for (c,τ)-Max-IP and its maximum diameter in ℓ_2 distance as D_X. Next, the quantization involves two steps:

Step 1. Preprocessing. We quantize Q into a lattice $\widehat{Q}$ with quantization error λ/d, in this way, each coordinate will be quantized into a multiple of λ/d.

Step 2. Query. Given a query $x \in Q$, we first quantize it to the nearest $\widehat{q} \in \widehat{Q}$ and perform (c,τ)-Max-IP using $\widehat{q}$.

Now, we note that each $\widehat{q} \in \widehat{Q}$ is independent, hence one can union bound the failure probability of adaptive queries. However, this would cause a λ additive error in our inner product estimation. We will later show that how to properly handle this additive error.

One alternative way to view this quantization process is by a λ-net argument: consider generating a λ-net for the space of query points, for each query, we use the vertex in λ-net as the query point to the data structure. Hence, bounding the failure probability of the iterative process can be turned into union bounding all vertices in this λ-net.

Lemma 5.10. *Let $c \in (0,1)$, $\tau \in (0,1)$ and $\lambda \in (0,1)$. Given a set of n-points $Y \subset \mathbb{S}^{d-1}$, one can construct a data structure with $\mathcal{T}_{\mathsf{init}} \cdot \kappa$ preprocessing time and $\mathcal{S}_{\mathsf{space}} \cdot \kappa$ space so that for every $x \in \mathbb{S}^{d-1}$ in an adaptive sequence $X = \{x_1, \ldots, x_T\}$, the query time is $O(dn^\rho \cdot \kappa)$:*

- *If $\mathsf{Max\text{-}IP}(x, Y) \geq \tau$, then we output a vector in Y which is a (c, τ, λ)-$\mathsf{Max\text{-}IP}$ with respect to (x, Y) with probability at least $1 - \delta$, where $\rho = f(c, \tau) + o(1)$.*
- *Otherwise, we output* fail.

where $\kappa := d \log(ndD_X/(\lambda\delta))$ and $\rho \in (0,1)$, and D_X is the diameter of all queries in X.

Further,

- *If $\mathcal{T}_{\mathsf{init}} = O(dn^{1+\rho})$ and $\mathcal{S}_{\mathsf{space}} = O(n^{1+\rho} + dn)$, then $f(c,\tau) = \frac{1-\tau}{1-2c\tau+\tau}$.*
- *If $\mathcal{T}_{\mathsf{init}} = O(dn^{1+o(1)})$ and $\mathcal{S}_{\mathsf{space}} = O(n^{1+o(1)} + dn)$, then $f(c,\tau) = \frac{2(1-\tau)}{(1-c\tau)} - \frac{(1-\tau)^2}{(1-c\tau)^2}$.*

Proof. The failure probability for an adaptive sequence X is equivalent to the probability that at least one query $\widehat{q} \in \widehat{Q}$ fail in solving all κ number of (c,τ)-$\mathsf{Max\text{-}IP}$. We bound this failure probability as

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

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

For the success queries, it introduces a λ error in the inner product. Thus, the results is (c, τ, λ)-$\mathsf{Max\text{-}IP}$.

Then, following Theorem 5.6, we finish the proof. □

We will extend this result by first applying Lemma 4.12 then run the quantization process as we have shown above.

Theorem 5.11 (Formal version of Theorem 2.1)**.** *Let $c \in (0,1)$, $\tau \in (0,1)$, $\lambda \in (0,1)$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. We define the following additional parameters:*

- *$\alpha = O(\frac{1}{(nd)^9})$, the additive error by Lemma 4.12;*
- *$s \leq d$, the dimension of JLT;*
- *$k = O((d + \log(1/\delta)) \log(nd))$, number of independent JLT sketches;*
- *$\kappa = s \log(ns/(\lambda\delta))$;*
- *$\widetilde{\lambda} = O(\sqrt{\frac{1-c\tau}{1-\tau}}) \cdot (\lambda + \alpha)$, the additive error of* $\mathsf{Max\text{-}IP}$.

Let $\mathcal{T}_S(x)$ denote the time of applying S to a vector $x \in \mathbb{R}^d$. Given a set of n-points $Y \subset \mathbb{S}^{d-1}$ on the sphere, one can build a data structure with preprocessing time $\mathcal{T}_{\mathsf{init}}$ and space $\mathcal{S}_{\mathsf{space}} \cdot \kappa \cdot k$ so that for every query $x \in \mathbb{S}^{d-1}$ in an adaptive sequence $X = \{x_1, \ldots, x_T\}$, the query time is $\widetilde{O}(sn^\rho \cdot \kappa + \mathcal{T}_S(x))$:

- *if $\mathsf{Max\text{-}IP}(x, Y) \geq \tau$, then we output a vector in Y which is a $(c, \tau, \widetilde{\lambda})$-$\mathsf{Max\text{-}IP}$ with respect to (x, Y), where $\rho := f(c, \tau) + o(1)$.*
- *otherwise, we output* fail.

Further,

- *If* $\mathcal{T}_{\mathsf{init}} = O(sn^{1+\rho}\cdot\kappa\cdot k)+\mathcal{T}_S(Y)\cdot k$ *and* $\mathcal{S}_{\mathsf{space}} = O(n^{1+\rho}+sn)$*, then* $f(c,\tau) = \frac{(1-\tau)(1+\varepsilon)^2}{2-2c\tau-(1-\tau)(1+\varepsilon)^2}$.
- *If* $\mathcal{T}_{\mathsf{init}} = O(sn^{1+o(1)}\cdot\kappa\cdot k)+\mathcal{T}_S(Y)\cdot k$ *and* $\mathcal{S}_{\mathsf{space}} = O(n^{1+o(1)}+sn)$*, then* $f(c,\tau) = \frac{2(1-\tau)(1+\varepsilon)^2}{1-c\tau} - \frac{(1-\tau)^2(1+\varepsilon)^4}{(1-c\tau)^2}$.

Finally, the probability that all queries succeed is at least $1-\delta$.

Proof. We first use Lemma 4.12 to initiate $k \geq \Omega((d+\log(1/\delta))\log(nd))$ different JLT matrices with parameters $(m+1,\varepsilon,0.99)$. Then, for each JLT matrix $S_i \in \mathbb{R}^{s\times d}$, we run the quantization process on it. Specifically, this requires us to use $\kappa = s\log(ns/(\lambda\delta))$ independent ANN data structures due to Lemma 5.10.

Throughout the proof, we will condition on the event that there exists some $i \in [k]$ such that S_i preserves the pair-wise distances between any query point and points in X. To simplify the notation, we use S to denote the corresponding JLT matrix S_i.

We consider the following: given a query point $Sx \in \mathbb{R}^s$, we quantize it into a point $\widehat{x} \in \mathbb{R}^s$, then we use $\widehat{x}$ as our query. Let Sy be the nearest neighbor of $\widehat{x}$, the ANN data structure will output a point Sy' with the guarantee that $\|Sy'-\widehat{x}\|_2 \leq \overline{c}\cdot\|Sy-\widehat{x}\|_2$. Towards the end, we wish to have a bound on the term $\|x-y'\|_2$ in terms of $\|x-y\|_2$.

$$\begin{aligned}
\|Sy'-Sx\|_2 &= \|Sy'-\widehat{x}+\widehat{x}-Sx\|_2 \\
&\leq \|Sy'-\widehat{x}\|_2 + \|\widehat{x}-Sx\|_2 \\
&\leq \overline{c}\cdot\|Sy-\widehat{x}\|_2 + \lambda \\
&\leq \overline{c}\cdot\|Sy-Sx+Sx-\widehat{x}\|_2 + \lambda \\
&\leq \overline{c}\cdot(\|Sy-Sx\|_2+\lambda)+\lambda \\
&\leq \overline{c}\cdot((1+\varepsilon)\|y-x\|_2+\alpha+\lambda)+\lambda,
\end{aligned}$$

on the other hand, we know that $\|x-y'\|_2 \leq \frac{\|Sy'-Sx\|_2+\alpha}{1-\varepsilon}$, we hence conclude that

$$\begin{aligned}
\|x-y'\|_2 &\leq \frac{\overline{c}\cdot(1+\varepsilon)\|x-y\|_2+(1+\overline{c})\lambda+(\overline{c}+1)\alpha}{1-\varepsilon} \\
&= \underbrace{\overline{c}\cdot(1+O(\varepsilon))}_{\widetilde{c}}\|x-y\|_2 + \underbrace{(1+O(\varepsilon))\cdot((1+\overline{c})\cdot\lambda+(\overline{c}+1)\cdot\alpha)}_{\widetilde{\lambda}}.
\end{aligned}$$

By further setting $\widetilde{r} = \frac{r}{1+\varepsilon}$, we conclude we get a $(\widetilde{c},\widetilde{r})$-$\mathsf{ANN}$ data structure with additive error $\widetilde{\lambda}$. It is then instructive to compute the relationship between $\widetilde{c},\widetilde{r}$ and c,τ. Using the same proof strategy as in Theorem 5.6, we conclude a similar correctness result with the parameters (c,τ) set according to $\widetilde{c}$ and $\widetilde{r}$.

Per Theorem 5.6, we have $\widetilde{r}^2 = 2-2\tau$, hence $\tau = 1-0.5\widetilde{r}^2$ and $c = \frac{1-0.5\widetilde{c}\widetilde{r}^2}{1-0.5\widetilde{r}^2}$, which means $\widetilde{c}^2 = \frac{1-c\tau}{1-\tau}$. Finally, use the relationship $\widetilde{c} = \overline{c}(1+\varepsilon)$, we conclude that $\overline{c}^2 = \frac{1-c\tau}{(1-\tau)(1+\varepsilon)^2}$.

We can then consider using two different ANN data structures:

Part 1. If we are to use Theorem 5.4, we have the following relationship between ρ and $\overline{c}$: $\rho = \frac{1}{2\overline{c}^2-1}+o(1)$, this implies

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

Setting $f(c,\tau) = \frac{(1-\tau)(1+\varepsilon)^2}{2-2c\tau-(1-\tau)(1+\varepsilon)^2}$, we are done.

Part 2. Using Theorem 5.5, we know that $\rho = \frac{2}{\overline{c}^2} - \frac{1}{\overline{c}^4} + o(1)$, hence

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

It suffices to set $f(c,\tau)$ to this value.

Use the relationship $\overline{c}^2 = \frac{1-c\tau}{(1-\tau)(1+O(\varepsilon))^2}$ we derived above, we can further simplify $\widetilde{\lambda}$:

$$(1+O(\varepsilon)) \cdot ((1+\overline{c}) \cdot \lambda + \overline{c} \cdot \alpha) \leq O(1) \cdot \sqrt{\frac{1-c\tau}{1-\tau}} \cdot (\lambda + \alpha).$$

Therefore, we simplify $\widetilde{\lambda} \leq O(\sqrt{\frac{1-c\tau}{1-\tau}} \cdot (\lambda+\alpha))$, we conclude that we get a $(c,\tau,\widetilde{\lambda})$-Max-IP.

We remark that $\widetilde{\lambda}$ merely affects the *quality* of the Max-IP estimates, the success probability is still related to λ, since the quantization process generates a λ-net and it suffices to union bound over all vertices on this λ-net. Hence the number of independent data structures we need to use in order to boost success probability is $O(s\log(ns/(\lambda\delta)))$, as desired.

Finally, at query time, it suffices for us to sample $\Omega(\log(1/\delta))$ sketches uniformly at random to guarantee that with probability at least $1-2\delta$, there exists one sketch that preserves the distance of all query points. Hence, the query time is only blowed up by a $\log(1/\delta)$ factor. □

Remark 5.12. In our later applications, we are required to dynamize the Max-IP data structures. To achieve this, we use the standard dynamization technique [OvL81] used in [AR15, ALRW17]. In this way, we achieve insert and delete time in $\widetilde{O}(sn^{\rho+o(1)} \cdot \kappa \cdot k + \mathcal{T}_S(x) \cdot k)$.

6 Min-IP Data Structure: Furthest Neighbor Search, Low-Dimensional and Robust

In this section, we design an Min-IP data structure using efficient Johnson-Lindenstrauss transform and random projection. Our Min-IP data structure is robust against the adaptive adversary, which provides foundations for its application to iterative algorithms. This section is organized as below:

- In Section 6.1, we formally define the Min-IP problem and state how we solve it with approximate furthest neighbor search (AFN).
- In Section 6.2, we give a detailed introduction of an efficient AFN algorithm.
- In Section 6.3, we start the analysis and provide the success and failure probability of random projection used in AFN data structure.
- In Section 6.4, we give the guarantee for DFN data structures, which are the building blocks of AFN.
- In Section 6.5, we present the guarantee for solving AFN via DFN data structures.
- In Section 6.6, we show how to achieve efficient and robust Min-IP via robust AFN data structures.

Throughout this section, we will use n to denote the number of data points and d to denote the dimension of the data.

6.1 Solving Min-IP via Furthest Neighbor Search

In this section, we introduce the definition of Min-IP and its connection with Approximate Furthest Neighbor Search (AFN). We start with the definition of Min-IP.

Definition 6.1 (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 $\arg\min_{p \in P} \langle p, q \rangle$.

The naive brutal force algorithm solves Min-IP in $O(nd)$ time. However, there exist algorithms that achieve time complexity sublinear in n with relaxation on the retrieved vector. These algorithms aim at solving the approximate Min-IP problem.

Definition 6.2 (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 $\min_{p \in P} \langle p, q \rangle \leq \tau$, it reports a point $p' \in P$ with similarity $\langle p', q \rangle \leq \min_{p \in P} \langle p, q \rangle / c$.

Same as Section 5.3, we are required to handle adaptive queries through quantization. Therefore, we introduce the definition of quantized approximate Min-IP.

Definition 6.3 (Quantized approximate Min-IP)**.** Let $c \in (0,1)$ and $\tau \in (0,1)$. Let $\lambda \geq 0$. Given an n-point dataset $Y \subset \mathbb{S}^{d-1}$, the goal of the (c,τ,λ)-Min-IP is to build a data structure, given a query $x \in \mathbb{S}^{d-1}$ with the promise that $\min_{p \in P} \langle p, q \rangle \leq \tau$, it reports a data point $z \in Y$ such that $\langle x, z \rangle \leq c^{-1} \min_{p \in P} \langle p, q \rangle + \lambda$.

The approximate Min-IP has a dual problem: approximate furthest neighbor (AFN). We could solve approximate Min-IP via solving AFN. To illustrate this, we first present the definition of AFN.

Definition 6.4 (Approximate Furthest Neighbor (AFN))**.** Let $\overline{c} > 1$ and $r \in (0,2)$. Given an n-point dataset $P \subset \mathbb{S}^{d-1}$ on the sphere, the goal of the $(\overline{c}, r)$-Approximate Furthest-Neighbor (AFN) problem is to build a data structure that, given a query $q \in \mathbb{S}^{d-1}$ with the promise that $\max_{p \in P} \|p - q\|_2 \geq r$, it reports a point $p' \in P$ with distance $\|p - q\|_2 \geq \max_{p \in P} \|p - q\|_2 / \overline{c}$.

Next, we show the connection between approximate Min-IP and 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 $(\overline{c} + \delta, r)$-AFN , it automatically becomes a data structure for (c,τ)-Min-IP with parameters $\tau = 1 - 0.5r^2$ and $c = \frac{1-0.5r^2}{1-0.5r^2/(\overline{c}+\delta)^2}$. This implies that

$$\begin{aligned}(\overline{c} + \delta)^2 &= \frac{cr^2}{2c - 2 + r^2} \\ &= \frac{c(2 - 2\tau)}{2c - 2 + (2 - 2\tau)} \\ &= \frac{c(1-\tau)}{c - \tau} \end{aligned} \tag{2}$$

Next, we show that $\frac{c(1-\tau)}{c-\tau}$ is increasing as τ increase.

Lemma 6.5. *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. □

Similarly, using transformations in Definition 5.7, AFN data structures could be used to solve approximate Min-IP problem where data and query vectors are non-unit. We could also adjust the D_X and D_Y for better efficiency.

6.2 Algorithm

In this section, we present our algorithm that solves approximate Min-IP efficiently. We start with presenting the SortedList data structure in Alg. 1.

Algorithm 1 Helper data structure SortedList

data structure SortedList ▷ This data structure can be implemented via various self-balancing binary search trees
  Init($P \in (\mathbb{R} \times \mathbb{R}^d)^n$) ▷ n points each has a real key and d dimensional data points, $O(n \log n)$ time
  Insert($p \in \mathbb{R} \times \mathbb{R}^d$) ▷ Insert a single key-value pair, $O(\log n)$ time
  Delete($p \in \mathbb{R} \times \mathbb{R}^d$) ▷ Remove a single key-value pair, $O(\log n)$ time
  SearchLeq($T \in \mathbb{R}$) ▷ Output a subtree with key less than or equal to T, $O(\log n)$ time
  SearchGeq($T \in \mathbb{R}$) ▷ Output a subtree with key greater than or equal to T, $O(\log n)$ time
  Max() ▷ Return max key-value pair, $O(\log n)$ time
  Min() ▷ Return min key-value pair, $O(\log n)$ time
end data structure

Next, we introduce a task called $(\overline{c}, r)$-DFN defined in Task 6.6.

Task 6.6. Let $P \subset \mathbb{R}^d$ be an n-point dataset. Let $\overline{c} > 1$ We define the $(\overline{c}, r)$-DFN problem as follows: given a point $q \in \mathbb{R}^d$ and $r > 0$, if there exists a point $p \in P$ such that, if $\|p - q\|_2 \geq r$, then the data structure reports a point $\widehat{p} \in P$ such that $\|\widehat{p} - q\|_2 \geq r/\overline{c}$, otherwise, it reports "Fail".

The data structure for $(\overline{c}, r)$-DFN shown in Alg. 2 and Alg. 3 is the building block of our approximate Min-IP algorithm.

Algorithm 2 Data structure DFN: members, init, insert and delete

data structure DFN ▷ Theorem 6.9

members
- $\ell \in \mathcal{N}_+$ ▷ Number of random directions
- $G \in \mathbb{R}^{\ell \times d}$ ▷ Random Gaussian vectors
- SortedList $L_1, \ldots, L_\ell$ ▷ ℓ sorted lists, Alg. 1
- $t \in \mathbb{R}_+$ ▷ Threshold parameter
- $\overline{c} \in (1, \infty)$ ▷ Approximation parameter

end members

procedure Init($A \in \mathbb{R}^{n \times d}, \overline{c} \in (1, \infty)$)
- $\overline{c} \leftarrow \overline{c}$
- $\ell \leftarrow \Theta(n^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n)$
- $t \leftarrow \Theta(\sqrt{\log n})$ ▷ t is the solution to $e^{t^2/2}/t = 2n$
- $G_{i,j} \sim \mathcal{N}(0,1), \forall i \in [\ell], \forall j \in [d]$ ▷ Each entry is a standard Gaussian
- SortedList $L_1, \ldots, L_\ell$ ▷ Alg. 1
- Let g_i denote the i-th row of G and A_i denote the i-th row of A
- **for** $i = 1 \to \ell$ **do**
  - $P_i \leftarrow \{(\langle g_i, a_j \rangle, a_j) : j \in [n]\}$
  - L_i.Init(P_i) ▷ Alg. 1
- **end for**

end procedure

procedure Insert($p \in \mathbb{R}^d$)
- **for** $i = 1 \to \ell$ **do**
  - $k \leftarrow \langle g_i, p \rangle$
  - L_i.Insert((k, p)) ▷ Alg. 1
- **end for**

end procedure

procedure Delete($p \in \mathbb{R}^d$)
- **for** $i = 1 \to \ell$ **do**
  - $k \leftarrow \langle g_i, p \rangle$
  - L_i.Delete((k, p)) ▷ Alg. 1
- **end for**

end procedure

end data structure

Algorithm 3 Data structure DFN: query

data structure DFN ▷ Theorem 6.9

procedure QUERY($q \in \mathbb{R}^d, r \in \mathbb{R}_+$)
    $T \leftarrow rt/\overline{c}$
    $i \leftarrow 1$, $m \leftarrow 0$
    $S \leftarrow \emptyset$
    while $i \leq \ell$ and $m \leq 2\ell + 1$ **do**
        dist $\leftarrow \langle g_i, q \rangle$
        $T_1 \leftarrow L_i$.SEARCHLEQ(dist $- T$)
        $T_2 \leftarrow L_i$.SEARCHGEQ($T +$ dist)
                ▷ Search for the subtree such that $|\langle g_i, q - p \rangle| \geq T$
        if $m + |T_1| + |T_2| \leq 2\ell + 1$ **then**
            $S \leftarrow S \cup T_1 \cup T_2$
            $m \leftarrow m + |T_1| + |T_2|$
        else
            Add points from T_1 and T_2 to S until $|S| = 2\ell + 1$
            $m \leftarrow 2\ell + 1$
        end if
        $i \leftarrow i + 1$
    end while
    for $p \in S$ **do**
        if $\|p - q\|_2 \geq r/\overline{c}$ **then**
            return p
        else
            return "Fail"
        end if
    end for
end procedure

end data structure

Finally, in Alg. 4 and Alg. 5, we present our algorithm that solves AFN (see Definition 6.4). As AFN is the dual problem of approximate Min-IP, this algorithm could be used to solve approximate Min-IP.

Algorithm 4 AFN Algorithm: members, init, insert and delete

data structure AFN ▷ Theorem 6.11

members
- $\varepsilon \in (0,1)$
- $\delta \in (0,1)$
- $s \in \mathcal{N}_+$ ▷ Number of data structures
- DFN $\text{dfn}_1, \text{dfn}_2, \ldots, \text{dfn}_s$
- $\text{bw} \in \mathbb{R}_+$ ▷ Boxwidth of all points
- SortedList $T_1, \ldots, T_d$ ▷ Max/min value for each dimension

end members

procedure Init($A \in \mathbb{R}^{n \times d}, \overline{c} \in (1, \infty), \delta \in (0,1)$)
- $\varepsilon \leftarrow \overline{c} - 1$
- $s \leftarrow \Theta(\log\log(d/\delta))$
- dfn_i.Init$(A, \overline{c})$ for all $i \in [s]$ ▷ Alg. 2
- **for** $j = 1 \to d$ **do**
  - T_j.Init$(A_{*,j})$ ▷ Alg. 1
- **end for**
- $\text{bw} \leftarrow \max_{j \in [d]} |T_j.\text{Max}() - T_j.\text{Min}()|$ ▷ 1d boxwidth

end procedure

procedure Insert($p \in \mathbb{R}^d$)
- dfn_i.Insert(p) for all $i \in [s]$ ▷ Alg. 2
- **for** $j = 1 \to d$ **do**
  - T_j.Insert(p_j) ▷ Alg. 1
- **end for**
- $\text{bw} \leftarrow \max_{j \in [d]} |T_j.\text{Max}() - T_j.\text{Min}()|$

end procedure

procedure Delete($p \in \mathbb{R}^d$)
- dfn_i.Delete(p) for all $i \in [s]$ ▷ Alg. 2
- **for** $j = 1 \to d$ **do**
  - T_j.Delete(p_j) ▷ Alg. 1
- **end for**
- $\text{bw} \leftarrow \max_{j \in [d]} |T_j.\text{Max}() - T_j.\text{Min}()|$

end procedure

end data structure

Algorithm 5 AFN Algorithm: query

data structure AFN ▷ Theorem 6.11

procedure QUERY($q \in \mathbb{R}^d$)
  lo $\leftarrow$ bw$/2$
  hi $\leftarrow \sqrt{d}/\varepsilon \cdot$ bw
  Binary search over the range [lo, hi] to search for $r \in \mathbb{R}_+$,
  with the predicate dfn_i.QUERY(q, r) for $i \in [s]$
  ▷ $\Theta(\log(d/\varepsilon\delta))$ rounds
  for $i = 1 \to s$ **do**
    $p \leftarrow \mathrm{dfn}_i$.QUERY$(q, r)$
    if $p \neq$ "Fail" **then**
      return p
    end if
  end for
  return "Fail"
end procedure

end data structure

6.3 Success and Failure Probability of Random Projection

In this section, we analyze both the success and failure probability of random projection. To start with, we supply a technical lemma that upper bounds the failure probability that two points are far in the random direction but close in the original space.

Lemma 6.7. *Let t be the solution to $e^{t^2/2}/t = 2n$ and $T = rt/\overline{c}$. Let $\widehat{p}$ be a point such that $\|\widehat{p} - q\|_2 < r/\overline{c}$. Then*

$$\Pr_{g \sim \mathcal{N}(0,I)}[|\langle g, \widehat{p}\rangle - \langle g, q\rangle| \geq T] \leq 1/n.$$

Proof. Observe that

$$\begin{aligned}
\Pr[|\langle g, \widehat{p}\rangle - \langle g, q\rangle| \geq T] &= \Pr\left[\frac{|\langle g, \widehat{p} - q\rangle|}{\|\widehat{p} - q\|_2} \geq T/\|\widehat{p} - q\|_2\right] \\
&\leq \Pr\left[\frac{|\langle g, \widehat{p} - q\rangle|}{\|\widehat{p} - q\|_2} \geq \frac{T}{r/\overline{c}}\right] \\
&= \Pr\left[\frac{|\langle g, \widehat{p} - q\rangle|}{\|\widehat{p} - q\|_2} \geq t\right] \\
&\leq 2\exp(-t^2/2)/t \\
&\leq 1/n.
\end{aligned}$$

The second step follows from $\|\widehat{p} - q\|_2 < r/\overline{c}$. For the fourth step, note that since standard Gaussian is 2-stable (Fact 3.6), we know that $\frac{\langle g, \widehat{p} - q\rangle}{\|\widehat{p} - q\|_2}$ follows a standard Gaussian distribution, so we can apply part 1 of the Gaussian concentration bound (Fact 3.7). □

Next, we provide a lower bound on the success probability that when two points are far away from each other in the random direction, then they are far away in the original space.

Lemma 6.8. *Let t be the solution to $e^{t^2/2}/t = 2n$ and $T = rt/\overline{c}$. Let $\ell = O(n^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n)$. Let p be a point such that $\|p - q\|_2 \geq r$. Then*

$$\Pr_{g \sim N(0,I)}[|\langle g, p\rangle - \langle g, q\rangle| \geq T] \geq 1/\ell.$$

Proof. The proof is similar to Lemma 6.7, consider

$$\begin{aligned}
\Pr[|\langle g, p\rangle - \langle g, q\rangle| \geq T] = & \Pr\left[\frac{|\langle g, p-q\rangle|}{\|p-q\|_2} \geq \frac{T}{\|p-q\|_2}\right] \\
\geq & \Pr\left[\frac{|\langle g, p-q\rangle|}{\|p-q\|_2} \geq \frac{T}{r}\right] \\
= & \Pr\left[\frac{|\langle g, p-q\rangle|}{\|p-q\|_2} \geq \frac{t}{\overline{c}}\right] \\
\geq & 2B \cdot \exp(-(t/\overline{c})^2/2)/(t/\overline{c}) \\
= & \frac{2B \cdot \overline{c}(\exp(-t^2/2)/t)^{1/\overline{c}^2}}{t^{1-1/\overline{c}^2}} \\
= & \frac{2B\overline{c}}{n^{1/\overline{c}^2} t^{1-1/\overline{c}^2}}.
\end{aligned}$$

The second step follows from $\|p - q\|_2 \geq r$, the fourth step follows from Fact 3.7. Note that by picking $\ell = O(n^{1/\overline{c}^2} \log^{(1-1/\overline{c}^2)/2} n)$, we get our desired result. □

6.4 Guarantees of DFN Data Structure

In this section, we setup the theoretical guarantees of Alg. 2. We state and prove the following theorem regarding our DFN data structure.

Theorem 6.9. *Let $P \subset \mathbb{R}^d$ be an n-point dataset and $\overline{c} > 1$. There exists a randomized dynamic data structure (Alg. 2, 3) that solves $\overline{c}$-DFN task (see Task 6.6) using $O(n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n)$ space with the following operations:*

- Init: *Preprocess P in $O(n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n)$ time;*
- Query: *Given a point $q \in \mathbb{R}^d$ and $r > 0$, either outputs a point $\widehat{p} \in P$ such that $\|\widehat{p} - q\|_2 \geq r/\overline{c}$ with constant probability or outputs "Fail" in $O(n^{1/\overline{c}^2}(d + \log n) \log n)$ time;*
- Insert: *Insert a point $p \in \mathbb{R}^d$ into the data structure in $O(n^{1/\overline{c}^2} \log^2 n)$ time;*
- Delete: *Delete a point $p \in P$ in $O(n^{1/\overline{c}^2} \log^2 n)$ time.*

Proof. We prove four corresponding parts of Theorem 6.9 accordingly.

Space: Storing the $\ell \times d$ standard Gaussian matrix takes $O(\ell d)$ space. Maintaining ℓ sorted list takes $O(\ell n)$ space. Thus, the total space is

$$O(\ell(d + n)) = O(n^{1+1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n + dn^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n).$$

Procedure Init: By Alg. 2, the initiation needs to initialize an $\ell \times d$ standard Gaussian matrix, which takes $O(\ell d)$ time, processing all points into sorted lists takes $O(\ell(d + n \log n))$ time. Thus, the total time for Init is

$$O(\ell(d + n \log n)) = O(n^{1+1/\overline{c}^2} \log n \log^{(1-1/\overline{c})/2} n + dn^{1/\overline{c}^2} \log^{(1-1/\overline{c})/2} n).$$

Procedure Query: We first show the correctness. Our goal is to prove that with constant probability, our data structure retrieves a pair (p, i) among the first $2\ell+1$ pairs where each point p satisfies $|\langle g_i, p\rangle - \langle g_i, q\rangle| \geq T$ and at least one of the point p has the guarantee that $\|p-q\|_2 \geq r/\overline{c}$.

We first justify that picking $2\ell+1$ pairs suffices for at least one point has desired distance guarantee, with constant probability. Let $Y_{\widehat{p},i}$ denote the event that a pair $(\widehat{p}, i) \in P \times [\ell]$ has the property that $\|\widehat{p}-q\|_2 < r/\overline{c}$ and $|\langle g_i, q\rangle - \langle g_i, \widehat{p}\rangle| \geq T$. Then

$$\begin{aligned}\mathbb{E}[\sum_{(\widehat{p},i)} Y_{\widehat{p},i}] &= n\ell \cdot \Pr[Y_{\widehat{p},i}] \\ &\leq n\ell \cdot \frac{1}{n} \\ &= \ell.\end{aligned}$$

The second step follows from Lemma 6.7. Note that $\mathbb{E}[\sum_{(\widehat{p},i)} Y_{\widehat{p},i}]$ is the expected total number of such pairs, this means via a Markov bound, with the probability at least $1/2$, there are no more than 2ℓ such pairs. Thus, if we retrieve exactly $2\ell+1$ such pairs, there must be at least one pair (p, i) with $\|p-q\|_2 \geq r/\overline{c}$. Next, we analyze the failure probability when picking $2\ell+1$ pairs. Note that for a point $p \in P$ with $|\langle g_i, p\rangle - \langle g_i, q\rangle| \geq T$, the probability that $\|p-q\|_2 < r/\overline{c}$ is at most $1-1/\ell$, due to Lemma 6.8. This means the probability that *some* i among the first $2\ell+1$ pairs has the property that $\|p-q\|_2 \geq r/\overline{c}$ is at least

$$1-(1-1/\ell)^{2\ell+1} \geq 1-1/e,$$

this means we have a constant probability of success. Thus, our DFN data structure has a constant probability to output a point which is *not* within the distance of $r/\overline{c}$ from q.

For the running time, note that we do at most ℓ rounds of search, at each round, we search the sorted lists, so we pay a total of $O(\ell \log n)$ for searching the lists. Finally, we need to examine these $2\ell+1$ pairs for their distances, this takes $O(\ell d)$ time. Therefore, the total running time is

$$O(n^{1/\overline{c}^2} \log n \log^{(1-1/\overline{c}^2)/2} n + dn^{1/\overline{c}^2} \log^{(1-1/\overline{c}^2)/2} n).$$

Procedure Insert and Delete: It is obvious that the running time of both procedures is $O(\ell(d+\log n))$, which is the same as the time of procedure Query. □

6.5 Guarantees of AFN Data Structure

In this section we provide an analysis for an AFN data structure implemented via DFN data structure. The idea is to use binary search to find the correct distance r. The search range is determined via the notion of *box width*.

Definition 6.10. Given a dataset $P \subset \mathbb{R}^d$, we define the box width of P, denoted as $\mathrm{bw}(P)$ or bw if P is clear from context as

$$\mathrm{bw}(P) := \max_{i\in[d]} |\max_{p\in P}(p_i) - \min_{p\in P}(p_i)|.$$

Note that p_i denotes the i-th coordinate of point p.

We now proceed with the formal statement and proof.

Theorem 6.11. *Let $P \subset \mathbb{R}^d$ be an n-point dataset, $\overline{c} > 1$, $r > 0$ and $\delta > 0$. Let $\varepsilon = \overline{c} - 1$. There exists a randomized dynamic data structure (Alg. 4, 5) that solves $(\overline{c}+\delta, r)$-AFN task using space $O((n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta) + dn)$ with the following operations:*

- INIT*: Preprocess P in $O((n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta))$ time;*
- QUERY*: Given a point $q \in \mathbb{R}^d$, returns a $(\overline{c}+\delta)$-approximate furthest neighbor $p \in P$ with constant probability in $O(n^{1/\overline{c}^2}(d + \log n) \log n \log(d/\varepsilon\delta) \log\log(d/\varepsilon\delta))$ time;*
- INSERT*: Insert a point $p \in \mathbb{R}^d$ into the data structure in $O(n^{1/\overline{c}^2} \log^2 n \log\log(d/\varepsilon\delta) + d \log n)$ time;*
- DELETE*: Delete a point $p \in \mathbb{R}^d$ from the data structure in $O(n^{1/\overline{c}^2} \log^2 n \log\log(d/\varepsilon\delta) + d \log n)$ time.*

Proof. We start with the space complexity

Space: We note that there are $s = O(\log\log(d/\varepsilon\delta))$ DFN data structures to initialize, each data structure takes $O(n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n)$ space. Moreover. the d different sorted lists for each dimension takes $O(dn)$ space. Therefore, the final space is

$$O((n^{1+1/\overline{c}^2} \log n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta) + dn)$$

Next, we prove four parts separately.

Procedure INIT: We note that there are $s = O(\log\log(d/\varepsilon\delta))$ DFN data structures to initialize, each data structure takes $O(n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n)$ time. To initialize d different sorted lists for each dimension, it takes $O(dn \log n)$ time. Finally, computing boxwidth bw takes $O(\log n)$ time. Thus, the total time in initialization phase is

$$O((n^{1+1/\overline{c}^2} \log^2 n + dn^{1/\overline{c}^2} \log n) \log\log(d/\varepsilon\delta)).$$

Procedure QUERY: We need to prove the runtime and correctness of the procedure. For the runtime, we note that QUERY makes $O(\log(d/\varepsilon\delta))$ calls to binary search with $O(\log\log(d/\varepsilon\delta))$ different data structures. Each call takes $O(n^{1/\overline{c}^2}(d + \log n) \log n)$ time by Theorem 6.9. This completes the proof of runtime.

For correctness, note that for any query $q \in \mathbb{R}^d$, if p is its furthest neighbor then $\|p-q\|_2 \geq \text{bw}/2$ since q must be further from one point defining boxwidth. On the other hand, if the distance from q to the center of box is at least $2\sqrt{d}/\varepsilon \cdot \text{bw}$, then any point in P is a $(1+\varepsilon)$-approximate furthest neighbor. To see this, note that any point $p \in P$ is at most $\sqrt{d}/2 \cdot \text{bw}$ away from the center, so the nearest point from the box to the center is at least $(\frac{2}{\varepsilon} - \frac{1}{2})\sqrt{d} \cdot \text{bw}$. On the other hand, the furthest point on the box to q has a distance $(\frac{2}{\varepsilon} + \frac{1}{2})\sqrt{d} \cdot \text{bw}$, it suffices to show that $(\frac{2}{\varepsilon} + \frac{1}{2})\sqrt{d} \cdot \text{bw}/(1+\varepsilon) \leq (\frac{2}{\varepsilon} - \frac{1}{2})\sqrt{d} \cdot \text{bw}$, since the furthest neighbor to q from dataset P must have distance smaller than $(\frac{2}{\varepsilon} + \frac{1}{2})\sqrt{d} \cdot \text{bw}$. Note that

$$(\frac{2}{\varepsilon} + \frac{1}{2})/(1+\varepsilon) = \frac{4+\varepsilon}{2\varepsilon(1+\varepsilon)},$$

On the other hand,

$$\frac{2}{\varepsilon} - \frac{1}{2} = \frac{4-\varepsilon}{2\varepsilon}$$

$$= \frac{4 + 3\varepsilon - \varepsilon^2}{2\varepsilon(1+\varepsilon)}.$$

Since $\varepsilon \in (0, 1)$, we always have $3\varepsilon - \varepsilon^2 > \varepsilon$, as desired.

This gives a lower and upper bound on binary search, namely we search the range $[\text{bw}/2, 2\sqrt{d}/\varepsilon \cdot \text{bw}]$, hence, we need $O(\log \frac{d}{\varepsilon\delta})$ rounds to achieve a δ-precision solution. This leads to a $\overline{c}(1+\varepsilon)(1+\delta) = (1+\varepsilon)^2(1+\delta)$-approximation furthest neighbor. By picking ε as $\varepsilon/2$ and δ as $\delta/3$, this leads to a $(1+\varepsilon+\delta) = (\overline{c}+\delta)$-approximate furthest neighbor. Finally, to amplify the success probability of each query, we need to use $O(\log\log \frac{d}{\varepsilon\delta})$ different data structures. This completes the correctness analysis.

Procedure Insert and Delete: Both of these procedures require to insert or delete a point to s different data structures and update the sorted list for each dimension, then compute the new boxwidth. The insert/delete point step takes $O(n^{1/\overline{c}^2} \log^2 n \log\log \frac{d}{\varepsilon\delta})$ time and update the sorted list takes $O(d \log n)$ time. This completes the proof. □

Next, we introduce several corollaries that simplify the time complexity in solving AFN.

Corollary 6.12. *Let $P \subset \mathbb{R}^d$ be an n-point dataset, $\overline{c} > \sqrt{2}$, and $r > 0$. There exists a randomized dynamic data structure (Alg. 4, 5) that solves $(2\overline{c}, r)$-AFN with query time $O(n^{0.5}(d + \log n)\log n \log d \log\log d)$, preprocessing time $O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d)$ and space $O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d + nd)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.5}\log^2 n \log\log d + d\log n)$ time.*

Proof. If $\overline{c} > \sqrt{2}$, we have $1/\overline{c}^2 < 0.5$. We take this fact into the preprocessing, query, insert and delete time and get the following:

Space

$$O((n^{1+1/\overline{c}^2}\log^2 n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta) + nd) = O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d + nd)$$

Preprocesing time

$$O((n^{1+1/\overline{c}^2}\log^2 n + dn^{1/\overline{c}^2}\log n)\log\log(d/\varepsilon\delta)) = O((n^{1.5}\log^2 n + dn^{0.5}\log n)\log\log d)$$

Query time

$$O(n^{1/\overline{c}^2}(d+\log n)\log n \log(d/\varepsilon\delta)\log\log(d/\varepsilon\delta)) = O(n^{0.5}(d+\log n)\log n \log d \log\log d)$$

Insert/delete time

$$O(n^{1/\overline{c}^2}\log^2 n \log\log(d/\varepsilon\delta) + d\log n) = O(n^{0.5}\log^2 n \log\log d + d\log n)$$

□

Corollary 6.13. *Let $P \subset \mathbb{R}^d$ be an n-point dataset, $\overline{c} > 10$, and $r > 0$. There exists a randomized dynamic data structure (Alg. 4, 5) that solves $(2\overline{c}, r)$-AFN with query time $O(n^{0.01}(d + \log n)\log n \log d \log\log d)$, preprocessing time $O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d)$ and space $O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d + nd)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.01}\log^2 n \log\log d + d\log n)$ time.*

Proof. If $\bar{c} > 10$, we have $1/\bar{c}^2 < 0.01$. We take this fact into the preprocessing, query, insert and delete time and get the following:

Space

$$O((n^{1+1/\bar{c}^2}\log^2 n + dn^{1/\bar{c}^2}\log n)\log\log(d/\varepsilon\delta) + nd) = O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d + nd)$$

Preprocesing time

$$O((n^{1+1/\bar{c}^2}\log^2 n + dn^{1/\bar{c}^2}\log n)\log\log(d/\varepsilon\delta)) = O((n^{1.01}\log^2 n + dn^{0.01}\log n)\log\log d)$$

Query time

$$O(n^{1/\bar{c}^2}(d+\log n)\log n\log(d/\varepsilon\delta)\log\log(d/\varepsilon\delta)) = O(n^{0.01}(d+\log n)\log n\log d\log\log d)$$

Insert/delete time

$$O(n^{1/\bar{c}^2}\log^2 n\log\log(d/\varepsilon\delta) + d\log n) = O(n^{0.01}\log^2 n\log\log d + d\log n)$$

□

6.6 Guarantees of Approximate Min-IP via AFN

In this section, we show how to use AFN data structure to solve approximate Min-IP. We take the adaptive query in iterative optimization algorithm into consideration and design a robust algorithm against adversary.

Theorem 6.14 (Formal version of Theorem 2.2)**.** *Let $c \in (0,1)$, $\tau \in (0,1)$, $\lambda \in (0,1)$, $\varepsilon \in (0,1)$ and $\delta \in (0,1)$. We define the following additional parameters:*

- $\alpha = O(\frac{1}{(nd)^9})$*, the additive error by Lemma 4.12;*
- $s \leq d$*, the dimension of JLT;*
- $k = O((d+\log(1/\delta))\log(nd))$*, number of independent JLT sketches;*
- $\kappa = s\log(ns/(\lambda\delta))$*;*
- $\widetilde{\lambda} = O(\sqrt{\frac{c-\tau}{c(1-\tau)}})\cdot(\lambda+\alpha)$*, the additive error of* Min-IP*.*

Let $\mathcal{T}_S(x)$ denote the time of applying S to a vector $x \in \mathbb{R}^d$. Given a set of n-points $Y \subset \mathbb{S}^{d-1}$ on the sphere, one can build a dynamic data structure with preprocessing time $\mathcal{T}_{\mathsf{init}}\cdot\kappa\cdot k + \mathcal{T}_S(Y)\cdot k$, space $\mathcal{S}_{\mathsf{space}}\cdot\kappa\cdot k$ insert time $(\mathcal{T}_{\mathsf{insert}}\cdot\kappa + \mathcal{T}_S(x))\cdot k$ and delete time $(\mathcal{T}_{\mathsf{delete}}\cdot\kappa + \mathcal{T}_S(x))\cdot k$ so that for every query $x \in \mathbb{S}^{d-1}$ in an adaptive sequence $X = \{x_1,\ldots,x_T\}$, the query time is $\widetilde{O}(\mathcal{T}_{\mathsf{query}}\cdot\kappa + \mathcal{T}_S(x))$:

- *if* Min-IP$(x,Y) \leq \tau$*, then we output a vector in Y that is a $(c,\tau,\widetilde{\lambda})$-*Min-IP *with respect to (x,Y).*
- *otherwise, we output* fail*.*

Further,

- *If $c \in (\tau, \frac{8\tau}{(1-\varepsilon)^2\tau+2\varepsilon+7})$, we have $\mathcal{T}_{\mathsf{init}} = O((n^{1.5}\log^2 n + sn^{0.5}\log n)\log\log s)$, $\mathcal{S}_{\mathsf{space}} = O((n^{1.5}\log^2 n + sn^{0.5}\log n)\log\log s + ns)$, $\mathcal{T}_{\mathsf{query}} = O(n^{0.5}(s+\log n)\log n\log s\log\log s)$ and $\mathcal{T}_{\mathsf{insert}} = \mathcal{T}_{\mathsf{delete}} = O(n^{0.5}\log^2 n\log\log s + s\log n)$*

- *If* $c \in (\tau, \frac{400\tau}{(1-\varepsilon)^2\tau+2\varepsilon+399})$, *we have* $\mathcal{T}_{\mathsf{init}} = O((n^{1.01}\log^2 n + sn^{0.01}\log n)\log\log s)$, $\mathcal{S}_{\mathsf{space}} = O((n^{1.01}\log^2 n + sn^{0.01}\log n)\log\log s + ns)$, $\mathcal{T}_{\mathsf{query}} = O(n^{0.01}(s+\log n)\log n \log s \log\log s)$ *and* $\mathcal{T}_{\mathsf{insert}} = \mathcal{T}_{\mathsf{delete}} = O(n^{0.01}\log^2 n \log\log s + s\log n)$

Finally, the probability that all queries succeed is at least $1-\delta$.

Proof. We first use Lemma 4.12 to initiate $k \geq \Omega((d+\log(1/\delta))\log(nd))$ different JLT matrices with parameters $(m+1, \varepsilon, 0.99)$. Then, for each JLT matrix $S_i \in \mathbb{R}^{s\times d}$, we run the quantization process on it. Specifically, this requires us to use $\kappa = s\log(ns/(\lambda\delta))$ independent AFN data structures due to Lemma 5.10.

Throughout the proof, we will condition on the event that there exists some $i \in [k]$ such that S_i preserves the pair-wise distances between any query point and points in X. To simplify the notation, we use S to denote the corresponding JLT matrix S_i.

We consider the following: given a query point $Sx \in \mathbb{R}^s$, we quantize it into a point $\widehat{x} \in \mathbb{R}^s$, then we use $\widehat{x}$ as our query. Let Sy be the furthest neighbor of $\widehat{x}$, the AFN data structure will output a point Sy' with the guarantee that $\|Sy' - \widehat{x}\|_2 \geq \|Sy - \widehat{x}\|_2/\overline{c}$. Towards the end, we wish to have a bound on the term $\|x - y'\|_2$ in terms of $\|x-y\|_2$.

$$\begin{aligned}
\|Sy' - Sx\|_2 &= \|Sy' - \widehat{x} + \widehat{x} - Sx\|_2 \\
&\geq \|Sy' - \widehat{x}\|_2 - \|Sx - \widehat{x}\|_2 \\
&\geq \|Sy - \widehat{x}\|_2/\overline{c} - \lambda \\
&\geq \|Sy - Sx + Sx - \widehat{x}\|_2/\overline{c} - \lambda \\
&\geq (\|Sy - Sx\|_2 - \lambda)/\overline{c} - \lambda \\
&\geq \overline{c}^{-1}\cdot((1-\varepsilon)\|y-x\|_2 - \alpha - \lambda) - \lambda,
\end{aligned}$$

on the other hand, we know that $\|x - y'\|_2 \geq \frac{\|Sy'-Sx\|_2 - \alpha}{1+\varepsilon}$, we hence conclude that

$$\begin{aligned}
\|x - y'\|_2 &\geq \frac{\overline{c}^{-1}\cdot(1-\varepsilon)\|x-y\|_2 - (1+\overline{c}^{-1})\lambda - (1+\overline{c}^{-1})\alpha}{1+\varepsilon} \\
&= \underbrace{\overline{c}^{-1}\cdot(1-O(\varepsilon))}_{\widetilde{c}^{-1}}\|x-y\|_2 - \underbrace{(1-O(\varepsilon))\cdot((1+\overline{c}^{-1})\cdot\lambda + (1+\overline{c}^{-1})\cdot\alpha)}_{\widetilde{\lambda}}.
\end{aligned}$$

By further setting $\widetilde{r} = \frac{r}{1-\varepsilon}$, we conclude we get a $(\widetilde{c}, \widetilde{r})$-AFN data structure with additive error $\widetilde{\lambda}$. Moreover, this $(\widetilde{c}, \widetilde{r})$-AFN data structure would also be a data structure for (c,τ)-Min-IP with $\tau = 1 - 0.5\widetilde{r}^2$ and $c = \frac{1-0.5r^2}{1-0.5\widetilde{r}^2/\widetilde{c}^2}$. Using Eq. (2), we have $\widetilde{c}^2 = \frac{c(1-c\tau)}{c-\tau}$.

Next, we present how to obtain the desired query, preprocessing insert, and delete time complexity in the statement.

Part 1. Let $\widetilde{c} = 2\overline{c}/(1-\varepsilon)$, we conclude that $\overline{c}^2 = \frac{c(1-c\tau)(1-\varepsilon)^2}{4(c-\tau)}$. If $\tau \in (0,1)$ and $c \in (\tau, \frac{8\tau}{(1-\varepsilon)^2\tau+2\varepsilon+7})$, we have

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

$$= \frac{2(1-\varepsilon)^2\tau(1-\tau)}{8\tau-(1-\varepsilon)^2\tau^2+(1-\varepsilon)^2\tau-8\tau}$$
$$= 2$$

where the second and third steps follow from Lemma 6.5.

Then, we use Corollary 6.12 with $\overline{c}^2 > 2$ and obtain the query time $O(n^{0.5}(s+\log n)\log n \log s \log\log s)$, preprocessing time $O((n^{1.5}\log^2 n + sn^{0.5}\log n)\log\log s)$ and space $O((n^{1.5}\log^2 n + sn^{0.5}\log n)\log\log s + ns)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.5}\log^2 n \log\log s + s\log n)$ time.

Part 2. Let $\widetilde{c} = 2\overline{c}/(1-\varepsilon)$, we conclude that $\overline{c}^2 = \frac{c(1-c\tau)(1-\varepsilon)^2}{4(c-\tau)}$. If $\tau \in (0,1)$ and $c \in (\tau, \frac{400\tau}{(1-\varepsilon)^2\tau+2\varepsilon+399})$, we have

$$\begin{aligned}
\overline{c}^2 &= \frac{c(1-\tau)(1-\varepsilon)^2}{4(c-\tau)} \\
&> (1-\varepsilon)^2 \cdot \frac{400\tau}{(1-\varepsilon)^2\tau+2\varepsilon+399}) \cdot \frac{1-\tau}{4(\frac{400\tau}{(1-\varepsilon)^2\tau+2\varepsilon+399}) - \tau)} \\
&> (1-\varepsilon)^2 \cdot \frac{400\tau}{(1-\varepsilon)^2\tau-(1-\varepsilon)^2+400}) \cdot \frac{1-\tau}{4(\frac{400\tau}{(1-\varepsilon)^2\tau-(1-\varepsilon)^2+400}) - \tau)} \\
&= \frac{100(1-\varepsilon)^2\tau(1-\tau)}{400\tau-(1-\varepsilon)^2\tau^2+(1-\varepsilon)^2\tau-400\tau} \\
&= 10
\end{aligned}$$

where the second and third steps follow from Lemma 6.5.

Then, we use Corollary 6.13 with $\overline{c}^2 > 100$ and obtain the query time $O(n^{0.01}(s+\log n)\log n \log s \log\log s)$, preprocessing time $O((n^{1.01}\log^2 n + sn^{0.01}\log n)\log\log s)$ and space $O((n^{1.01}\log^2 n + sn^{0.01}\log n)\log\log s + ns)$. Moreover, the dynamic data structure supports insert or delete in $O(n^{0.01}\log^2 n \log\log s + s\log n)$ time.

Next, we analyze the additive error. Use the relationship $\overline{c}^2 = \frac{c(1-\tau)(1-\varepsilon)^2}{4(c-\tau)}$ we derived above, we can further simplify $\widetilde{\lambda}$:

$$(1-O(\varepsilon))\cdot((1+\overline{c}^{-1})\cdot\lambda + (1+\overline{c}^{-1})\cdot\alpha) \leq O(1)\cdot\sqrt{\frac{c-\tau}{c(1-\tau)}}\cdot(\lambda+\alpha).$$

Therefore, we simplify $\widetilde{\lambda} \leq O(\sqrt{\frac{c-\tau}{c(1-\tau)}}\cdot(\lambda+\alpha))$, we conclude that we get a $(c,\tau,\widetilde{\lambda})$-Min-IP. □

7 Generalized Sum of Rank-1 Matrices Sparsification

In this section, we study a generalized version of the graph sparsification problem considered in [BSS12]: to obtain a sparsifier of optimal size for a sum-of-rank-1-matrices.

- In Section 7.1, we setup the sparsification task for a sum of rank-1 matrices.
- In Section 7.2, we describe the vanilla BSS algorithm and state several lemmas.
- In Section 7.3, we present our algorithm that solves the BSS sparsifier problem with Max-IP data structure.
- In Section 7.4, we compare our method with other fast algorithms to construct BSS sparsifiers and introduce other numerical linear algebra tasks involving BSS sparsifiers that can be improved via our algorithm.

7.1 Problem Setup

In this section, we setup the problem. Specifically, we formulate the graph sparsification problem considered in [BSS12] as a general task of sparsifying a sum of rank-1 matrices, which is essentially the core result proved in [BSS12]. Throughout this section, we refer to the sparsifier in [BSS12] as BSS sparsifier.

Definition 7.1. Suppose we are given m vectors $v_i, \ldots, v_m \in \mathbb{R}^d$ satisfying $\sum_{i=1}^m v_i v_i^\top = I$, we want to find scalars $\{s_i\}_{i=1}^m$ satisfying

$$|\{s_i : s_i \neq 0\}| = O(d/\varepsilon^2),$$

such that

$$(1-\varepsilon)\cdot I \preceq \sum_{i=1}^m s_i v_i v_i^\top \preceq (1+\varepsilon)\cdot I.$$

We remark that in the standard graph setting, one has $m \leq d^2$, where m is the number of edges and d is the number of vertices. However, in the general setting we consider, it is possible that $m \gg d^2$, this also enables the study of the regime when one requires a *high precision sparsifier*, i.e., $\varepsilon \sim \frac{1}{\mathrm{poly}(d)}$. Note that in the graph setting, one cannot pick ε more than $\frac{1}{\sqrt{d}}$, which would result in $\Theta(d^2)$ edges in the graph.

We define several notions that will be used extensively in the proof of BSS sparsifier.

Definition 7.2. Let $A \in \mathbb{R}^{d\times d}$ be a symmetric matrix with eigenvalues $\lambda_1, \ldots, \lambda_d$ and $u, \ell \in \mathbb{R}$, define:

$$\Phi^u(A) := \mathrm{tr}[(uI - A)^{-1}] = \sum_{i=1}^d \frac{1}{u-\lambda_i}$$

$$\Phi_\ell(A) := \mathrm{tr}[(A - \ell I)^{-1}] = \sum_{i=1}^d \frac{1}{\lambda_i - \ell}.$$

7.2 The BSS Algorithm

We start by giving a brief overview of the BSS algorithm: the algorithm starts by maintaining two "barriers" of eigenvalues $u_0 = \frac{d}{\varepsilon}$ and $\ell_0 = -\frac{d}{\varepsilon}$. Iteratively, the algorithm searches for an index $i \in [m]$ such that the inner product between $v_i v_i^\top$ and a quantity related to lower barrier is large while the inner product related to barrier is small. Then we add this outer product $v_i v_i^\top$ with a scaling into the matrix A we are forming. After $\Theta(d/\varepsilon^2)$ iterations, we've constructed a matrix A with the property that $A \approx_\varepsilon I$.

We formalize the algorithm as follows:

Algorithm 6 BSS algorithm

procedure BSS($\{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$)
  $u_0 \leftarrow \frac{d}{\varepsilon}, \ell_0 \leftarrow -\frac{d}{\varepsilon}$
  $A_0 \leftarrow \mathbf{0}_{n\times n}$
  $T \leftarrow \frac{d}{\varepsilon^2}$
  $\delta_U \leftarrow 1, \delta_L \leftarrow \frac{1}{1+2\varepsilon}$
  for $i = 1 \to T$ **do**
    $u_t \leftarrow u_{t-1} + \delta_U, \ell_t \leftarrow \ell_{t-1} + \delta_L$
    $L_t \leftarrow \frac{(A_{t-1}-\ell_t I)^{-2}}{\Phi_{\ell_t}(A_{t-1})-\Phi_{\ell_{t-1}}(A_{t-1})} - (A_{t-1} - \ell_t I)^{-1}$
    $U_t \leftarrow \frac{(u_t I - A_{t-1})^{-2}}{\Phi_{u_{t-1}}(A_{t-1})-\Phi_{u_t}(A_{t-1})} + (u_t I - A_{t-1})^{-1}$
    Find an index j such that[5]

$$v_j^\top (L_t - U_t) v_j \geq 0$$

    $c \leftarrow \frac{v_j^\top (L_t + U_t) v_j}{2}$
    $A_t \leftarrow A_{t-1} + \frac{1}{c} \cdot v_j v_j^\top$
  end for
  return A_T/d
end procedure

The central lemma that guarantees the BSS algorithm to find a good sparsifier that satisfies both upper and lower bound is the following:

Lemma 7.3 (Lemma 3.5 of [BSS12]). *Suppose $A \in \mathbb{R}^{d\times d}$ satisfying $\ell I \prec A \prec uI$, let $\varepsilon \in (0,1)$ and suppose $\Phi_u(A) \leq \varepsilon, \Phi_\ell(A) \leq \varepsilon$, and $\varepsilon, \delta_U, \delta_L$ satisfying*

$$0 \leq \frac{1}{\delta_U} + \varepsilon \leq \frac{1}{\delta_L} - \varepsilon,$$

then we have

1. **Lower bounding lower barrier.**

$$\sum_{i=1}^m v_i^\top \left(\frac{(A - (\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A) - \Phi_\ell(A)} - (A - (\ell+\delta_L)I)^{-1}\right) v_i \geq \frac{1}{\delta_L} - \varepsilon.$$

2. **Upper bounding upper barrier.**

$$\sum_{i=1}^m v_i^\top \left(\frac{((u+\delta_U)I - A)^{-2}}{\Phi^u(A) - \Phi^{u+\delta_U}(A)} + ((u+\delta_U)I - A)^{-1}\right) v_i \leq \frac{1}{\delta_U} + \varepsilon.$$

We also record two lemmas that control the growth of lower and upper barriers.

Lemma 7.4 (Lemma 3.3 of [BSS12]). *Suppose $A \prec uI$ and $v \in \mathbb{R}^d$ is any vector. If*

$$c \geq v^\top \left(\frac{((u+\delta_U)I - A)^{-2}}{\Phi^u(A) - \Phi^{u+\delta_U}(A)} + ((u+\delta_U)I - A)^{-1}\right) v,$$

then

$$\Phi^{u+\delta_U}\left(A + \frac{1}{c}\cdot vv^\top\right) \leq \Phi^u(A) \text{ and } A + \frac{1}{c}\cdot vv^\top \prec (u+\delta_U)I.$$

Lemma 7.5 (Lemma 3.4 of [BSS12]). *Suppose $A \succ \ell I$, $\Phi_\ell(A) \le 1/\delta_L$ and $v \in \mathbb{R}^d$ is any vector. If*

$$0 < c \le v^\top \left(\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1}\right)v,$$

then

$$\Phi_{\ell+\delta_L}\left(A+\frac{1}{c}\cdot vv^\top\right) \le \Phi_\ell(A) \text{ and } A+\frac{1}{c}\cdot vv^\top \succ (\ell+\delta_L)I.$$

Combining the above 3 lemmas, we derive a corollary that justifies that in each iteration of Alg. 6, we can always find an index j satisfying the inequality on line 10:

Corollary 7.6. *Suppose $A \in \mathbb{R}^{d\times d}$ satisfying $\ell I \prec A \prec uI$, let $\varepsilon \in (0,1)$ and suppose $\Phi^u(A) \le \varepsilon, \Phi_\ell(A) \le \varepsilon$, and $\varepsilon, \delta_U, \delta_L$ satisfying*

$$0 \le \frac{1}{\delta_U} + \varepsilon \le \frac{1}{\delta_L} - \varepsilon,$$

then there exists an index $j \in [m]$ and a positive value c such that

1. **Witness of gap between lower and upper barriers.**

$$v_j^\top \left(\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1}\right)v_j \ge c \ge v_j^\top \left(\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1}\right)v_j.$$

2. **Spectral property.**

$$(\ell+\delta_L)I \prec A + \frac{1}{c}\cdot v_j v_j^\top \prec (u+\delta_U)I.$$

Moreover, if we further have

$$0 \le \frac{1}{\delta_U} + \varepsilon < \frac{1}{\delta_L} - \varepsilon,$$

then the witness of gap between lower and upper barriers has a strict inequality between the two quantities:

$$v_j^\top \left(\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1}\right)v_j > c > v_j^\top \left(\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1}\right)v_j.$$

Proof. By Lemma 7.3, we have the following:

$$\sum_{i=1}^m v_i^\top \left(\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1}\right)v_i \ge \frac{1}{\delta_L} - \varepsilon,$$

$$\sum_{i=1}^m v_i^\top \left(\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1}\right)v_i \le \frac{1}{\delta_U} + \varepsilon.$$

By an averaging argument, there must exist an index $j \in [m]$ that witnesses this inequality, i.e.,

$$v_j^\top (\frac{(A-(\ell+\delta_L)I)^{-2}}{\Phi_{\ell+\delta_L}(A)-\Phi_\ell(A)} - (A-(\ell+\delta_L)I)^{-1})v_j \geq v_j^\top (\frac{((u+\delta_U)I-A)^{-2}}{\Phi^u(A)-\Phi^{u+\delta_U}(A)} + ((u+\delta_U)I-A)^{-1})v_j.$$

The spectral property is guaranteed by Lemma 7.5 and Lemma 7.4.

For the strict inequality, note that if we have $\frac{1}{\delta_L} - \varepsilon > \frac{1}{\delta_U} + \varepsilon$, then by Lemma 7.3 and again by an averaging argument, we conclude that witness also exhibits a strict inequality. □

Remark 7.7. In the original statement of [BSS12], the strict inequality is not required. Here we prove a version for strict inequality since in order to use our Max-IP data structure, we have to make sure that the maximum inner product is strictly greater than 0. Fortunately, this condition can be guaranteed via choosing parameters δ_U, δ_L more carefully. We further note that this does not affect our solution qualitatively.

We also include a proof for the main Theorem of [BSS12] here, since we will need to later adapt our data structure for this problem.

Lemma 7.8 (Theorem 3.1 of [BSS12])**.** *Suppose we are given m vectors $v_1, \ldots, v_m \in \mathbb{R}^d$ satisfying $\sum_{i=1}^m v_i v_i^\top = I$, then there exists a deterministic algorithm (Alg. 6) can find scalars $\{s_i\}_{i=1}^m$ satisfying*

$$|\{s_i : s_i \neq 0\}| = O(d/\varepsilon^2),$$

such that

$$(1-\varepsilon)\cdot I \preceq \sum_{i=1}^m s_i v_i v_i^\top \preceq (1+\varepsilon)I.$$

The algorithm (Alg. 6) runs in time $O(md^3/\varepsilon^2)$.

Proof. We first prove the correctness. By the update rule of Alg. 6, we know that we maintain the following invariants across all iterations due to Lemma 7.5 and Lemma 7.4

$$\Phi_{u_t}(A_t) \leq \Phi_{u_{t-1}}(A_{t-1}) \text{ and } \Phi_{\ell_t}(A_t) \leq \Phi_{\ell_{t-1}}(A_{t-1}),$$

which means it suffices to examine $\Phi_{u_0}(A_0)$ and $\Phi_{\ell_0}(A_0)$ respectively, recall that we choose $u_0 = \frac{d}{\varepsilon}$, $\ell_0 = -\frac{d}{\varepsilon}$, hence we have

$$\begin{aligned}\Phi_{u_0}(A_0) &= \sum_{i=1}^d \frac{\varepsilon}{d} \\ &= \varepsilon, \\ \Phi_{\ell_0}(A_0) &= \sum_{i=1}^d \frac{\varepsilon}{d} \\ &= \varepsilon.\end{aligned}$$

To conclude the proof, we shall apply Lemma 7.3 for $T = \Theta(d/\varepsilon^2)$ times, so we verify the relations between $\varepsilon, \delta_U, \delta_L$:

$$\frac{1}{\delta_U} + \varepsilon = 1 + \varepsilon \geq 0,$$

$$\frac{1}{\delta_L} - \varepsilon = 1 + \varepsilon \geq \frac{1}{\delta_U} + \varepsilon.$$

This means we can apply Lemma 7.3 and have

$$(\ell_0 + T\delta_L)I \prec A_T \prec (u_0 + T\delta_U)I,$$

plug in the values of $\ell_0, u_0, \delta_L, \delta_U$ and T, we conclude that

$$(1 - \varepsilon - 2\varepsilon^2)I \prec A_T/d \prec (1 + \varepsilon)I.$$

Now we analyze the running time, the algorithm iterates for $T = \Theta(d/\varepsilon^2)$ iterations, and for each iteration t, we compute L_t and U_t in $O(d^\omega)$ time, and the search for index j takes $O(md^2)$ time. Hence, the total running time is $O(md^3/\varepsilon^2)$. □

7.3 Faster BSS via Max-IP Search

We note that in the vanilla BSS algorithm (Alg. 6), one has to compute a quantity for each $i \in [m]$ and search over all m such quantities. While this is fine in small m setting (such as for graph, m is at most $O(d^2)$), the running time becomes troublesome when m is large ($m \gg d^2$) and one is asked to compute a sparsifier at each round under a constant number of vector insertions and deletions. We will show in this section that if we are allowed to preprocess vectors, we can remove the linear dependence on m in the iterative process. To achieve this goal, we observe that at each iteration, the search for index j can be formulated as a Max-IP search problem: we wish to find an index j such that $\langle v_j v_j^\top, L_t - U_t \rangle$ is maximized. Note that this guarantees the condition $v_j^\top (L_t - U_t) v_j \geq 0$ to be satisfied by Lemma 7.3.

Roughly speaking, our algorithm will first preprocess m outer products $\{v_i v_i^\top\}_{i=1}^m$ into our Max-IP data structure, this takes $\widetilde{O}(m^{1+\rho}d^2 + \mathrm{nnz}(V) \cdot d^2)$ time. Then at each iteration, we find the valid index j by querying the Max-IP data structure, giving a cost per iteration of $\widetilde{O}(m^\rho + d^\omega)$. This gives an overall running time of $\widetilde{O}(m^{1+\rho}d^2 + \mathrm{nnz}(V) \cdot d^2 + \varepsilon^2 d(m^\rho + d^\omega))$. We remark that since we only require the index j to satisfy the non-negative condition: $\langle v_j v_j^\top, L_t - U_t \rangle \geq 0$, we can use a Max-IP data structure with lossy approximation factor, yielding an algorithm that has an almost-linear dependence on m in initialization phase and the cost per iteration is almost $\widetilde{O}(d^\omega)$.

We present our algorithm as follows:

Algorithm 7 BSS with Max-IP

procedure BSS($V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$)
  $u_0 \leftarrow \frac{d}{\varepsilon}, \ell_0 \leftarrow -\frac{d}{\varepsilon}$
  $A_0 \leftarrow \mathbf{0}_{n\times n}$
  $T \leftarrow \frac{d}{\varepsilon^2}$
  $\delta_U \leftarrow 1, \delta_L \leftarrow \frac{1}{1+3\varepsilon}$
  MaxIP MI
  $c \leftarrow \frac{1}{\varepsilon m^9}, \tau \leftarrow 0.9$
  MI.Init(d^2, m, V, c, τ) ▷ In the downstream ANN data structure, the dimension has been reduced to $\Theta(\log(m))$
  for $i = 1 \to T$ **do**
    $u_t \leftarrow u_{t-1} + \delta_U, \ell_t \leftarrow \ell_{t-1} + \delta_L$
    $L_t \leftarrow \frac{(A_{t-1}-\ell_t I)^{-2}}{\Phi_{\ell_t}(A_{t-1})-\Phi_{\ell_{t-1}}(A_{t-1})} - (A_{t-1} - \ell_t I)^{-1}$
    $U_t \leftarrow \frac{(u_t I - A_{t-1})^{-2}}{\Phi^{u_{t-1}}(A_{t-1})-\Phi^{u_t}(A_{t-1})} + (u_t I - A_{t-1})^{-1}$
    $q \leftarrow L_t - U_t$
    $v_j \leftarrow$ MI.Query(q)
    $c \leftarrow \frac{v_j^\top (L_t+U_t) v_j}{2}$
    $A_t \leftarrow A_{t-1} + \frac{1}{c} \cdot v_j v_j^\top$
  end for
  return A_T/d
end procedure

We first prove the correctness of our algorithm. The main challenge here is that we output an approximate estimate, and we need to make sure that this estimate v_j has the property that $v_j^\top (L_t - U_t) v_j > 0$.

Theorem 7.9 (Formal statement of (correctness part of) Theorem 2.5)**.** *Let A be the output of Alg. 7, then we have*

$$(1 - O(\varepsilon)) \cdot I \preceq A \preceq (1 + O(\varepsilon)) \cdot I.$$

Proof. We start by observing the following: since we choose δ_U as 1 and δ_L as $\frac{1}{1+3\varepsilon}$, we have that for any $t \in [T]$, the following holds:

$$\sum_{i=1}^m v_i^\top L_t v_i \geq 1 + 2\varepsilon,$$
$$\sum_{i=1}^m v_i^\top U_t v_i \leq 1 + \varepsilon.$$

This is due to Corollary 7.6. Combining these two inequalities yields

$$\sum_{i=1}^m v_i^\top (L_t - U_t) v_i \geq \varepsilon.$$

By a simple averaging argument, it is not hard to see that

$$\arg\max_{j\in[m]} v_j^\top (L_t - U_t) v_j \geq \frac{\varepsilon}{m},$$

we will use v_{j^*} to denote this vector.

By Theorem 5.11, we know that at each iteration, we are giving a point v_j with the following guarantee:

$$v_j^\top (L_t - U_t) v_j \geq c \cdot v_{j^*}^\top (L_t - U_t) v_{j^*} - \widetilde{\lambda},$$

recall we have chosen $c = \frac{1}{\varepsilon m^9}$, so $c \cdot v_{j^*}^\top (L_t - U_t) v_{j^*} \geq \frac{1}{m^{10}}$. It suffices to show that $\widetilde{\lambda} \leq \frac{1}{m^{10}}$. Per Theorem 5.11, we define $\widetilde{\lambda}$ as $O(\sqrt{\frac{1-c\tau}{1-\tau}}) \cdot (\lambda + \alpha)$, where λ is the diameter of the net we use when we are in the lower dimension after sketching, and hence it suffices to pick it as $O(\frac{1}{(ms)^{10}}) \leq O(\frac{1}{m^{10}})$ and α is the diameter of the net we use for Robust JLT (Lemma 4.12), and $\alpha = O(\frac{1}{(md^2)^{10}}) \leq O(\frac{1}{m^{10}})$. Finally, plug in the choice of c and τ, we conclude $O(\sqrt{\frac{1-c\tau}{1-\tau}}) = O(1)$. Hence, the additive error term $\widetilde{\lambda} = O(\frac{1}{m^{10}})$, as desired.

The above calculation reveals that at each iteration, we find an index $j \in [m]$ such that $v_j^\top (L_t - U_t) v_j \geq 0$, hence by Lemma 7.8, the output of our algorithm satisfies the desired guarantee. □

Now that the correctness has been guaranteed, we move to discuss the running time. We present the running time under two different tensor-oriented JLTs: TensorSRHT and TensorSparse.

Theorem 7.10 (Running time with TensorSRHT)**.** *If Alg. 7 is realized with* TensorSRHT *matrices, then the running time of the algorithm is*

- $$\widetilde{O}(m^{1+\rho} d^2 + m d^3 + \varepsilon^{-2} d (m^\rho + d^\omega)),$$

  where $\rho = \frac{(1-\tau)(1+\varepsilon_1)^2}{2 - 2c\tau - (1-\tau)(1+\varepsilon_1)^2} + o(1)$ *with* $\varepsilon_1 = 0.01$.

- $$\widetilde{O}(m^{1+o(1)} d^2 + m d^3 + \varepsilon^{-2} d (m^\rho + d^\omega)),$$

  where $\rho = \frac{2(1-\tau)(1+\varepsilon_1)^2}{1-c\tau} - \frac{(1-\tau)^2(1+\varepsilon_1)^4}{(1-c\tau)^2} + o(1)$ *with* $\varepsilon_1 = 0.01$.

Proof. We note that the precision parameter for JLT only affects the running time, for the sake of simplicity we set $\varepsilon = 0.01$ here. To use Theorem 5.11, we need to initiate k independent TensorSRHT matrices with $k = \widetilde{O}(d^2)$. Apply each TensorSRHT to each vector v_i and compute its approximate tensor product takes $\widetilde{O}(d)$ time, yielding an overall time of $\widetilde{O}(md^3)$ time. All points are then mapped into a much lower dimension $s = O(\log^3 m)$, so all our subsequent discussions are in this s dimension. We will then initialize $k \cdot \kappa = \widetilde{O}(d^2)$ Max-IP data structures, and the proprocessing time over all these data structures is $\widetilde{O}(m^{1+\rho} d^2)$ time (Bullet 1 of Theorem 5.11) or $\widetilde{O}(m^{1+o(1)} d^2)$ (Bullet 2 of Theorem 5.11).

We then consider the cost per iteration. For each round, we compute matrix L_t and U_t, which takes $O(d^\omega)$ time. Then, we will apply the TensorSRHT to $L_t - U_t$ for $O(\log(1/\delta))$ different sketches, which takes $\widetilde{O}(d^\omega)$ time. Finally, we send a low-dimensional query point to $\widetilde{O}(1)$ Max-IP data structures, which takes $\widetilde{O}(m^\rho)$ time. This concludes the overall running time analysis. □

A similar analysis can be conducted for TensorSparse. One nice property of TensorSparse is it has $O(\log m)$ sparsity at each column for constant ε and δ. This means for a TensorSparse matrix R, we have

- For $v \in \mathbb{R}^d$, compute $R(v \otimes v)$ in time $\widetilde{O}(\mathrm{nnz}(v))$;
- For $A \in \mathbb{R}^{d \times d}$, compute $R(\mathrm{vec}(A))$ in time $\widetilde{O}(\mathrm{nnz}(A))$.

This leads to the following result:

Theorem 7.11 (Formal statement of (running time part of) Theorem 2.5)**.** *Let $V = \{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$, if Alg. 7 is realized with* TensorSparse *matrices, then the running time of the algorithm is*

- $$\widetilde{O}(m^{1+\rho} d^2 + \mathrm{nnz}(V) \cdot d^2 + \varepsilon^{-2} d(m^\rho + d^\omega)),$$
  where $\rho = \frac{(1-\tau)(1+\varepsilon_1)^2}{2-2c\tau-(1-\tau)(1+\varepsilon_1)^2} + o(1)$ *with* $\varepsilon_1 = 0.01$.
- $$\widetilde{O}(m^{1+o(1)} d^2 + \mathrm{nnz}(V) \cdot d^2 + \varepsilon^{-2} d(m^\rho + d^\omega)),$$
  where $\rho = \frac{2(1-\tau)(1+\varepsilon_1)^2}{1-c\tau} - \frac{(1-\tau)^2(1+\varepsilon_1)^4}{(1-c\tau)^2} + o(1)$ *with* $\varepsilon_1 = 0.01$.

Proof. The proof is the same as Theorem 7.10 except replacing the running time of using TensorSRHT by TensorSparse. □

Remark 7.12. We now perform a computation on the exponent ρ. Use the first bullet of Theorem 5.11, we have $f(c,\tau) = \frac{(1-\tau)(1+\varepsilon)^2}{2-2c\tau-(1-\tau)(1+\varepsilon)^2}$, with the choice $\tau = 0.9$, $c = \frac{1}{\varepsilon m^9}$ and $\varepsilon = 0.01$, we have $f(c,\tau) = \frac{0.103}{1.8-\frac{1.8}{\varepsilon m}}$, if $1/(\varepsilon m) \to 0$, then $f(c,\tau) \in (0.05, 0.06)$.

If we are to use the second bullet to make the preprocessing time $m^{1+o(1)}$, then $f(c,\tau) = \frac{2(1-\tau)(1+\varepsilon)^2}{1-c\tau} - \frac{(1-\tau)^2(1+\varepsilon)^4}{(1-c\tau)^2}$, plug in the choice of $\tau = 0.9, c = \frac{1}{\varepsilon m^9}$ and $\varepsilon = 0.01$, we get $f(c,\tau) \approx 0.2$.

7.4 Max-IP-based BSS: Comparisons and Applications

We compare our Max-IP-based BSS algorithm with the almost-linear time randomized BSS algorithm proposed by Lee and Sun [LS15] and an SDP-based method [LS17] that has nearly-linear dependence on m but worse depedence on ε (runtime dependence from ε^{-2} to $\varepsilon^{-O(1)}$). We will later refer to their algorithm as LS15 and LS17 algorithm. We give a brief summary of these algorithms.

LS15 Algorithm. Compared to the vanilla BSS algorithm, LS15 algorithm combined ideas from [SS11] where they sample from a distribution defined by the barrier functions, and they sample a batch of vectors instead of one-by-one. By doing so, they can reuse the computations shared by various different vectors. The running time of LS15 algorithm is $\widetilde{O}(\varepsilon^{-2} q m d^{\omega-1+3/q})$ where $q \geq 10$ is a tuning parameter. It is worth noticing that the parameter q is directly related to the sparsity of the final output, i.e., the output $\sum_{i=1}^m s_i v_i v_i^\top$ has the property that $|\{s_i : s_i \neq 0\}| = O(\varepsilon^{-2} q d)$. Hence, to get a high quality sparsifier, one can only pick $q = O(1)$.

LS17 Algorithm. LS17 is an SDP-based algorithm that can also be generalized to sparsify sum-of-PSD-matrices, instead of using the standard barrier potential functions, they design novel potential functions that can be computed efficiently. Then, they show that by implementing an oracle for one potential function, one can reduce the original two potential functions argument to one. Further, they show that this oracle can be solved efficiently via a parallel SDP solver. For the sparsification of sum-of-rank-1-matrices task, they achieve a running time of $\widetilde{O}(\varepsilon^{-O(1)} \cdot (\mathrm{nnz}(X) + d^\omega))$, where $X = \{v_1 v_1^\top, \ldots, v_m v_m^\top\} (\mathbb{R}^{d^2})^m$.

Comparison with Our Method. We note that LS17 essentially achieves the (near) optimality of this problem in the static setting. However, this is no longer the case when we consider the *dynamic* setting, where at each timestamp, a constant number of vectors have been deleted from V and a constant number of vectors have been added into V to form the new V'. Then we are asked to output a sparsifier for the new set V'. One can view it as a generalization of the dynamic graph sparsifier problem [CGH+20, BBG+20]. Current known methods for this problem can only output a sparsifier with sub-optimal size, i.e., the sparsifier contains $\Theta(\varepsilon^{-2}d\log d)$ vectors. To achieve the *optimality* of the size, it seems unavoidable to use variants of BSS sparsifier.

Even with the fastest known algorithm for BSS sparisifier, i.e., LS17, one would pay $\Omega(m)$ time at each timestamp. In contrast, our algorithm avoids this linear dependence on m entirely: note that the Max-IP data structure does support insertion and deletion, hence, given these requests, one can first spend $\widetilde{O}((m^\rho + \mathrm{nnz}(V'\Delta V))d^2)$ inserting and deleting vectors, where $A\Delta B$ denote the symmetric difference between sets A and B. Then, one can simply run the iterative process to output a new sparisifier in time $\widetilde{O}(\varepsilon^{-2}d(m^\rho + d^\omega))$.

Our method is also robust against adaptive adversary for free: it follows naturally from the robustness of our JLT and Max-IP data structure. This can be viewed as by using a *locally robust* data structure to achieve the *global robustness* of a dynamic task. This is one clear advantage of our high level approach — i.e., using an efficient and robust data structure to solve an iterative optimization problem to achieve a lower cost per iteration and automatically gives robust guarantee in dynamic settings.

Applications. Besides graph sparsifiers, the BSS sparsifiers find their applications in constrained linear regression, multiple-response regression [BDMi13], matrix CUR decomposition [BW14, SWZ17] and tensor CURT decomposition [SWZ19]. Given a matrix $A \in \mathbb{R}^{m\times d}$, the goal is to select a subset of rows/columns of size $\Theta(k/\varepsilon^{-2})$, where k is the rank of A. One can view A as comprising m data points each of dimension d. In the dynamic setting, a constant number of data points have been removed and added, and we are asked to either compute a CUR decomposition or solve a regression task. Moreover, in such tasks, it is standard to have $m \gg d^2$, hence we cannot afford a runtime linear on m at each timestamp. Using our method, we remove the linear dependence on m for the subsequent updates.

Extensions to Sparsify PSD Matrices. We remark that our framework can be further extended to sparsify sum of PSD matrices, using the regret minimization approach introduced by Allen-Zhu, Liao and Orecchia [AZLO15]. Observe that their mirror descent algorithm can also be viewed as a variant of non-negative inner product search. We leave the task of generalizing our approach for sparsification of sum of PSD matrices as a future direction.

8 One-Sided Kadison-Singer via Min-IP Data Structure

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

- In Section 8.1, we formally define the one-sided Kadison-Singer problem.
- In Section 8.2, we prove the correctness of a greedy process with an approximate guarantee.
- In Section 8.3, we provide an analysis for a straightforward implementation of the greedy process.

- In Section 8.4, we use approximate Min-IP data structure to achieve both fast preprocessing and query time.

8.1 Problem Setup

We consider a simpler and one-sided version of the well-known Kadison-Singer problem studied by Weaver [Wea13], which is similar to the restricted invertibility problem [Sri10].

Question 8.1. *Does there exist a constant $N \in \mathbb{N}$, such that if $\{v_1, \ldots, v_m\} \in (\mathbb{R}^d)^m$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$, and*

$$\sum_{i=1}^{m} v_i v_i^\top = I,$$

then there exists a subset $S \subseteq \{1, \ldots, m\}$ such that for any $q \in (0, 1)$, we have

$$\| \sum_{i \in S} v_i v_i^\top \| \leq q - \frac{1}{\sqrt{N}}.$$

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

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

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.

8.2 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 8.2 (Approximate greedy lemma)**.** *Let $N \in \mathbb{R}_+$, if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and*

$$\sum_{i=1}^{m} v_i v_i^\top = I.$$

Then for any $n < m$ and any unit vector u, we can find a set S ($|S| = n$) such that

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

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 $i_1, \ldots, i_n$ such that the following two things hold

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

where Φ^a is the upper barrier potential as in Def. 7.2.

Assume the above two conditions hold, then we will have

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

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

$$\mathrm{tr}[(a_{j+1} I - T_j)^{-1}(I - \beta T_j)] = \mathrm{tr}[Q_j^{-1}(a_{j+1} I - D_j)^{-1}(I - \beta D_j) Q_j]$$

$$
\begin{aligned}
&= \mathrm{tr}[(a_{j+1}I - D_j)^{-1}(I - \beta D_j)] \\
&= \sum_{l=1}^{d} \frac{1}{a_{j+1} - \lambda_l}(1 - \beta\lambda_l) \\
&\le \frac{1}{d}\sum_{l=1}^{d} \frac{1}{a_{j+1} - \lambda_l} \sum_{l=1}^{d}(1 - \beta\lambda_l) \\
&= \frac{1}{d} \cdot \mathrm{tr}[(a_{j+1}I - T_j)^{-1}] \cdot \mathrm{tr}[I - \beta T_j] \\
&\le \frac{1}{d} \cdot \mathrm{tr}[(a_j I - T_j)^{-1}] \cdot \mathrm{tr}[I - \beta T_j] \\
&= \frac{1}{d}\Phi^{a_j}(T_j) \cdot \mathrm{tr}[I - \beta T_j] \\
&\le \frac{1}{d}\Phi^{a_0}(T_0) \cdot \mathrm{tr}[I - \beta T_j] \\
&= \sqrt{N} \cdot \mathrm{tr}[I - \beta T_j], \qquad (3)
\end{aligned}
$$

where the fourth step follows from sorting inequality 3.5, the sixth step follows from $a_{j+1} > a_j$, the eighth step follows from the inductive hypothesis.

Consequently, we have

$$
\begin{aligned}
\Phi^{a_j}(T_j) - \Phi^{a_{j+1}}(T_j) &= \mathrm{tr}[(a_j I - T_j)^{-1} - (a_{j+1}I - T_j)^{-1}] \\
&= \mathrm{tr}[Q_j^{-1}(a_j I - D_j)^{-1}Q_j - Q_j^{-1}(a_{j+1}I - D_j)^{-1}Q_j] \\
&= \mathrm{tr}[(a_j I - D_j)^{-1} - (a_{j+1}I - D_j)^{-1}] \\
&= (a_{j+1} - a_j)\mathrm{tr}[(a_j I - D_j)^{-1}(a_{j+1}I - D_j)^{-1}] \\
&= (1 + \frac{1}{\sqrt{N} - 1})\frac{1}{m} \cdot \mathrm{tr}[(a_j I - T_j)^{-1}(a_{j+1}I - T_j)^{-1}] \\
&\ge (1 + \frac{1}{\sqrt{N} - 1})\frac{1}{m} \cdot \mathrm{tr}[(a_{j+1}I - T_j)^{-2}] \qquad (4)
\end{aligned}
$$

where the forth step follows from Fact 3.4, 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

$$
\begin{aligned}
\mathrm{tr}[(a_{j+1}I - T_j)^{-2}(I - \beta T_j)] &= \sum_{l=1}^{d} \frac{1}{(a_{j+1} - \lambda_l)^2}(1 - \beta\lambda_l) \\
&\le \frac{1}{d}\sum_{l=1}^{d} \frac{1}{(a_{j+1} - \lambda_l)^2} \sum_{l=1}^{d}(1 - \beta\lambda_l) \\
&= \frac{1}{d}\mathrm{tr}[(a_{j+1}I - T_j)^{-2}] \cdot \mathrm{tr}[I - \beta T_j]. \qquad (5)
\end{aligned}
$$

Combining Eq. (4) and (5), we get

$$
\frac{\mathrm{tr}[(a_{j+1}I - T_j)^{-2}(I - \beta T_j)]}{\Phi^{a_j}(T_j) - \Phi^{a_{j+1}}(T_j)} \le \frac{m}{d}\frac{\sqrt{N} - 1}{\sqrt{N}} \cdot \mathrm{tr}[I - \beta T_j]
$$

$$= N(1 - \frac{1}{\sqrt{N}}) \cdot \mathrm{tr}[I - \beta T_j], \tag{6}$$

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

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

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

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. (3) and (6). The last step follows from $\mathrm{tr}[\beta T_j] = j/N$ 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 7.4, we conclude $\|T_{j+1}\| < a_{j+1}$ and $\Phi^{a_{j+1}}(T_{j+1}) \leq \Phi^{a_j}(T_j)$. □

Remark 8.3. 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.

8.3 An $O(nmd^\omega)$ Implementation

Note that Algorithm 8 is a straightforward implementation of the process derived from the proof of Lemma 8.2.

Theorem 8.4. *Let $N \in \mathbb{N}_+$, if $\{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m v_i v_i^\top = I$. Then for any $n < m$, there exists a deterministic algorithm that takes time $O(nmd^\omega$ to find a set S with cardinality n such that*

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

Proof. The correctness proof is straightforward, since Algorithm 8 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 $O(d^\omega)$ time, and compute the quantity in

Algorithm 8 Vanilla greedy algorithm derived from [Wea13], it takes $nm \cdot \mathcal{T}_{\mathrm{mat}}(d,d,d)$ time.

procedure VANILLAGREEDY($\{v_1, \ldots, v_m\}, N, n$) ▷ Theorem 8.4
  $T_0 \leftarrow \mathbf{0}_{d \times d}$
  $S \leftarrow \emptyset$
  for $j = 0 \to n$ **do**
    $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
  end for
  for $j = 0 \to n$ **do**
    for $i \in [m] \setminus S$ **do**
      $c_i \leftarrow (\Phi^{a_{j-1}}(T_j) - \Phi^{a_j}(T_j))^{-1} \cdot v_i^\top (a_j I - T_j)^{-2} v_i + v_i^\top (a_j I - T_j)^{-1} v_i$ ▷ $\mathcal{T}_{\mathrm{mat}}(d,d,d)$ time
    end for
    $i^* = \arg\min_{i \in [m] \setminus S} c_i$
    $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
    $S \leftarrow S \cup \{i^*\}$
  end for
  return S
end procedure

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

$$O(nmd^\omega).$$

□

8.4 One-Sided Kadison-Singer with Min-IP 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 8.5 (Formal version of Theorem 2.6)**.** *Let $\tau, c, \delta \in (0,1)$ and $N \in \mathbb{N}_+$, if $V = \{v_1, \ldots, v_m\}$ is a finite sequence of vectors in $\mathbb{R}^d$ satisfying $\|v_i\|_2 = \frac{1}{\sqrt{N}}, \forall i \in [m]$ and $\sum_{i=1}^m v_i v_i^\top = I$. Then for any $n < m$, there exists a randomized algorithm that takes time $\mathcal{T}$ to find a set S ($|S| = n$) such that with probability at least $1 - \delta$,*

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

Further, we have

- *If $c \in (\tau, \frac{8\tau}{7+\tau})$, then $\mathcal{T} = \widetilde{O}((m^{1.5} + \mathrm{nnz}(V))d^2 + n(\sqrt{m}d^2 + d^\omega))$;*
- *If $c \in (\tau, \frac{400\tau}{399+\tau})$, then $\mathcal{T} = \widetilde{O}((m^{1.01} + \mathrm{nnz}(V))d^2 + n(m^{0.01}d^2 + d^\omega))$.*

Proof. Note that since we are using the approximate Min-IP data structure with parameter c and τ, we are promised to get an index i^* such that

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

Algorithm 9 Our Approximate Min-IP data structure implementation.

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 8.5
  $T_0 \leftarrow \mathbf{0}_{d \times d}$
  $S \leftarrow \emptyset$
  Construct V ▷ $V = [v_1, v_2, \cdots, v_m]$
  for $j = 0 \to n$ **do**
    $a_j = \frac{1}{\sqrt{N}} + (1 + \frac{1}{\sqrt{N}-1})\frac{j}{m}$
  end for
  MINIP MI
  MI.INIT(d^2,m,V,c,τ) ▷ The dimension input to the Min-IP has been reduced by JLT
  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 d^ω time
    $N_j \leftarrow (a_{j-1} I - T_j)^{-1}$ ▷ $N_j \in \mathbb{R}^{d \times d}$, it takes d^ω time
    $q \leftarrow \mathrm{vec}((\mathrm{tr}[N_j] - \mathrm{tr}[M_j])^{-1} M_j M_j + M_j))$
    $i^* \leftarrow$ MI.QUERY(q)
    $T_{j+1} \leftarrow T_j + v_{i^*} v_{i^*}^\top$
    $S \leftarrow S \cup \{i^*\}$
    MI.DELETE($\mathrm{vec}(v_{i^*} v_{i^*}^\top)$)
  end for
  return S
end procedure

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

i.e., we obtain an index with $\frac{1}{c}$ approximation guarantee. As we showed in Theorem 8.2, 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 6.14 combined with the transformation illustrated in 5.7, we complete the proof of correctness of the data structure.

Now, we prove the correctness of the running time, which follows directly from Theorem 6.14. Note that it would incur an additive $\frac{\tau}{c}$ to the guarantee of inner product, which means the quality of approximation becomes $\frac{2}{c}$, with a success probability at least $1 - \delta$.

This completes the proof. □

Remark 8.6. 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.

9 Experimental Design via Min-IP Data Structure

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

- In Section 9.1, we introduce definitions and formally state the problem.
- In Section 9.2, we state some useful facts and tools for later proofs.
- In Section 9.3, we present our algorithm with Min-IP data structure.
- In Section 9.4, we prove an approximate regret lemma, which will provide a lower bound on the eigenvalue.
- In Section 9.5, we state the lemma that justifies the correctness of our algorithm.
- In Section 9.6, we prove the minimum inner product part of swapping algorithm.
- In Section 9.7, we prove the maximum inner product part of swapping algorithm.
- In Section 9.8, we discuss the implication of the swapping lemma and guide our algorithm design.
- In Section 9.9, we prove the main result of this section.

9.1 Definitions and Problem Setup

Definition 9.1. 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 9.2. 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 9.3. 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 9.4. 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:

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

Definition 9.5. 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 9.8, these two definitions are equivalent.

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

Question 9.6. *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 $\varepsilon \in (0, \frac{1}{\gamma}]$. Does there exist a subset $S \subset [m]$ with $|S| \leq n$ such that*

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

9.2 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 9.7 (Lemma 2.7 in [AZLSW20]). *Let $\Delta_{d\times d}$ be defined as Definition 9.1. 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 9.8 (Claim 2.9 in [AZLSW20]). *Let $\widetilde{A}_t, A_t \in \mathbb{R}^{d\times d}$ be the matrices defined in Def. 9.4, 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 9.9 (Claim 2.10 in [AZLSW20]). *Let $\Delta_{d\times d}$ be defined as in Definition 9.1. 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 9.10 (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*

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

9.3 Algorithm

Algorithm 10 Swapping algorithm with Min-IP data structure

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 9.16
  $\alpha \leftarrow \sqrt{d}\beta/\varepsilon$ and $T \leftarrow n/\varepsilon$
  $X \leftarrow X(X^\top \mathrm{diag}(\pi) X)^{-1/2}$ ▷ Whitening
  $S_0 \subseteq [m]$ be an arbitrary subset of support n
  $t \leftarrow 1$
  /* Initialize data structure with S_0 */
  MINIP MI
  MI.INIT(d^2, m, X, c, τ)
  while $t \leq T$ and $\lambda_{\min}(\sum_{i\in S_{t-1}} x_i x_i^\top) \leq 1 - \gamma\varepsilon$ **do**
    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
    $A_t \leftarrow (c_t I_d + \alpha \sum_{i\in S_{t-1}} x_i x_i^\top)^{-2}$
    $q \leftarrow \mathrm{vec}(\frac{A_t}{(1-\varepsilon)/n} + 2\alpha A_t^{1/2})$
    /* Query q */
    $i_t \leftarrow$ MI.QUERY(q)
    $j_t \leftarrow \arg\max_{j\in \overline{S}_{t-1}} B^+(x_j)$ ▷ Def. 9.12 with $\frac{1}{c}$ as β
    $S_t \leftarrow S_{t-1} \cup \{j_t\} \setminus \{i_t\}$
    $t \leftarrow t + 1$ ▷ Increase the counter
    /* Updating data structure by swapping j_t and i_t */
    MI.DELETE($x_{i_t} x_{i_t}^\top$)
    MI.INSERT($x_{j_t} x_{j_t}^\top$)
  end while
  return S_{t-1}
end procedure

9.4 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 9.11 (Approximate regret lemma)**.** *Let $\beta \geq 1$. Suppose $F_t = u_t u_t^\top - v_t v_t^\top$ for vectors $u_t, v_t \in \mathbb{R}^d$ and $A_0, \ldots, A_{T-1} \in \Delta_{d\times d}$ are defined in Def. 9.5 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 $U \in \Delta_{d\times d}$,*

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

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 9.8 and 9.9. The proof relies on the mirror descent matrices $\widetilde{A}_t$ and A_t we defined Def. 9.4, we need to modify the definition of $\widetilde{A}_t$ with $\overline{\alpha}$ instead of α. Per Claim 9.8, 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

$$\begin{aligned}
\langle \alpha F_{t-1}, A_{t-1} - U\rangle &= \langle \nabla\psi(A_{t-1}) - \nabla\psi(\widetilde{A}_t), A_{t-1} - U\rangle \\
&= \Delta_\psi(A_{t-1}, U) - \Delta_\psi(\widetilde{A}_t, U) + \Delta_\psi(\widetilde{A}_t, A_{t-1}) \\
&\leq \Delta_\psi(\widetilde{A}_{t-1}, U) - \Delta_\psi(\widetilde{A}_t, U) + \Delta_\psi(\widetilde{A}_t, A_{t-1}). \qquad (7)
\end{aligned}$$

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

$$\begin{aligned}
\Delta_\psi(\widetilde{A}_t, A_{t-1}) &= \psi(A_{t-1}) - \psi(\widetilde{A}_t) - \langle \nabla\psi(\widetilde{A}_t), A_{t-1} - \widetilde{A}_t\rangle \\
&= -2\mathrm{tr}[A_{t-1}^{-1/2}] + 2\mathrm{tr}[\widetilde{A}_t^{1/2}] + \langle \widetilde{A}_t^{-1/2}, A_{t-1} - \widetilde{A}_t\rangle \\
&= \langle \widetilde{A}_t^{-1/2}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - 2\mathrm{tr}[A_{t-1}^{1/2}] \\
&= \langle A_{t-1}^{-1/2} + \overline{\alpha} F_{t-1}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - 2\mathrm{tr}[A_{t-1}^{1/2}] \\
&= \overline{\alpha}\langle F_{t-1}, A_{t-1}\rangle + \mathrm{tr}[\widetilde{A}_t^{1/2}] - \mathrm{tr}[A_{t-1}^{1/2}]. \qquad (8)
\end{aligned}$$

Combining Eqs. (7) and (8) and telescoping t from 1 to T yields

$$\begin{aligned}
-\overline{\alpha}\sum_{t=0}^{T-1}\langle F_t, U\rangle &\leq \Delta_\psi(A_0, U) - \Delta_\psi(\widetilde{A}_T, U) + \sum_{t=0}^{T-1}\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}] \\
&\leq \Delta_\psi(A_0, U) + \sum_{t=0}^{T-1}\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}], \qquad (9)
\end{aligned}$$

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 $\sqrt{\overline{\alpha}}[u_t \quad v_t] \in \mathbb{R}^{d\times 2}$ 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. 3.3), we have

$$\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] = \mathrm{tr}[(A_t^{-1/2} + P_t J P_t^\top)^{-1}] = \mathrm{tr}[A_t^{1/2} - A_t^{1/2} P_t (J + P_t^\top A_t^{1/2} P_t)^{-1} P_t^\top A_t^{1/2}]. \qquad (10)$$

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 9.9 as a lower bound:

$$\begin{aligned}
\mathrm{tr}[\widetilde{A}_{t+1}^{1/2}] - \mathrm{tr}[A_t^{1/2}] = & -\mathrm{tr}[-A_t^{1/2} P_t (J + P_t^\top A_t^{1/2} P_t)^{-1} P_t^\top A_t^{1/2}] \\
\leq & -\mathrm{tr}[-A_t^{1/2} P_t (J + \mathrm{diag}(2\overline{\alpha} u_t^\top A_t^{1/2} u_t, 2\overline{\alpha} v_t^\top A_t^{1/2} v_t))^{-1} P_t^\top A_t^{1/2}] \\
= & -\frac{\overline{\alpha} u_t^\top A_t u_t}{1 + 2\overline{\alpha} u_t^\top A_t^{1/2} u_t} + \frac{\overline{\alpha} v_t^\top A_t v_t}{1 - 2\overline{\alpha} v_t^\top A_t^{1/2} v_t}. \qquad (11)
\end{aligned}$$

Plugging Eq. (11) into Eq. (9), we arrive at the desired result:

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

□

9.5 Approximate Swapping Lemma

The goal of this section is to present and prove Lemma 9.13. We start with a helpful definition.

Definition 9.12 (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 9.13. *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 $\mathrm{tr}[A] = 1$. 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 9.10, 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

$$d + \alpha\sqrt{d} = (1 + \frac{\beta}{\varepsilon}) d \ , \sqrt{d}/\alpha = \frac{\varepsilon}{\beta}. \qquad (12)$$

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 9.14 and Claim 9.15. □

9.6 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 9.14 (Part 1 of Lemma 9.13)**.** *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

$$\sum_{i\in S} 2\alpha\langle A^{1/2}, x_i x_i^\top\rangle = 2\alpha\langle A^{1/2}, Z\rangle \ge |S| = n. \tag{13}$$

On the other hand, because $Z \succeq 0$ and $\lambda_{\min}(Z) < 1$, invoking Claim 9.10 we get

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

which contradicts Eq. (13) 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 9.10; 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.

□

9.7 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 9.15 (Part 2 of Lemma 9.13)**.** *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

$$\sum_{j\in\overline{S}} \pi_j \langle A, x_j x_j^\top \rangle \geq t \cdot \sum_{j\in\overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle), \tag{14}$$

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,

$$\begin{aligned}
&\sum_{j\in\overline{S}} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \sum_{j\in\overline{S}} \pi_j (\beta + 2\alpha \langle A^{1/2}, x_j x_j^\top \rangle) \\
&\geq \mathrm{tr}[A] - \sum_{j\in S} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \beta \cdot (n - \sum_{j\in S} \pi_j) - 2\alpha t \cdot \mathrm{tr}[A^{1/2}] \\
&\geq 1 - \sum_{j\in S} \pi_j \langle A, x_j x_j^\top \rangle - t \cdot \beta \cdot (n - \sum_{j\in S} \pi_j) - 2\alpha t \sqrt{d} \\
&= 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j\in S} \pi_j (\langle A, x_j x_j^\top \rangle - t\beta) \\
&\geq 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j\in S} \max\{\langle A, x_j x_j^\top \rangle - t\beta, 0\} \\
&= 1 - t\beta n - 2t\alpha\sqrt{d} - \sum_{j\in S} (\langle A, x_j x_j^\top \rangle - t\beta) - \sum_{j\in S} \max\{(t\beta - \langle A, x_j x_j^\top \rangle), 0\} \\
&\geq 1 - 2t\alpha\sqrt{d} - \sqrt{d}/\alpha - \lambda_{\min}(\sum_{j\in S} x_j x_j^\top) - \sum_{j\in S} \max\{(t\beta - \langle A, x_j x_j^\top \rangle), 0\} \\
&\geq (\gamma - \beta)\varepsilon - \frac{2d}{\varepsilon n} - \sum_{j\in S} \max\{t\beta - \langle A, x_j x_j^\top \rangle, 0\}
\end{aligned} \tag{15}$$

where the second step follows from Fact 3.2 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 9.10. 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 9.10 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

$$\begin{aligned}
\sum_{j\in S'} \max\{\beta t - \langle A, x_j x_j^\top\rangle, 0\} &= \sum_{j\in S'}(\beta t - \langle A, x_j x_j^\top\rangle) \\
&= \beta(t-\nu)|S'| + \sum_{j\in S'}(\beta\nu - \langle A, x_j x_j^\top\rangle) \\
&\le \beta(t-\nu)n + 2\nu(d + \alpha\sqrt{d}) \\
&\le \varepsilon + \frac{4d/\varepsilon}{n} \qquad (16)
\end{aligned}$$

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.(15) and (16) 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. (14) and thus also the proof of Part 2. □

9.8 Implication of Swapping Lemma

Note that Lemma 9.13 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. 10, 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 Max-IP search. Though it can also be realized by our Max-IP data structure, the two approximation factor $c_{\text{Max-IP}}$ and $c_{\text{Min-IP}}$ would impose a restriction on the relationship between them and ε. Also, due to such precision requirement, one can not use a lossy estimation for Max-IP data structure as in the BSS case. Hence, we only present an algorithmic result that deals with one-side of the sets. We will discuss how to implement the two data structures paradigm towards the end of next section.

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.

9.9 Main Result

In this section, we present the correctness and runtime analysis of Algorithm 10. The correctness follows from the approximate regret and swap lemma, while the runtime comes from the approximate Min-IP data structure.

Theorem 9.16 (Formal version of Theorem 2.7)**.** *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:

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

$$\widetilde{O}(\mathcal{T}_{\text{mat}}(m,d,d) + n^{1.5} d^2 + \frac{n}{\varepsilon^2} \cdot (d^\omega + \sqrt{n} d^2 + (m-n) \cdot d^2)).$$

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

$$\widetilde{O}(\mathcal{T}_{\text{mat}}(m,d,d) + n^{1.01} d^2 + \frac{n}{\varepsilon^2} \cdot (d^\omega + n^{0.01} d^2 + (m-n) \cdot d^2)).$$

Proof. We will show Alg. 10 satisfies the properties in the theorem statement. Similar to the proof of Theorem 8.5, 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

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

Set $\beta = \frac{1}{c}$, note this is equivalent to find an index i satisfying $B^-(x_i) \leq \frac{1-\varepsilon}{n}$.

On the other hand, we can search the index $j \in \overline{S}_{t-1}$ such that

$$B^+(x_j) \geq \frac{1}{\beta n}$$

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

$$B^-(x_{i_t}) - B^+(x_{j_t}) \leq \frac{(\beta - 1) - \beta\varepsilon}{\beta n}.$$

We can pick $\beta = 1 + \varepsilon$, this would yield

$$B^-(x_{i_t}) - B^+(x_{j_t}) \leq -\frac{\varepsilon^2}{n}$$

Combining this fact with Lemma 9.11 and Claim 9.7, we have

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

$$\leq -T \cdot \frac{\varepsilon^2}{n} + 2\varepsilon,$$

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 $T = \frac{n}{\varepsilon^2}$, 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 $\widetilde{O}((n^{1.5} + \mathrm{nnz}(X))d^2)$ or $\widetilde{O}((n^{1.01} + \mathrm{nnz}(X))d^2)$ based on the choice of c (Theorem 6.14);

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 d^ω to form the matrix and compute its trace;
- The time of querying data structure is either $\widetilde{O}(\sqrt{n})$ or $\widetilde{O}(n^{0.01})$;
- The brute force search for j takes $O((m-n) \cdot d^2)$ if we pre-compute A_t and $A_t^{1/2}$;
- The insertion and deletion takes $\widetilde{O}((n^\rho + d)d^2)$ time.

This concludes the proof of running time. □

Remark 9.17. 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, we can choose β to a proper value to guarantee that we can use $\Theta(n/\varepsilon^2)$ instead of $\Theta(n/\varepsilon)$ iterations to converge. When ε is a constant, this does not affect the convergence rate much. 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.

One might consider to use Max-IP data structure for searching in set $\overline{S}$, but it turns out that the approximation factors of these two problems would cause problem and impose extra restrictions on the setting. Suppose $c_{\text{Max-IP}} \in (0, 1)$ is the approximation for Max-IP and $c_{\text{Min-IP}} = \beta > 1$ is its counter part for Min-IP. Use merely Min-IP, one get an index i with the property that $B^-(x_i) \leq \frac{1-\varepsilon}{n}$, if one would use Max-IP, then she would yield an index j such that $B^+(x_j) \geq \frac{c_{\text{Max-IP}}}{c_{\text{Min-IP}}} \cdot \frac{1}{n}$. In order

for the algorithm to progress, it instructive to have $\frac{c_{\text{Max-IP}}}{c_{\text{Min-IP}}} \geq 1 - \varepsilon$. Suppose there exists $a > 0$ such that $\frac{c_{\text{Min-IP}} - c_{\text{Max-IP}}}{c_{\text{Min-IP}} - a} \leq \varepsilon$, then we can guarantee that

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

However, this would require to pick both $c_{\text{Min-IP}}$ small and $c_{\text{Max-IP}}$ large. While selecting a small $c_{\text{Min-IP}}$ has a relatively benign impact on the running time of Min-IP, choosing a large $c_{\text{Max-IP}}$ would have a much worse influence on the running time of Max-IP. We leave combining both Min-IP and Max-IP for improving this swapping algorithm as a future direction.

References

[AC06] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *STOC*, pages 557—-563, 2006.

[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.

[AIL+15] Alexandr Andoni, Piotr Indyk, TMM Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1225–1233. Curran Associates, 2015.

[AINR14] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1018–1028. SIAM, 2014.

[AIR18] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3287–3318. World Scientific, 2018.

[AKK+20] Thomas D Ahle, Michael Kapralov, Jakob BT Knudsen, Rasmus Pagh, Ameya Velingker, David P Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 141–160. SIAM, 2020.

[ALRW17] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 47–66. SIAM, 2017.

[AM93] Sunil Arya and David M Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280. Citeseer, 1993.

[ANW14] Haim Avron, Huy Nguyen, and David Woodruff. Subspace embeddings for the polynomial kernel. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2258–2266. 2014.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 793–801, 2015.

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[AZLO15] Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. STOC '15, 2015.

[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.

[BBG+20] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary, 2020.

[BDMi13] Christos Boutsidis, Petros Drineas, and Malik Magdon-ismail. Near-optimal coresets for least-squares regression. *IEEE Transactions on Information Theory*, 2013.

[BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\widetilde{O}(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, 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.

[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.

[BW14] Christos Boutsidis and David P. Woodruff. Optimal cur matrix decompositions. STOC '14, 2014.

[C+06] Kenneth L Clarkson et al. Nearest-neighbor searching and metric space dimensions. *Nearest-neighbor methods for learning and vision: theory and practice*, pages 15–59, 2006.

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

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, 2002.

[CGH+20] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *61st Annual IEEE Symposium on Foundations of Computer Science*, November 2020.

[CJN18] Michael B Cohen, TS Jayram, and Jelani Nelson. Simple analyses of the sparse johnson-lindenstrauss transform. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[Cla83] Kenneth L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, FOCS '83, 1983.

[Cla97] Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, 1997.

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

[CN21] Yeshwanth Cherapanamjeri and Jelani Nelson. Terminal embeddings in sublinear time. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '21)*, 2021.

[CW79] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[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.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry (SoCG)*, pages 253–262, 2004.

[DIRW19] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations*, 2019.

[DJS+19] Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David P. Woodruff. Optimal sketching for kronecker product regression and low rank approximation. In *NeurIPS*, 2019.

[DKS10] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlos. A sparse johnson: Lindenstrauss transform. STOC '10, 2010.

[DSSW18] Huaian Diao, Zhao Song, Wen Sun, and David Woodruff. Sketching for kronecker product regression and p-splines. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1299–1308. PMLR, 2018.

[EFN17] Michael Elkin, Arnold Filtser, and Ofer Neiman. Terminal embeddings. *Theoretical Computer Science*, 2017.

[FL19] Casper Benjamin Freksen and Kasper Green Larsen. On using toeplitz and circulant matrices for johnson–lindenstrauss transforms. *Algorithmica*, 82:338–354, 2019.

[HW71] D. L. Hanson and F. T. Wright. A bound on tail probabilities for quadratic forms in independent random variables. *The Annals of Mathematical Statistics*, 42(3):1079–1083, 1971.

[HW87] David Haussler and Emo Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 604–613, 1998.

[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.

[IW18] Piotr Indyk and Tal Wagner. Approximate nearest neighbors in limited space. In *Conference On Learning Theory*, pages 2012–2036. PMLR, 2018.

[JL84] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[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.

[KLM+] M. Kapralov, Y. T. Lee, C. N. Musco, C. P. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477.

[KMM+20] Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. *Fast and Space Efficient Spectral Sparsification in Dynamic Streams*. 2020.

[KN10] Daniel M Kane and Jelani Nelson. A derandomized sparse johnson-lindenstrauss transform. 2010.

[KN14] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[KNST19] Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. Dynamic streaming spectral sparsification in nearly linear time and space. *ArXiv*, abs/1903.12150, 2019.

[KW14] Michael Kapralov and David Woodruff. Spanners and sparsifiers in dynamic streams. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC)*, 2014.

[LG14] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation (ISSAC)*, pages 296–303. ACM, 2014.

[LMWY20] Kasper Green Larsen, Tal Malkin, Omri Weinstein, and Kevin Yeo. *Lower Bounds for Oblivious Near-Neighbor Search*, pages 1116–1134. 2020.

[LN17] Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 633–638. IEEE, 2017.

[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.

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

[MMMR18] Sepideh Mahabadi, Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Nonlinear dimension reduction via outer bi-lipschitz extensions. In *STOC*, 2018.

[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.

[NN19] Shyam Narayanan and Jelani Nelson. Optimal terminal dimensionality reduction in euclidean space. In *STOC*, 2019.

[OvL81] Mark H Overmars and Jan van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Information Processing Letters*, 12(4):168–173, 1981.

[Pag13] Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–17, 2013.

[PT12] Mihai Pundefinedtraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 2012.

[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.

[Sri10] Nikhil Srivastava. *Spectral Sparsification and Restricted Invertibility*. PhD thesis, USA, 2010.

[SS11] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[SWYZ21] Zhao Song, David P. Woodruff, Zheng Yu, and Lichen Zhang. Fast sketching of polynomial kernels of polynomial degree. In *ICML*, 2021.

[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.

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

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 887–898. ACM, 2012.

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

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

[WZ20] David P Woodruff and Amir Zandieh. Near input sparsity time kernel embeddings via adaptive sampling. In *ICML*, 2020.