Dynamic SFT with Structured Measurements: Fast Queries, Fast Updates, Provable Guarantees

Yang Cao*, Zhao Song†

The sparse Fourier transform typically proceeds in two stages: frequency estimation and signal estimation. The first recovers the set of frequencies from noisy time-domain samples; the second constructs their corresponding magnitudes. In most methods, signal estimation is only approximate and depends on the frequencies identified in the first stage. In this paper, we study a complementary question: given access to an oracle that returns the exact magnitude for any queried frequency, what is the minimum number of oracle calls needed to perform a sparse Fourier transform? For an n-point discrete Fourier transform, the naive approach queries all n frequencies. We design the first algorithm that requires only $o(n)$ oracle invocations. We further complement this upper bound with a lower bound, derived using tools from computational complexity.

1. Introduction

Fourier transform is a fundamental tool in signal processing, engineering, theoretical computer science, etc. Over the years of development, Fourier transform has diverse variations and profound applications, among which Discrete Fourier transform is one of the major trends due to the discrete nature of practical signals. The Discrete Fourier transform is defined as follows:

$$\widehat{x}_f = \frac{1}{\sqrt{n}} \sum_{t=1}^{n} x_t \exp(\frac{-2\pi \mathbf{i} f t}{n}), \text{ where } f \in [n], \tag{1}$$

where $x \in \mathbb{C}^n$ represents the input signal, $\widehat{x} \in \mathbb{C}^n$ represents the Fourier transform of x.

A central problem in the field of Discrete Fourier transform is how many resources we need to perform Discrete Fourier transform [1–3]. Over the last century, well-known algorithms such as Fast Fourier Transform [1] already showed that $n \log(n)$ time is enough for performing Discrete Fourier transform and obtaining the precise magnitudes in the frequency domain.

However, we can do better than FFT under certain situations. In many applications [4], the reason for using Discrete Fourier transform is that the frequency domain is sparse. In fact, the assumption that the real signal is sparse under Fourier basis decomposition is widely recognized in signal processing. A line of works [2, 3, 5–8] are centered around Sparse Fourier transform, where they study Fourier transform when the frequency domain is sparse. From a computational aspect, sparse Fourier transform concerns not only the time complexity but also the sample complexity of reconstructing the signal [2, 3]. Here the sample complexity means the number of noisy samples that the algorithm need to reconstruct frequency domain.

Sparse Fourier transform can be viewed as a special compressed sensing problem [8]. Given sketch $F\widehat{x}$, where $\widehat{x} \in \mathbb{C}^n$ be a discrete vector, F is the matrix consisting of Fourier basis. The goal is to design measurement matrix S, such that $\widehat{x}$ can be efficiently recovered from observation $\Phi\widehat{x}$ using time and samples as little as possible, where $\Phi = S \cdot F$. In general, such an algorithm does not exist due to the fact that the system is under-determined [9]. However, results in compressed sensing [9] motivate that if we assume $\widehat{x}$ is sparse, such a problem is doable.

*ycao4@wyomingseminary.org. Wyoming Seminary.

†magic.linuxkde@gmail.com. Simons Institute for the Theory of Computing, University of California, Berkeley.

Third Conference on Parsimony and Learning (CPAL 2026).

Following the idea of compressed sensing, people have developed better algorithms than FFT [2, 3, 5]. Suppose the energy of $\widehat{x^*}$ is concentrated in a set S of k coordinates, i.e. $\|\widehat{x^*}\|_0 \leq k$, observation is noisy samples from $x = x^* + g$, where g is the noise. [2] proves that there is an algorithm that recovers $\widehat{x^*}$ using $k \log(n) \log(n/k)$ times and samples. When $k \ll n$, such a sub-linear algorithm is meaningful and allows us to analyze the hidden trait under big data, efficiently compute Fourier transform under specific scenarios, reduce the number of measurements, etc. Existing works in this field always contain two steps [2, 6–8]. In the first step, they do frequency estimation to recover heavy coordinates. In the second step, they do signal estimation, where they recover magnitude and reconstruct the signal based on the coordinates obtained in the former step. To better analyze the sparse Fourier transform problem, a line of works, initialized by [10], only studies the second step, that is how to recover the magnitude with the coordinates known. However, the first step of sparse Fourier transform problem is not well-understood, which motivated us to propose a new problem for a better understanding of sparse Fourier transform.

To establish the frequency estimation step, we propose a novel problem formulated as the existence of a dynamic data structure, which cleverly hides the signal estimation step and helps us focus on frequency estimation. In our new dynamic data structure, we use verification to replace traditional magnitude estimation. By providing coordinates, we are able to determine magnitude exactly. This allows us to separate the signal estimation step. Additionally, we confirm the existence of a dynamic data structure. Our dynamic data structure supports initialize, query, and update procedures. Note that previous works on sparse Fourier transform need to run the full frequency and signal estimation algorithm when each query or update comes. Thus, they do not support dynamic queries and updates.

On the other hand, to evaluate how far we are away from the best algorithm possible, inspiring from the computational complexity, we provide a lower bound for the exact algorithm, which shows that, we are in fact beating the barrier by using randomized approximation algorithm. This completes our contribution from another aspect.

2. Related Work

Compressed sensing Compressed sensing is a powerful mathematical framework with profound applications [11–16]. Compressed sensing aims to recover heavy coordinates of $x \in \mathbb{R}^n$ from measurement $y = \Phi x \in \mathbb{R}^m$. By exploiting the sparsity of x ($\|x\|_0 \leq k$), compressed sensing reduces the number of measurements m and achieve sublinear dependency on n. There are two kinds of guarantee in compressed sensing: for-all and for-each. In for-all guarantee, researchers design a sketch for all vectors. In the for-each guarantee, researchers design a distribution over sketches for a fixed vector. Here, we only consider the for-each model. Initial works [9, 17] shows that, for Gaussian matrix Φ with $O(k \log(n/k))$ measurements, it is possible to get ℓ_2/ℓ_1 guarantee for any vectors x. Later, [18, 19] achieve the ℓ_∞/ℓ_2 guarantee with $O(k \log n)$ measurement, which is information theoretical optimal. Breakthrough work [20] achieves the ℓ_2/ℓ_2 guarantee with $O(k \log(n/k))$ measurement, and $k \log^c n$ decoding time. Sparse Fourier transform is closely related to Compressed sensing. However, in compressed sensing, people are allowed to design sketch matrix Φ arbitrary, in sparse Fourier transform, the sketch matrix Φ is fixed to Fourier transform matrix.

Discrete Fourier transform In general, any algorithm that exactly computes Discrete Fourier transform must take $\Omega(n)$ time. The Fast Fourier Transform algorithm [1] runs in $O(n \log n)$ time and has far-reaching impact on image processing, audio processing, telecommunications, seismology, polynomial multiplication, etc. By exploiting sparsity assumption, there are algorithms that can outperform FFT. There are two lines of works. One line only focuses on optimizing sample complexity based on the renowned Restricted Isometry Property (RIP) [21–26]. While the other line of research achieves both sub-linear time and sample complexity. They only sample a few Fourier coefficients. [2, 3, 5, 27–29] use filter functions that are sparse in the time domain and act as a bandpass filter in the frequency domain. More specifically, they use the HashToBins trick to uniformly scatter heavy coordinates to $O(k)$ bins, filter the frequency domain, use permutation to locate co-

ordinates, and recover magnitude for coordinates. [30] combines the standard FFT algorithm with careful aliasing and avoid the curse of dimensionality, while previous works have exponential dependency on dimensionality due to the construction of the filter. Our work belongs to the second line.

Orthogonal Vector Problem and Complexity The Orthogonal Vector problem (OV) in fine-grained complexity asks whether, given sets $X, Y \subseteq \{0,1\}^d$ of size n, there exist vectors $x \in X$ and $y \in Y$ such that their dot product $\langle x, y \rangle = 0$. The best algorithm for this problem [31, 32] has a time complexity of $n^{2-1/O(\log c)}$ for $d = c \log n$, with $c \geq 1$. As d increases, the complexity approaches n^2. The orthogonal vector conjecture (OVC) proposes a lower bound for OV when $d = \omega(\log n)$, and the Strong Exponential Time Hypothesis (SETH) suggests k-SAT's difficulty supports OVC. This conjecture is crucial for establishing conditional lower bounds for many polynomial-time problems in pattern matching [33–40], graph theory [41–46], computational geometry [47–51], attention computation [52–54]. For more details, see the survey [55].

3. Problem Formulation

3.1. Notations

We use $[n]$ to denote the set $\{1, \ldots, n\}$. We use $\mathbf{i}$ to denote the $\sqrt{-1}$. We define $\omega = e^{-2\pi \mathbf{i}/n}$ to be an nth root of unity. For any complex number a, we use $\phi(a) \in [0, 2\pi]$ to denote the phase of a. For a complex number a and a real positive number b, the expression $a \pm b$ denotes a complex number a' such that $|a - a'| \leq b$. For a vector $x \in \mathbb{C}^n$, its support is denoted by $\mathrm{supp}(x) \subset [n]$. We use $\|x\|_0$ to denote $|\mathrm{supp}(x)|$, the number of non-zero coordinates of x. Its Fourier spectrum is denoted by $\widehat{x}$, with $\widehat{x}_i = \frac{1}{\sqrt{n}} \sum_{j \in [n]} \omega^{ij} x_j$. For a vector of length n, indices should be interpreted modulo n, so $x_{-i} = x_{n-i}$. This allows us to define convolution $(x * y)_i = \sum_{j \in [n]} x_j y_{i-j}$ and the coordinate-wise product $(x \cdot y)_i = x_i y_i$, so $\widehat{x \cdot y} = \widehat{x} * \widehat{y}$. In addition, we define $\pi_{\sigma,b}(i) = \sigma(i - b) \bmod n, h_{\sigma,b}(i) = \mathrm{round}(\pi_{\sigma,b}(i)B/n)$ and $o_{\sigma,b}(i) = \pi_{\sigma,b}(i) - h_{\sigma,b}(i)n/B$. We say $h_{\sigma,b}(i)$ is the "bin" that frequency i is mapped into, and $o_{\sigma,b}(i)$ is the "offset". We define $h^{-1}_{\sigma,b}(j) = \{i \in [n] \mid h_{\sigma,b}(i) = j\}$.

We define the time complexity and exponent of matrix multiplication as follows.

Definition 3.1 (Time Complexity of Matrix Multiplication)**.** *we use $\mathcal{T}_{\mathrm{mat}}(a, b, c)$ to denote the time of multiplying an $a \times b$ matrix with another $b \times c$ matrix.*

Definition 3.2 (Fast matrix multiplication)**.** *We use ω to denote the exponent of matrix multiplication, i.e., $\mathcal{T}_{\mathrm{mat}}(n, n, n) = n^{\omega}$. Currently $\omega \approx 2.371$ [56–60].*

3.2. Structured Fourier Measurement-based Data Structure

Now, we formulate our Structured Fourier Measurement-based Data Structure problem.

Given a matrix $H \in \mathbb{R}^{d \times d}$ and a tensor family that are composed by rank 1 matrix $\{[v_1 v_1^\top, \cdots, v_n v_n^\top]^\top \in \mathbb{R}^{n \times d \times d} \mid v_1, \cdots, v_n \in \mathbb{R}^d\}$ [61–67] [3], we want to recover heavy coordinates (the coordinates with large entries) of $\mathbb{R}^d$ vector in following space:

$$\{[\langle v_1 v_1^\top, H \rangle, \cdots, \langle v_n v_n^\top, H \rangle]^\top \in \mathbb{R}^n | \ v_1, \cdots, v_n \in \mathbb{R}^d\}.$$

To simplify the notation, we squeeze matrix to tensor and tensor to matrix. Then we can restate our objective as follows: Let $U \in \mathbb{R}^{n \times d^2}$ be a linear projection in a specific family such that

$$U \in \{[\mathrm{vec}(v_1 v_1^\top), \cdots, \mathrm{vec}(v_n v_n^\top)]^\top | \ v_1, \cdots, v_n \in \mathbb{R}^d\}.$$

We will focus on studying Sparse Fourier transform on $\{U \cdot h | h \in d^2\}$, a subspace of $\mathbb{R}^n$. Suppose we can observe Fourier measurement $F \cdot (U \cdot h) \in \mathbb{R}^n$ of $U \cdot h$, we only sample a small size $m \ll n$ subset

[3] Except for a number of about matrix sensing. Another angle to motivate this problem is semi-definite programming [68, 69]. We believe defining U as an $n \times d^2$ matrix has connection to the semidefinite programming which with n rank-1 $d \times d$ data matrix [68, 69]. For more detailed discussion, we refer the readers to Section E.

of $[n]$. We call the sub-sampling matrix $S \in \mathbb{R}^{m \times n}$, and we are allowed to design S to optimize costs. Thus, combining Fourier coefficient matrix with sub-sampling matrix, we get that the sketch matrix of $U \cdot h$ is $\Phi = S \cdot F \in \mathbb{R}^{m \times n}$, and the samples are $\Phi \cdot (U \cdot h) = (S \cdot F) \cdot (U \cdot h) \in \mathbb{R}^m$. We want to approximately recover top k heavy coordinates of $U \cdot h$, that is, we aim to output a k-sparse matrix $y \in \mathbb{R}^n$ such that $\|y - Uh\|_2 \leq (1+\epsilon) \min_{k-\mathrm{sparse}\ z} \|z - Uh\|_2$.

However, given fixed U and h, designing S or Φ to optimize time cost is a static problem. To make the problem more difficult, we study a dynamic version of the problem above. We hope to support two dynamic operations Update and Query. Update supports updating $U \in \mathbb{R}^{n \times d^2}$. Query supports to query different $h \in \mathbb{R}^{d^2}$ for current U, and output desired y.

First, we formally define Fourier measurement as follows:

Definition 3.3 (Fourier measurement)**.** *Given a collection of vectors $v_1, \cdots, v_n \in \mathbb{R}^d$, $U \in \mathbb{R}^{n \times d^2}$ denotes the matrix where the i-th row is the vectorization of $v_i v_i^\top \in \mathbb{R}^{d \times d}$. Given vector $h \in \mathbb{R}^{d^2}$.*

Let $F \in \mathbb{C}^{n \times n}$ be a discrete Fourier matrix: $F_{i,j} = e^{-2\pi \mathbf{i} ij/n}$. For each $i \in [n]$, we define Fourier measurement (or Fourier sample) as $(FUh)_i$.

We can see that $\Phi = S \cdot F$ select a few rows from F. Note that for any i, $(FUh)_i$ is Fourier measurement. We are allowed to take any Fourier measurements, e.g. can access any i. But our algorithm can only take a small subset of all the Fourier measurements.

The formal definition of our problem is as follows:

Definition 3.4 (Structured Fourier Measurement (SFM)-based Data Structure)**.** *Given a collection of vectors $v_1, \cdots, v_n \in \mathbb{R}^d$. The goal is to design* **Fourier** *measurements (defined in Definition 3.3) based data structure that supports the following operations:*

- Init$(v_1, \cdots, v_n \in \mathbb{R}^d, n \in \mathbb{N}_+)$*: the data structure finishes initialization.*
- Update$(i \in [n], v_{\mathbf{new}} \in \mathbb{R}^d)$*: Given index i and vector v, the* Update *operation assign v_i with $v_{\mathbf{new}}$.*
- Query$(h \in \mathbb{R}^{d^2})$*: Given vector h, the* Query *operation outputs a k-sparse vector y such that $\|y - Uh\|_2 \leq (1+\epsilon) \min_{k-\mathrm{sparse}\ z} \|z - Uh\|_2$ with high probability, where $U \in \mathbb{R}^{n \times d^2}$ denotes the matrix where the i-th row is the vectorization of $v_i v_i^\top \in \mathbb{R}^{d \times d}$.*

Note that $\Phi \in \mathbb{R}^{m \times n}$. Suppose we can get sub-linear sample complexity, then m is sub-linear. However, because the size of Φ is linear in mn, to recover y in sub-linear time, we can only access a few elements of Φ, which makes our problem highly non-trivial.

4. Our Result - A Fast Approximated Data Structure

In this section, we state the main result. We give an affirmative answer to the problem that we proposed in Definition 3.4. It states that there is a structured Fourier measurement-based data structure that can handle online Update and Query efficiently. For the data structure that we proposed, we also give the value of the three metrics in Definition 4.3, 4.4, 4.5.

4.1. Main Result

Theorem 4.1 (Our result, informal version of Theorem D.1)**.** *Given a collection of vectors $v_1, \cdots, v_n \in \mathbb{R}^d$. Let $U \in \mathbb{R}^{n \times d^2}$ denote the matrix where the i-th row is the vectorization of $v_i v_i^\top \in \mathbb{R}^{d \times d}$. Let $m := k \log n \log(n/k)$. There is a data structure that uses $O(md^2 + nd)$ space that supports the following operations:*

- Init$(v_1, \cdots, v_n \subset \mathbb{R}^d, n \in \mathbb{N}_+, d \in \mathbb{N}_+, k \in \mathbb{N}_+)$*, this operation takes $n(m + d^2)$ time. This operation takes $v_1, \cdots, v_n$ as inputs. It stores $V \in \mathbb{R}^{n \times d}$, and Fourier measurement $\Phi U \in \mathbb{R}^{m \times d^2}$ (where $\Phi = S \cdot F$, $S \in \mathbb{R}^{m \times n}$ is a subsampled matrix where each row only has a single nonzero entries, $F \in \mathbb{C}^{n \times n}$ is a discrete Fourier matrix)*

- *Update*$(i \in [n], v_{\mathbf{new}} \in \mathbb{R}^d)$*, it takes index* i *and vector* v *as input, runs in* $O(md^2)$ *time, and replaces* v_i *with* $v_{\mathbf{new}}$*.*
- *Query*$(h \in \mathbb{R}^{d^2})$*, it takes vector* $h \in \mathbb{R}^{d^2}$ *as input, it outputs* $O(k)$*-sparse* $y \in \mathbb{R}^n$ *such that*

$$\|y - Uh\|_2 \leq (1+\epsilon) \min_{k-\text{sparse } z} \|z - Uh\|_2.$$

  This operation takes time $O(\epsilon^{-1} k \log(n) d^2 + k \log n \cdot \log(n/k))$*.*

Further, let $S = \{i \in [n] \mid |(U \cdot h)_i| > 1\}$*. If* $|S| = k$*, and for any* $i \notin S$*,* $|(U \cdot h)_i| < 1/\operatorname{poly}(n)$*, our Query outputs* k*-sparse* $y \in \mathbb{R}^n$ *such that*

$$\|y - Uh\|_2 \leq \min_{k-\text{sparse } z} \|z - Uh\|_2.$$

In the Query operation, if ϵ goes to 0, and y be exactly k-sparse, the inequality becomes the equal, and y must contain exactly the top-k coordinates in $U \cdot h$. More specifically, the k-sparse z that minimize $\|z - Uh\|_2$ contains the largest k elements in $U \cdot h$ because each term in $U \cdot h$ contributes independently to the ℓ_2 norm and the minimized $\|z - Uh\|_2$ picks out the large terms in Uh greedily. If the ground truth signal $U \cdot h$ has k heavy entries larger than 1, and all the noise entries below $1/\operatorname{poly}(n)$, our Query can find the those k heavy entries exactly. In general, our output y is $O(k)$-sparse, our approximation ratio is also $1+\epsilon$, thus our guarantee indicates that we can find the top-k coordinates approximately. Or equivalently, we can find the "heavy" coordinates in $U \cdot h$. Because m depends linearly on k and logarithmically on n, thus m is sub-linear. When we choose d also to be sub-linear, both the Query and Update time is sub-linear, and only the Init time is linear in n, which shows that our algorithm is very efficient.

4.2. Optimization Objectives

In this section, we introduce objectives in solving our problem in Definition 3.4. First, we define Measurements or Samples as follows:

Definition 4.2 (Measurements/Samples)**.** *Let* $S \in \mathbb{R}^{m \times n}$ *be the subsampling matrix where each row contains only a single nonzero entry and* m *is the number of measurements. Then, we define the sketch matrix* $\Phi = S \cdot F$*.*

Although we can sample any Fourier measurement $(FUh)_i$ as long as $i \in [n]$, we hope to sample as few as possible while keeping the time complexity small. The subsampling matrix S represents the indices of the samples that our algorithm takes.

Definition 4.3 (Computing Measurements/Samples time)**.** *For any query vector* $h \in \mathbb{R}^{d^2}$*, we say "the time of computing measurements/samples" is the time of computing* $\Phi U \cdot h$*.*

This definition corresponds to the encoding time of compressed sensing.

Then, we define the the time of recovering:

Definition 4.4 (Recover time)**.** *Let* U, h *be defined as in Definition 3.4,* Φ *be defined as in Definition 4.2. Given access to* $\Phi Uh \in \mathbb{R}^m$*, we say "the time of recovering" is the time of finding an index set* $L \subset [n]$*, such that there exists a vector* $y \in \mathbb{R}^n$ *such that*

$$\|y_L - Uh\|_2 \leq (1+\epsilon) \min_{k-\text{sparse } z} \|z - Uh\|_2,$$

Next, we define the the time of verification:

Definition 4.5 (Verification time)**.** *Given* $U \in \mathbb{R}^{n \times d^2}$*, for any query vector* $h \in \mathbb{R}^{d^2}$ *and any set* $L \subset [n]$*, we say verification time for set* L *is the time of computing* $(Uh)_L$*.*

Those three metrics can be viewed as a detailed discussion on Query operation. Combine the three time complexity together, we can get the time complexity of the Query operation. We state our results as follows:

Theorem 4.6 (Metrics)**.** *For the data structure in Theorem 4.1, we have that The time of computing measurements/samples in Definition 4.3 is $\mathcal{T}_{\mathrm{mat}}(|L|, d^2, 1)$. The time of recovering in Definition 4.4 is $O((k \log n + \mathcal{T}_{\mathrm{sam}}) \cdot \log(n/k))$. The time of verification in Definition 4.5 is $(|L|/d) \cdot \mathcal{T}_{\mathrm{mat}}(d, d, d)$. $\mathcal{T}_{\mathrm{mat}}(a, b, c)$ is the time of multiplying an $a \times b$ matrix with $b \times c$ matrix, $\mathcal{T}_{\mathrm{sam}}$ is the running time of sampling an element from a set uniformly.*

Note that all of the three time is sub-linear in n. Moreover, since we need to output at least k sparsity, the time of recovering is theoretically nearly optimal. We provide lower bound analysis for exact running time in the next section.

5. Our Result - A Lower Bound For Exact Data Structure

In this section, we provide a lower bound discussion here.

5.1. Previous results on Maximum Inner Product

We state a result considering the maximum bichromatic inner product lower bound here [50]. For more background about complexity involving SETH and OVC, we refer reader to Appendix A.7.

Definition 5.1 (Bichromatic Maximum Inner Product ($\mathsf{Max-IP}_{n,d}$))**.** *For $n, d \in \mathcal{N}$, the $\mathsf{Max-IP}_{\mathsf{n,d}}$ probelm is defined as: given two sets A, B each consisting of n vectors from $\{0,1\}^d$ compute*

$$\mathsf{OPT}(A, B) := \max_{a \in A, b \in B} a \cdot b.$$

We use $\mathbb{Z} - \mathsf{Max-IP}_{n,d}$ to denote the same problem, with A, B being sets of vectors from $\mathbb{Z}^d$.

Then we introduce the theorem in previous work about lower bound for Maximum bichromatic inner product.

Theorem 5.2 (Maximum bichromatic inner product lower bound [50])**.** *Under assumption of* SETH (*Definition A.7*) *or* OVC (*Definition A.9*)*, there is a constant c such that any exact algorithm for $\mathbb{Z} - \mathsf{Max-IP}_{\mathsf{n,d}}$ in dimension $d = c^{\log^* n}$ requires $n^{2-o(1)}$ time, with vectors of $O(\log n)$-bit entries.*

5.2. Lower Bound by Reduction from Maximum Inner Product Search

Definition 5.3 (Exact Structured Fourier Measurement (SFM)-based Data Structure, exact version of Definition 3.4)**.** *Following Definition 3.4, our goal is given vector h, $\textsc{Query}(h \in \mathbb{R}^{d^2}, k \in [n])$ can output the top-k entries of vector Uh, where $U \in \mathbb{R}^{n \times d^2}$ denotes the matrix where the i-th row is the vectorization of $v_i v_i^\top \in \mathbb{R}^{d \times d}$.*

Theorem 5.4 (Lower Bound for exact SFM)**.** *Let $d = 2^{O(\log^* n)}$. Given* SETH *and* OVC*, the exact* SFM (*Definition 3.4*) *cannot support initialization time of $O(n)$ and query time of $O(n^{1-\epsilon})$ for any $\epsilon \in (0, 1)$.*

Proof. Let $\epsilon > 0$ be a parameter. We consider $m \geq n$ and $d = c^{\log^*(n)}$, where c is defined as in Theorem 5.2. Assume there is a data structure on a instance on a pair of size n sets of points in d dimensional space, with an update time of $O(n^{1-\epsilon})$ and query time of $O(n^{1-\epsilon})$. Following Theorem 5.2, we construct a hard instance of $\mathbb{Z} - \mathsf{Max-IP}_{m,d}$ problem with $H = \{h_1, \ldots, h_n\} \subseteq \mathbb{Z}^{d^2}, V = \{v_1, \ldots, v_n\} \subseteq \mathbb{Z}^{d^2}$.

We first call $\textsc{Init}(V)$ to initialize the data structure, which takes time $O(n)$.

Now for each $i \in [n]$, call $\textsc{Query}(h_i, 1)$ to get the largest inner product between h_i and for every v_j for $j \in [n]$. This takes time $O(n \cdot n^{1-\epsilon}) = O(n^{2-\epsilon})$ for all $i \in [n]$. Now by comparing all the largest inner product from the previous step, we get $\mathsf{OPT}(H, V)$. This step takes time $O(n)$.

Above algorithm implies that $\mathbb{Z} - \mathsf{Max-IP}_{n,d}$ problem can be solved in time $O(n^{2-\epsilon} + n) = O(n^{2-\epsilon})$. This contradicts Theorem 5.2. Hence, this data structure cannot exist. □

Our data structure (Theorem 4.1) in fact runs faster than this lower bound, which shows that we beats the computational barrier by randomization techniques, with a little sacrifice of accuracy.

6. Technical Overview

The following sections contain a streamlined technical overview of the main ideas and techniques required to prove our results in Theorem 4.1. Our goal is to support the UPDATE operation and QUERY operation in Definition 3.4, while optimizing the time of computing measurement in Definition 4.3, the time of recovering in Definition 4.4, and the time of verification in Definition 4.5.

Our three steps' running times are not independent. Their costs depend on the previous step. Note that verification has a strong connection with recovering heavy coordinates. Verification correctly gives all the information of the set of coordinates. Although verification can not tell the relative magnitude among all the n coordinates, verification answers the magnitude of a set of coordinates exactly. Suppose we have a black box to recover a constant fraction of heavy coordinates in frequency domain, we call the black box or oracle, and successively verify the magnitude of the set of coordinates outputted by the black box. The set of coordinates and its corresponding magnitude form a recovered signal. This recovered signal may not be good, but we can repeatedly call the black box again by subtracting this signal from the original signal. Doing this iteratively can recover the signal with high accuracy. Thus, with verification as oracle, we may get a better recovery algorithm with better running time.

The key is to utilize the dynamic inherent property of our problem and the properties of matrix multiplication. By moving redundant computation to the initialization step, we save $\mathcal{T}_{\mathrm{mat}}(m, n, d^2)$ time each time we encounter a QUERY operation. In addition to the above optimization, we leverage the properties of sparse matrix, rank 1 matrix, and fast matrix multiplication, which leads to a better result on time complexity.

6.1. Update

To support dynamic operations, computing measurement, recovering, and verification are called when each online operations come. By leveraging the similarity between each call of those operations, we may be able to save the cost of time by pre-processing. In measurement computing, we need to compute $S \cdot F \cdot U \cdot h$. When a different query h comes, computing $S \cdot F \cdot U$ is redundant. Thus, if there is no UPDATE operation, we can simply pre-compute $M = S \cdot F \cdot U$ in INIT procedure, and compute $M \cdot h$ each time. When an update of v_i comes, $U = [\mathrm{vec}(v_1 v_1^\top), \mathrm{vec}(v_2 v_2^\top), \cdots, \mathrm{vec}(v_n v_n^\top)]^\top$ changed correspondingly. Thus, we need to compute a new M_{new} based on the new U_{new}. Naively, M_{new} is computed by multiplying, S, F, U together again. However, by carefully analyzing the difference brought by the update, we can do better. As shown in Algorithm 1 Procedure UPDATE, we represent the difference of U by $C = U_{\mathrm{new}} - U = \begin{bmatrix} \mathbf{0} & \cdots & \mathrm{vec}(v_{\mathrm{new}} v_{\mathrm{new}}^\top) - \mathrm{vec}(v_i v_i^\top) & \cdots & \mathbf{0} \end{bmatrix}^\top$, where $\mathbf{0}$ is an all zero vectors that has length d^2. Then, we have $M_{\mathrm{new}} = SFU_{\mathrm{new}} = \Phi C + M$. Note that C is a sparse matrix, computing $S \cdot F \cdot C$ is much faster than $S \cdot F \cdot U_{\mathrm{new}}$.

Lemma 6.1 (Running Time of INIT, Algorithm 1, Algorithm 2, see Lemma B.1 for formal version with proof)**.** *Given $\{v_1, \cdots, v_n\}, n \in \mathbb{N}_+, d \in \mathbb{N}_+, k \in \mathbb{N}_+$, procedure INIT stores vectors $\{v_1, \cdots, v_n\}$ into V, creates $\Phi \in \mathbb{R}^{m \times n}$ and stores ΦU into M, where U is in $\mathbb{R}^{n \times d^2}$ and i-th row of U is vectorization $v_i v_i^\top$. This procedure takes $O(n(m + d^2))$.*

Lemma 6.2 (Running Time of UPDATE, see Lemma B.2 for formal version with proof)**.** *Given $i \in [n]$ and $z \in \mathbb{R}^d$, procedure UPDATE takes i and v as inputs and runs in $O(md^2)$ time.*

6.2. Verification

As shown in Algorithm 1, Procedure VERIFIED verifies the magnitudes for a set of coordinates L of $U \cdot h$. When large coordinates are located, previous works need to estimate these magnitudes by using HASHTOBINS. They use HASHTOBINS because when the frequency is isolated, the total energy in the hashed bin can be used to approximate the magnitude. However, using HASHTOBINS to estimate magnitude is time-consuming and has a low accuracy. In our setting, the structured signal makes it easy to verify the large coefficients. We have direct access to them by calculating entries of $U \cdot h$. Given an index set $L \subset [n]$ returned by Procedure FINDLARGEINDEX in Algorithm 1, we can verify

the coefficients by calculating $(U \cdot h)_L$ in $O(|L|/d \cdot \mathcal{T}_{\mathrm{mat}}(d,d,d))$ time. The small running time is due to a critical feature of our data structure: Our input signal are well-structured, and it supports dynamic operations. Let V in $\mathbb{R}^{d \times d}$ denotes the matrix whose i-th row is v_i. Let matrix H in $\mathbb{R}^{d \times d}$ be the matrix version of $h \in \mathbb{R}^{d^2}$. Then, we are able to accelerate computation of $(Uh)_L$ by computing matrix $VHV^\top$ and then taking the diagonal entries: $(VHV^\top)_{i,i} = (Uh)_i$.

6.3. Structured HashToBins

StructuredHashToBins randomly shuffles all the n frequency coordinates, divides the frequency domain into $B = O(k)$ equally-spaced bins, and collects rotated magnitudes. With a high probability, each bin only contains a single approximately top-k frequency, which allows us to recover its magnitude. We call a single approximately top-k coordinate isolated when it is hashed to a bin that only contains itself. Then, StructuredHashToBins transforms the problem of recovering k heavy coordinates to recovering each isolated coordinate separately. This transformation can be implemented by multiplying the signal in the frequency domain with a filter function. However, each approximately top-k coordinate becomes isolated only with constant probability, which means we can only recover a constant fraction of coordinates at one time. Note that we do not use the full sketch Φ, we only use a part of it. In each call to StructuredHashToBins, we do Fourier transform on $O(k)$ measurements, which means that we only use $O(k)$ rows out of the n rows of Φ.

Our StructuredHashToBins supports multiple Query and Update operations on the fly. The key idea is that we fix the randomness when we initialize our data structure StructuredFourier. We create M each time online operations come. By fixing $\Phi, \mathcal{L}, \mathcal{M}$ and pre-processing them during initialization, we only observe part of M when StructuredHashToBins is invoked, thus reduce time complexity. More specifically, we pre-compute the coordinates of measurements in time domain by the parameters of our filter function and the parameters of our random shuffling. When each Query comes, we call ComputeMeasurements to calculate measurements online, then we transform this information to the frequency domain.

Lemma 6.3 (Running Time of ComputingMeasurements, see Lemma B.4 for formal version with proof)**.** *Given an index set $L \subset [n]$, the running time of ComputingMeasurements is $\mathcal{T}_{\mathrm{mat}}(|L|, d^2, 1)$.*

Lemma 6.4 (Running Time of StructuredHashToBins, Algorithm 4, see Lemma B.5 for formal version with proof)**.** *Let $\eta = |\{i \in \mathrm{supp}(\widehat{z}) \mid E_{\mathrm{off}}(i)\}|$. The running time of StructuredHashToBins is $O(\mathcal{T}_{\mathrm{mat}}(\frac{B}{\alpha}\log(n/\delta), d^2, 1) + \|\widehat{z}\|_0 + \eta \log(n/\delta))$.*

6.4. Find large index

In this step, we utilize the StructuredHashToBins trick. The transformed Fourier measurements of StructuredHashToBins contains information about large coefficients. Then, we extract the phase and frequency from the Fourier measurements, and determine the location (index) of the large coordinates by a voting procedure. The running time of FindLargeIndex is dominated by the call of StructuredHashToBins. The running time of StructuredHashToBins is dominated by the number of bins, which is linear in k.

In the FindLargeIndex procedure in Algorithm 1, R is the number of iterations. $\widehat{z}^{(j)}$ is the residual signal, where j is number of iterations. Naively, FindLargeIndex should take $x - z^{(j)}$ as parameter. However, computing $x - z^{(j)}$ efficiently in sub-linear is intractable. Thus, we first hash x into the bins. We defer the computation of residual signal after we get the energy in each bins. We shuffle $\widehat{z}^{(j)}$ according to the same strategy that shuffle $\widehat{x}$, and subtract the energy in each bins by the corresponding energy of $\widehat{z}^{(j)}$. The parameter B_r is the number of bins in this round. The parameter α_r controls the fraction of heavy coordinates that can be recovered in this iteration.

To prove the correctness of the FindLargeIndex procedure in Algorithm 1, we first notice that there are three kinds of bad event. In the first event, two heavy coordinates are hashed into the same bin. This is unlikely to happen because we can choose B larger than any constant times of k. Moreover, each coordinate can be shuffled into any new coordinates. It is hard to collide. In the second event, the heavy coordinate are hashed into the edge of a bin. This happens because we are not using ideal filter. To get sub-linear time, we hope our filter to be sparse in time domain, while look like a ideal

filter as much as possible. Because the filter should be sparse, thus band-limit in time domain, the filter drops high frequency component and becomes smooth on the edge of what supposed to be sharp in a box function. However, we can control the length of the smooth area to be small. Thus this kind of event is also not likely to happen. In the third event, the non-heavy coordinates is not shuffled evenly, which means that the heavy coordinate can not dominate the bin because the noise is too strong. Since our shuffle strategy is random enough, by Markov inequality, this is also unlikely to happen.

Conditioned on only a good event happening, there is exactly one heavy coordinate in a bin, and all the other bins are filtered out by the approximate ideal filter. This process narrows down our focus, allowing us to efficiently locate the index of the heavy coordinates in the bin. It's important to recognize that any shift in the time domain results in a corresponding rotation in the frequency domain. This rotation is not solely related to time shifts but is also influenced by the position of the coordinates in the frequency domain. By randomly shifting x, we reduce the impact of the length of the shift in the frequency domain, providing us with helpful clues to identify the position of the heavy coordinate. These random shifts and the filtering together help refine the accuracy of the identification process, ensuring we can locate the heavy coordinate effectively despite any potential noise or complexities in the system.

However, rotation in frequency domain always modular 2π. We still need to decode the real position from the phase of a complex number. In general, it is impossible to recover an exact coordinate by a single phase. However, by repeating with different length of shifting in time domain, we can get different phases. We can prove that, with a low probability, two coordinates can perform similarly in all those shifting. Thus, we use $\log(n)$-ary search. We go through $\log(n)$ length arithmetic series and test whether the current coordinate have the desired phases. By repeating the search for logarithmic times, we can locate heavy coordinate exactly.

6.5. Query

Given a query $h \in \mathbb{R}^{d^2}$, we query the approximately largest k coordinates of structured signal $Uh \in \mathbb{R}^d$ by using Fourier measurements. As shown in Algorithm 1, Procedure Query calls Procedure FindLargeIndex for R times. Let the call to one FindLargeIndex be an iteration. After each iteration, we verify the magnitude of those coordinates, and reconstruct an signal that can approximate the Fourier measurement of the ground truth. We subtract our reconstructed signal out of the ground truth Fourier measurement and obtain a new signal. The new signal only contains a small fraction of energy of the signal that before subtraction. By iterating for logarithmic time, we can cover each approximately top-k coordinate with high probability. Thus, we successfully locate all the top-k heavy coordinates. Note that because we can recover a constant fraction of coordinates in each iteration, the number of recovered coordinates and the running time of each iteration forms a geometric series. Therefore, the computational cost of the first time iteration determines the total time complexity.

7. Conclusion

The study of Sparse Fourier Transform (SFT) is crucial in signal processing, especially for frequency and signal estimation, as shown in prior research [2, 6, 7]. In this work, we focus on signal estimation and introduce a new problem of frequency estimation under structured sparse recovery with Fourier Measurements. This dynamic sub-problem of the classic sparse Fourier transform allows for a detailed exploration of frequency estimation challenges. We propose an efficient algorithm to solve this problem, providing a foundation for future research and applications. Moreover, our data structure supports online updates and queries in sub-linear time, enhancing the practicality in real-world scenarios.

References

[1] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[2] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse fourier transform. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 563–578, 2012.

[3] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Simple and practical algorithm for sparse fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1183–1194. SIAM, 2012.

[4] Anna C. Gilbert, Piotr Indyk, Mark A. Iwen, and Ludwig Schmidt. Recent developments in the sparse fourier transform: A compressed fourier transform for big data. *IEEE Signal Process. Mag.*, 31(5):91–100, 2014. doi: 10.1109/MSP.2014.2329131. URL `https://doi.org/10.1109/MSP.2014.2329131`.

[5] Piotr Indyk and Michael Kapralov. Sample-optimal fourier sampling in any constant dimension. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 514–523. IEEE, 2014.

[6] Eric Price and Zhao Song. A robust sparse Fourier transform in the continuous setting. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 583–600. IEEE, 2015.

[7] Xue Chen, Daniel M Kane, Eric Price, and Zhao Song. Fourier-sparse interpolation without a frequency gap. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 741–750. IEEE, 2016.

[8] Vasileios Nakos, Zhao Song, and Zhengyu Wang. (nearly) sample-optimal sparse fourier transform in any dimension; ripless and filterless. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1568–1577. IEEE, 2019.

[9] Emmanuel J Candes, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(8):1207–1223, 2006.

[10] Eric Price. Efficient sketches for the set query problem. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 41–56, 2011.

[11] Yunhui Zheng, Nikos P Pitsianis, and David J Brady. Nonadaptive group testing based fiber sensor deployment for multiperson tracking. *IEEE Sensors Journal*, 6(2):490–494, 2006.

[12] Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. k-mismatch with don't cares. In *European Symposium on Algorithms*, pages 151–162. Springer, 2007.

[13] Marco F Duarte, Mark A Davenport, Dharmpal Takhar, Jason N Laska, Ting Sun, Kevin F Kelly, and Richard G Baraniuk. Single-pixel imaging via compressive sampling. *IEEE signal processing magazine*, 25(2):83–91, 2008.

[14] Raghunandan M Kainkaryam, Angela Bruex, Anna C Gilbert, John Schiefelbein, and Peter J Woolf. poolmc: Smart pooling of mrna samples in microarray experiments. *BMC bioinformatics*, 11(1):1–9, 2010.

[15] Surya Ganguli and Haim Sompolinsky. Compressed sensing, sparsity, and dimensionality in neuronal information processing and data analysis. *Annual review of neuroscience*, 35:485–508, 2012.

[16] Denisa Duma, Mary Wootters, Anna C Gilbert, Hung Q Ngo, Atri Rudra, Matthew Alpert, Timothy J Close, Gianfranco Ciardo, and Stefano Lonardi. Accurate decoding of pooled sequenced data using compressed sensing. In *International Workshop on Algorithms in Bioinformatics*, pages 70–84. Springer, 2013.

[17] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[18] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[19] Graham Cormode and S Muthukrishnan. Combinatorial algorithms for compressed sensing. In *International colloquium on structural information and communication complexity*, pages 280–294. Springer, 2006.

[20] Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing*, 41(2):436–453, 2012.

[21] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

[22] Mark Rudelson and Roman Vershynin. On sparse reconstruction from fourier and gaussian measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 61(8):1025–1045, 2008.

[23] Thomas Blumensath and Mike E Davies. Normalized iterative hard thresholding: Guaranteed stability and performance. *IEEE Journal of selected topics in signal processing*, 4(2):298–309, 2010.

[24] Deanna Needell and Roman Vershynin. Signal recovery from incomplete and inaccurate measurements via regularized orthogonal matching pursuit. *IEEE Journal of selected topics in signal processing*, 4(2):310–316, 2010.

[25] Jean Bourgain. An improved estimate in the restricted isometry problem. In *Geometric aspects of functional analysis*, pages 65–70. Springer, 2014.

[26] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled fourier matrices. In *Geometric aspects of functional analysis*, pages 163–179. Springer, 2017.

[27] Piotr Indyk, Michael Kapralov, and Eric Price. (nearly) sample-optimal sparse fourier transform. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 480–499. SIAM, 2014.

[28] Michael Kapralov. Sparse fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 264–277, 2016.

[29] Michael Kapralov. Sample efficient estimation and recovery in sparse FFT via isolation on average. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 651–662. IEEE Computer Society, 2017. doi: 10.1109/FOCS.2017.66. URL `https://doi.org/10.1109/FOCS.2017.66`.

[30] Michael Kapralov, Ameya Velingker, and Amir Zandieh. Dimension-independent sparse fourier transform. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2709–2728. SIAM, 2019.

[31] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 218–230. SIAM, 2014.

[32] Timothy M Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1246–1255. SIAM, 2016.

[33] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I 41*, pages 39–51. Springer, 2014.

[34] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014.

[35] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014.

[36] Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–466. IEEE, 2016.

[37] Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016.

[38] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE, 2017.

[39] Karl Bringman and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1216–1235. SIAM, 2018.

[40] 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.

[41] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524, 2013.

[42] Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. *ACM Transactions on Algorithms (TALG)*, 14(3):1–23, 2018.

[43] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Transactions on Algorithms (TALG)*, 15(2):1–35, 2018.

[44] Robert Krauthgamer and Ohad Trabelsi. Conditional lower bounds for all-pairs max-flow. *ACM Transactions on Algorithms (TALG)*, 14(4):1–15, 2018.

[45] Mina Dalirrooyfard, Ray Li, and Virginia Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1021–1032. IEEE, 2022.

[46] Timothy M Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Hardness for triangle problems under even more believable hypotheses: reductions from real apsp, real 3sum, and ov. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1501–1514, 2022.

[47] Kevin Buchin, Maike Buchin, Maximilian Konzack, Wolfgang Mulzer, and André Schulz. Fine-grained analysis of problems on curves. *EuroCG, Lugano, Switzerland*, 3, 2016.

[48] Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, pages 1260–1268, 2018.

[49] Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1207–1215. SIAM, 2018.

[50] Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. In *Proceedings of the 33rd Computational Complexity Conference*, pages 1–45, 2018.

[51] CS Karthik and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. *Combinatorica*, 40(4):539–573, 2020.

[52] Josh Alman and Zhao Song. Fast attention requires bounded entries. *Advances in Neural Information Processing Systems*, 36, 2023.

[53] Josh Alman and Zhao Song. How to capture higher-order correlations? generalizing matrix softmax attention to kronecker computation. In *The Twelfth International Conference on Learning Representations*, 2024.

[54] Josh Alman and Zhao Song. The fine-grained complexity of gradient computation for training large language models. *arXiv preprint arXiv:2402.04497*, 2024.

[55] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

[56] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi: 10.1145/2213977.2214056. URL `https://doi.org/10.1145/2213977.2214056`.

[57] 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.

[58] 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.

[59] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM, 2024.

[60] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2025.

[61] Kiryung Lee and Yoram Bresler. Guaranteed minimum rank approximation from linear observations by nuclear norm minimization with an ellipsoidal constraint. *arXiv preprint arXiv:0903.4742*, 2009.

[62] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[63] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. *IEEE transactions on information theory*, 56(6):2980–2998, 2010.

[64] Prateek Jain, Raghu Meka, and Inderjit Dhillon. Guaranteed rank minimization via singular value projection. *Advances in Neural Information Processing Systems*, 23, 2010.

[65] Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM review*, 52(3):471–501, 2010.

[66] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674, 2013.

[67] Kai Zhong, Prateek Jain, and Inderjit S Dhillon. Efficient matrix sensing using rank-1 gaussian measurements. In *International conference on algorithmic learning theory*, pages 3–18. Springer, 2015.

[68] Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In *FOCS*, 2020.

[69] Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving sdp faster: A robust ipm framework and efficient implementation. In *FOCS*, 2022.

[70] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[71] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

[72] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

[73] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78. IEEE, 2015.

Appendix

Roadmap. In Section A, we introduce some basic notations, definitions and properties that are useful to prove the our results. In Section B, we begin to analyse the running time of our algorithm. In Section C, we move to the correctness of the algorithm. In Section D, we put the results above together to prove our main Theorem 4.1. In Section E, we talk about the motivation of our setting from Semidefinite Programming. In Section F, we show a case when we can recover top-k coordinates exactly. Section G discusses the limitation of this work. Finally, Section H discusses potential societal impact of this work.

A. Preliminaries

In this section, we introduce the notation, definitions and properties used in the rest of this paper. In Section A.1, we state the notations we used in the proofs. In Section A.2 we introduce the modular property. In Section A.3 we introduce the permutation function, In Section A.4 we introduce the filter function. In Section A.5 we state the property of StructuredHashToBin. We introduce the Discrete Fourier Transform in Section A.6. We introduce objectives in solving our problem in Definition 3.4 in Section 4.2. We introduce SETH and OVC from computational complexity in Section A.7.

A.1. More Notation

We list the following notations that will be used in our proofs. Define $\pi_{\sigma,b}(i) = \sigma(i-b) \bmod n$, $h_{\sigma,b}(i) = \mathrm{round}(\pi_{\sigma,b}(i)B/n)$ and $o_{\sigma,b}(i) = \pi_{\sigma,b}(i) - h_{\sigma,b}(i)n/B$. We say $h_{\sigma,b}(i)$ is the "bin" that frequency i is mapped into, and $o_{\sigma,b}(i)$ is the "offset". We define $h^{-1}_{\sigma,b}(j) = \{i \in [n] \mid h_{\sigma,b}(i) = j\}$ and $\mathrm{error}(Uh, k) = \min_{k-\mathrm{sparse}y} \|Uh - y\|_2$.

A.2. Modular Property

The goal of this section is to introduce modular property.

Lemma A.1 (Lemma 4.3 in [2]). *Let $T \subset [m]$ consist of t consecutive integers, and suppose $\beta \in T$ uniformly at random. Then for any $i \in [n]$ and set $S \subset [n]$ of l consecutive integers.*

$$\Pr[\beta i \bmod n \in S] \leq \lceil im/n \rceil (1 + \lfloor l/i \rfloor)/t \leq \frac{1}{t} + \frac{im}{nt} + \frac{lm}{bt} + \frac{l}{it}$$

We use the following lemma from [3]:

Lemma A.2 (Lemma 3.6 of [3]). *If $j \neq 0, n$ is a power of two, and σ is a uniformly random odd number in $[n]$, then $\Pr[\sigma j \in [-C, C] \pmod n] \leq 4C/n$.*

A.3. Permutation Function

When $i \in \mathbb{Z}$ is an index into an n-dimensional vector, sometimes we use $|i|$ to denote $\min_{j \equiv i(\pmod n)} |j|$.

We use the standard permutation from literature [3].

Definition A.3. *Suppose σ^{-1} exists $\pmod n$. We define the permutation $P_{\sigma,a,b}$ by*

$$(P_{\sigma,a,b}x)_i = x_{\sigma(i-a)}\omega^{\sigma bi}.$$

We also define $\pi_{\sigma,b}(i) = \sigma(i-b) \pmod n$.

Next lemma shows how P permutate signal x in the frequency domain.

Lemma A.4 (Claim 2.2 in [2]). $\widehat{P}_{\sigma,a,b}x_{\pi_{\sigma,b}(i)} = \widehat{x}_i\omega^{a\sigma i}$.

A.4. Filter Function

In this section, we define the filter function.

Definition A.5 (Definition 2.3 of [2])**.** *We say that* $(G, \widehat{G}') = (G_{B,\delta,\alpha}, \widehat{G'}_{B,\delta,\alpha}) \in \mathbb{R}^n \times \mathbb{R}^n$ *is a flat window function with parameters* $B \geq 1, \delta > 0$*, and* $\alpha > 0$ *if* $|\operatorname{supp}(G)| = O(\frac{B}{\alpha}\log(n/\delta))$ *and* $\widehat{G}'$ *satisfies*

- $\widehat{G}'_i = 1$ *for* $|i| \leq (1-\alpha)n/(2B)$
- $\widehat{G}'_i = 0$ *for* $|i| \geq n/(2B)$
- $\widehat{G}'_i \in [0,1]$ *for all* i
- $\|\widehat{G}' - \widehat{G}\|_\infty < \delta$.

The above notion corresponds to the $(1/(2B), (1-\alpha)/(2B), \delta, O(B/\alpha \log(n/\delta))$-flat window function in [3].

The fact that $\widehat{G'}_i$ takes $\omega(1)$ time to compute for $i \in [(1-\alpha)n/(2B), n/(2B)]$ will add some complexity to our algorithm and analysis. We will need to ensure that we rarely need to compute such values. A practical implementation might find it more convenient to precompute the window functions in a preprocessing stage, rather than computing them on the fly.

A.5. Property of StructuredHashToBins

The goal of this section is to present the output of StructuredHashToBin

Lemma A.6. *Suppose* B *divides* n*. The output* $\widehat{\mathsf{u}}$ *of StructuredHashToBins satisfies*

$$\widehat{\mathsf{u}}_j = \sum_{h_{\sigma,b}(i)=j} (\widehat{Uh_L - z})_i (\widehat{G'_{B,\delta,\alpha}})_{-o_{\sigma,b}(i)} \omega^{a\sigma i} \pm \delta \|\widehat{Uh}\|_1$$

Proof. Define the flat window functions $G = G_{B,\delta,\alpha}$ and $\widehat{G'} = \widehat{G'B,\delta,\alpha}$. We have

$$\widehat{y} = G \cdot \widehat{P_{\sigma,a,b} U h} = \widehat{G} * \widehat{P_{\sigma,a,b} U h}$$
$$\widehat{y'} = \widehat{G'} * \widehat{P_{\sigma,a,b}(Uh - z)} + (\widehat{G} - \widehat{G'}) * \widehat{P_{\sigma,a,b} U h}$$

By lemma A.4, the coordinates of $\widehat{P_{\sigma,a,b} U h}$ and $\widehat{Uh}$ have the same magnitudes with different ordering and phase. Therefore

$$\|(\widehat{G} - \widehat{G'}) * \widehat{P_{\sigma,a,b} U h}\|_\infty \leq \|\widehat{G} - \widehat{G'}\|_\infty \|\widehat{P_{\sigma,a,b} U h}\|_1 \leq \delta \|\widehat{Uh}\|_1$$

and hence

$$\begin{aligned}
\widehat{\mathsf{u}}_j = \widehat{y}'_{jn/B} &= \sum_{|l| < n/(2B)} \widehat{G}'_{-l} (\widehat{P_{\sigma,a,b}(Uh - z)})_{jn/B+l} \pm \delta \|\widehat{Uh}\|_1 \\
&= \sum_{|\pi_{\sigma,b}(i) - jn/B| < n/(2B)} \widehat{G}'_{jn/B - \pi_{\sigma,b}(i)} (\widehat{P_{\sigma,a,b}(Uh - z)})_{\pi_{\sigma,b}(i)} \pm \delta \|\widehat{Uh}\|_1 \\
&= \sum_{h_{\sigma,b}(i)=j} \widehat{G}'_{-o_{\sigma,b}(i)} \widehat{Uh - z}_i w^{a\sigma i} \pm \delta \|\widehat{Uh}\|_1
\end{aligned}$$

Thus, we complete the proof.

□

A.6. Discrete Fourier transform

In this section, we define Fourier transform related notations. Discrete Fourier Transform is already defined as in Eq. (1). Next, we define inverse Fourier transform of $x \in \mathbb{C}^n$ as follows:

$$x_t = \frac{1}{\sqrt{n}} \sum_{f=1}^{n} \widehat{x}_f \exp(\frac{2\pi \mathbf{i} f t}{n}), \text{ where } t \in [n],$$

A.7. SETH and OVC

Here we introduce some notions from computational complexity, for our analysis of lower bound.

Definition A.7 (Strong exponential time hypothesis (SETH) [70, 71]). *For any $\varepsilon > 0$, there exists a $k = k(\varepsilon)$ such that the k-SAT problem with n variables cannot be solved in time $O(2^{1-\epsilon}n)$.*

In order to introduce OVC, we need to define Orthogonal Vector problem first.

Definition A.8 (Orthogonal Vector problem). *Given a set of n vectors $\{v_1, \ldots, v_n\} \subseteq \{0,1\}^d$ in d-dimensional space. We ask if there exists $(i,j) \in [n] \times [n]$ such that $\langle v_i, v_j \rangle = 0$.*

Definition A.9 (Orthogonal vector conjecture (OVC) [33, 36, 72, 73]). *For every $\varepsilon > 0$, there exists a $c = c(\varepsilon) > 1$ such that OV cannot be solved in $n^{2-\varepsilon}$ time when $d = c \log n$.*

B. Running Time Analysis

This section is organized as follows: In Section B.1, we present the running time of Init. We will show how to construct Fourier measurement Φ and calculate the structure matrix U, which involves fast matrix computation. Section B.2 presents the running time of Update. We show that to update the i-th ector v_i with a new vector z, it suffices to take $O(md^2)$ matrix computation time. We analyze the running time of Query in Section B.3. We unify the results in Section B.4, Section B.5 and Section B.6 to compute the running time of Query, which is based on the formula Query time = Computing sample time + Locate Signal time + Verification time. Next three sections describe the running time of three different procedures in the same iteration in Query. We will show the running time per iteration and for the whole iterative process. In section B.4, we present the running time of StructuredHashToBins. The computation cost is divided into four parts which correspond to the four steps of calculating StructuredHashToBins. It is dominated by the support of signals and number of bins. Section B.5 shows the running time of Verification. Due to the matrix structure, we can use fast matrix computation to accelerate the computation in Verification. Finally, we demonstrate the running time of LocatSignal in section B.6. Analysis in this part starts from FindLargeIndexInner, which is in a forloop of FindLargeIndex. Using the running time of the inner loop, we obtain the total running time by determining the number of loops

B.1. Running Time of Init

The goal of this section is to prove the running time of Init.

Lemma B.1 (Running Time of Init, Algorithm 1, Algorithm 2). *Given $\{v_1, \cdots, v_n\}, n \in \mathbb{N}_+, d \in \mathbb{N}_+, k \in \mathbb{N}_+$, procedure Init stores vectors $\{v_1, \cdots, v_n\}$ into V, creates $\Phi \in \mathbb{R}^{m \times n}$ and stores ΦU into M, where U is in $\mathbb{R}^{n \times d^2}$ and i-th row of U is vectorization $v_i v_i^\top$. This procedure takes $O(d^2 n \log n)$ time.*

Proof. In Init, line 8 takes $O(nd)$ time to store n vectors in $\mathbb{R}^d$, line 9 takes $O(mn)$ time to creat Φ and line 10 takes $O(nd^2)$ time to calculate ΦU. The running time of InitParameters and InitSupport is $O(\log(k)\log(n/k))$, which can be ignored compared to other steps. Putting it all together, the running time of Init is

$$\begin{aligned} & O(nd) + O(nm) + O(nd^2) \\ = & O(n(m+d^2)) \end{aligned}$$

Thus, we complete the proof. □

Algorithm 1 StructuredFourier

data structure StructuredFourier ▷ Theorem 4.1
member
  $n, k, d \in \mathbb{N}_+$
  $\delta \in (0, 0.1)$
  $V \in \mathbb{R}^{n \times d}, M \in \mathbb{R}^{m \times d^2}, \Phi \in \mathbb{R}^{m \times n}$
  list $\mathcal{L}$ ▷ A list that maps $(G_{B,\alpha,\delta}, P_{\sigma,a,b})$ to a subset of $[n]$
procedure Init($v_1, \cdots, v_n \in \mathbb{R}^d, n \in \mathbb{N}_+, d \in \mathbb{N}_+, k \in \mathbb{N}_+, \delta \in (0, 0.1)$) ▷ Lemma B.1
  $V \leftarrow \{v_1, \cdots, v_n\}$ ▷ This step takes nd time
  Create Φ and $\mathcal{L}$ ▷ This step takes mn time
  $M \leftarrow \Phi U$ ▷ This step takes $O(nd^2)$ time
  InitParameters(k, n)
  InitSupport($\sigma, a, b, B, \alpha, \delta$)
end procedure
procedure Update($i \in [n], v_{\mathrm{new}} \in \mathbb{R}^d$) ▷ Lemma B.2
  Let $C \in \mathbb{R}^{n \times d^2}$ where i-th row is $\mathrm{vec}(v_{\mathrm{new}} v_{\mathrm{new}}^\top) - \mathrm{vec}(v_i v_i^\top)$ and all the other rows is 0
  $v_i \leftarrow v_{\mathrm{new}}$
  $M \leftarrow M + \Phi \cdot C$ ▷ This step takes $O(md^2)$
end procedure
procedure Query($h \in \mathbb{R}^{d^2}$) ▷ Lemma B.3, C.2
  $\widehat{z}^{(1)} \leftarrow 0$
  for $r_1 = 1 \to R_1$ **do**
    $L_{r_1} \leftarrow$ FindLargeIndex($h, \widehat{z}^{(r_1)}, r_1$) ▷ Lemma B.11, B.12, C.5
    $\widehat{z}^{(r_1+1)} \leftarrow \widehat{z}^{(r_1)}$ + Verified(L_{r_1})
  end for
  return $\widehat{z}^{(R_1+1)}$
end procedure
procedure Verified($h \in \mathbb{R}^{d^2}, L \subset [n]$) ▷ Lemma B.9
  return $(U \cdot h)_L$
end procedure
procedure ComputingMeasurements($h \in \mathbb{R}^{d^2}, L \subset [n]$) ▷ Lemma B.4
  return $(M \cdot h)_L$
end procedure
end data structure

B.2. Running Time of Update

In this section, we prove the running time of Update.

Lemma B.2 (Running Time of Update)**.** *Given index $i \in [n]$ and vector $z \in \mathbb{R}^d$, procedure Update takes i and v as inputs and runs in $O(md^2)$ time.*

Proof. In Update, line 14 takes $O(d)$ time, line 15 takes $O(d^2)$ time and line 17 takes $O(md^2)$. Putting it all together, the total running time of Update is

$$\begin{aligned} & O(d) + O(d^2) + O(md^2) \\ = & O(md^2) \end{aligned}$$

Thus, we compelte the proof. □

B.3. Running Time of Query

The goal of this section is to prove the running time of Query.

Lemma B.3 (Running time of Query)**.** *Given query $h \in \mathbb{R}^{d^2}$, procedure Query runs in time*

$$O(\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon} \log(n/\delta), d^2, 1) + k \log n \cdot \log(n/k))$$

Algorithm 2 Structured Fourier, Initialize Parameters

data structure StructuredFourier
members
t, R_1 ▷ quantity
$B, \sigma, \alpha, \overline{\alpha}, b, w, R_2, R_3$ ▷ one dimensional array
width ▷ two dimensional array
a, β ▷ three dimensional array
end members
procedure InitParameters(k, n)
  $t \leftarrow \log n$
  $R \leftarrow O(\log k / \log\log k)$
  /*One dimensional array*/
  for $r_1 = 1 = 1 \to R_1$ **do**
    $B[r_1] \leftarrow O(k\alpha_{r_1}/(\epsilon {r_1}^2))$
    $\alpha[r_1] \leftarrow O(1/{r_1}^2)$
    $\overline{\alpha}[r_1] \leftarrow \Theta(\alpha[r_1]^{1/3})$
    $\sigma[r_1] \leftarrow$ Choose from odd numbers in $[n]$ uniformly at random
    $b[r_1] \leftarrow$ Choose from $[n]$ uniformly at random
    $w[r_1] \leftarrow n/B[r_1]$
    $R_2[r_1] \leftarrow \log_{t/4}(w[r_1] + 1)$
    $R_3[r_1] = \Theta(\log_{1/\alpha}(t/\alpha))$
  end for
  /*Two dimensional array*/
  for $r_1 = 1 \to R_1$ **do**
    for $r_2 = 1 \to R_2[r_1]$ **do**
      $\mathsf{width}[r_1, r_2] \leftarrow w[r_1]/(t/4)^{r_2 - 1}$
    end for
  end for
  /*Three dimensional array*/
  for $r_1 = 1 \to R_1$ **do**
    for $r_2 = 1 \to R_2[r_1]$ **do**
      for $r_3 = 1 \to R_3[r_1]$ **do**
        Choose $a[r_1, r_2, r_3] \in [n]$ uniformly at random
        Choose $\beta[r_1, r_2, r_3] \in \{\frac{\overline{\alpha}[r_1]nt}{4\mathsf{width}[r_1, r_2]}, \cdots, \frac{\overline{\alpha}[r_1]nt}{2\mathsf{width}[r_1, r_2]}\}$ uniformly at random
      end for
    end for
  end for
end procedure
procedure InitSupport$(B, \alpha, \delta, \sigma, a, b)$
  $\mathcal{L}[B, \alpha, \delta, \sigma, a, b] \leftarrow$ ComputeSupport$(G_{B,\alpha,\delta}, P_{\sigma,a,b})$
    ▷ $[B, \alpha, \delta, \sigma, a, b]$ is multi-index that we go over to compute list $\mathcal{L}$
end procedure
end data structure

Proof. The running time of Query consists of three parts: computing sample, recovery, and verification.

By Lemma B.6, B.12, B.10, the running time of Query is

$$\begin{aligned}
&\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon}\log(n/\delta), d^2, 1) + O((\frac{k}{\epsilon}\log n)\cdot\log(n/k)) + \mathcal{T}_{\mathrm{mat}}(k, d, d) \\
=\ &\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon}\log(n/\delta), d^2, 1) + (\frac{k}{\epsilon}\log n)\cdot\log(n/k)
\end{aligned}$$

Thus, we complete the proof. □

Algorithm 3 Structured Fourier, Find Large Index

data structure StructuredFourier
procedure FindLargeIndex($h, \widehat{z}, r_1$)

  Initialize $\mathsf{left}_i^{(1)} = (i-1)w[r_1]$ for $i \in [B[r_1]]$.
  for $r_2 = 1 \to R_2[r_1]$ **do**
    $\mathsf{left}^{(r_2+1)} \leftarrow$ FindLargeIndexInner($h, \widehat{z}, r_1, r_2$)
  end for
  $L \leftarrow \{\pi_{\sigma,b}^{-1}(\mathsf{left}_j^{(R_2[r_1]+1)}) \mid j \in [B[r_1]]\}$
  return L
end procedure
procedure FindLargeIndexInner($h, \widehat{z}, r_1, r_2$)

  Let $\mathrm{vote}[j][q] = 0$ for $(j,q) \in [B[r_1]] \times [t]$.
  for $r_3 = 1 \to R_3[r_1]$ **do**
    $a_{\mathsf{u}} \leftarrow a[r_1, r_2, r_3]$
    $a_{\mathsf{v}} \leftarrow a[r_1, r_2, r_3] + \beta[r_1, r_2, r_3]$
    $\mathsf{u} \leftarrow$ StructuredHashToBins $(h, \widehat{z}, a_{\mathsf{u}}, r_1)$.
    $\mathsf{v} \leftarrow$ StructuredHashToBins $(h, \widehat{z}, a_{\mathsf{v}}, r_1)$.
    for $j = 1 \to B[r_1]$ **do**
      $c_j \leftarrow \phi(\mathsf{u}_j / \mathsf{v}_j)$
      for $q \in [t]$ **do**
        $\mathsf{mid}_{j,q} \leftarrow \mathsf{left}_j + \frac{q-1/2}{t}\mathsf{width}[r_1, r_2]$
        $\theta_{j,q} \leftarrow \frac{2\pi(\mathsf{mid}_{j,q} + \sigma[r_1]b[r_1])}{n} \bmod 2\pi$
        if $\min(|\beta[r_1,r_2,r_3]\theta_{j,q} - c_j|, 2\pi - |\beta[r_1,r_2,r_3]\theta_{j,q} - c_j|) < \overline{\alpha}\pi$ **then**
          $\mathrm{vote}[j][q] \leftarrow \mathrm{vote}[j][q] + 1$
        end if
      end for
    end for
  end for
  for $j \in [B[r_1]]$ **do**
    $Q^* \leftarrow \{q \in [t] \mid \mathrm{vote}[j][q] > R_{loc}/2\}$
    if $Q^* \neq \emptyset$ **then**
      $\widetilde{\mathsf{left}}_j \leftarrow \min_{q \in Q^*} \mathsf{left}_j + \frac{q-1}{t}\mathsf{width}[r_1, r_2]$
    else
      $\widetilde{\mathsf{left}}_j \leftarrow \perp$
    end if
  end for
  return $\widetilde{\mathsf{left}}$
end procedure
end data structure

B.4. Running Time of StructuredHashToBins

The goal of this section is to prove the running time of StructuredHashToBins.

We first show the running time for computing measurements.

Lemma B.4 (Running Time of ComputingMeasurements)**.** *Given an index set $L \subset [n]$, the running time of ComputingMeasurements is $\mathcal{T}_{\mathrm{mat}}(|L|, d^2, 1)$.*

Proof. We apply fast matrix computation to $(M)_L \cdot h$ directly to obtain the result. □

Lemma B.5 (Running Time of StructuredHashToBins, algorithm 4)**.** *Let $\eta = |\{i \in \mathrm{supp}(\widehat{z}) \mid E_{\mathrm{off}}(i)\}|$. The running time of StructuredHashToBins is $O(\mathcal{T}_{\mathrm{mat}}(\frac{B}{\alpha}\log(n/\delta), d^2, 1) + \|\widehat{z}\|_0 + \eta\log(n/\delta))$*

Proof. We can compute StructuredHashToBins via the following method:

Algorithm 4 StructuredHashToBins

procedure StructuredHashToBins($h \in \mathbb{R}^{d^2}, \widehat{z}, a, r_1$) ▷ Lemma A.6
▷ The filter function G and permutation P will decide that we only need a subset of x, where we can think of $\widehat{x} = Uh$.
  $L \leftarrow \mathcal{L}[B[r_1], \alpha[r_1], \delta, \sigma[r_1], a, b[r_1]]$
  $x_L \leftarrow$ CopmuteMeasurements(h, L)
  $y \leftarrow G_{B[r_1],\alpha[r_1],\delta} \cdot (P_{\sigma[r_1],a,b[r_1]} x_L)$
  for $j \in [B[r_1]]$ **do**
    $v_j \leftarrow \sum_{i=1}^{n/B[r_1]} y_{j+iB[r_1]}$ ▷ v is a vector in $\mathbb{C}^{B[r_1]}$
  end for
  $\widehat{y} \leftarrow \mathrm{FFT}(v)$ ▷ $\widehat{y}_{jn/B[r_1]} = \widehat{v}_j, j \in [B[r_1]]$
  for $i \in \mathrm{supp}(\widehat{z})$ **do**
    $\widehat{y}'_{\frac{n}{B[r_1]} h_{\sigma[r_1],b}(i)} \leftarrow \widehat{y}_{\frac{n}{B[r_1]} h_{\sigma[r_1],b}(i)} - \widehat{G'}_{-o_{(\sigma[r_1],b)}(i)} \widehat{z}_i w^{a\sigma[r_1]i}$
  end for
  return $\widehat{u}$ where $\widehat{u}_j = \widehat{y}_{j+iB[r_1]}, j \in [B[r_1]]$
end procedure

1. Compute y with $\|y\|_0 = O(\frac{B}{\alpha}\log(n/\delta))$ in $\mathcal{T}_{\mathrm{mat}}(\|y\|_0, d^2, 1)$ time.
2. Compute $v \in \mathbb{C}^B$ given by $v_i = \sum_j y_{i+jB}$.
3. Because B divides n, by the definition of the Fourier transform (see also Claim 3.7 of [3]) we have $\widehat{y}_{jn/B} = \widehat{v}_j$ for all j. Hence we can compute it with a B-dimensional FFT in $O(B\log B)$ time.
4. For each coordinate $i \in \mathrm{supp}(\widehat{z})$, decrease $\widehat{y}_{\frac{n}{B}h_{\sigma,b}(i)}$ by $\widehat{G'} - o_{\sigma,b(i)}\widehat{z}_i\omega^{a\sigma i}$. This takes $O(\|\widehat{z}\|_0 + \eta\log(n/\delta))$ time, since computing $\widehat{G'} - o_{\sigma,b}(i)$ takes $O(\log(n/\delta))$ time if $E_{\mathrm{off}}(i)$ holds and $O(1)$ otherwise (See section 7 in [2]).

Thus the total running time of StructuredHashToBins is

$$\begin{aligned}
&O(\mathcal{T}_{\mathrm{mat}}(\frac{B}{\alpha}\log(n/\delta), d^2, 1) + O(B\log B) + O(\|\widehat{z}\|_0 + \eta\log(n/\delta)) \\
=&O(\frac{B}{\alpha}\log(n/\delta) + \mathcal{T}_{\mathrm{mat}}(\frac{B}{\alpha}\log(n/\delta), d^2, 1) + \|\widehat{z}\|_0 + \eta\log(n/\delta)) \\
=&O(\mathcal{T}_{\mathrm{mat}}(\frac{B}{\alpha}\log(n/\delta), d^2, 1) + \|\widehat{z}\|_0 + \eta\log(n/\delta))
\end{aligned}$$

where the first step follows from $B\log B = O(\frac{B}{\alpha}\log(n/\delta)))$ and the last step follows from $\frac{B}{\alpha}\log(n/\delta)) \le \mathcal{T}_{\mathrm{mat}}(\frac{B}{\alpha}\log(n/\delta)), d, d)$

□

Using the runninig time of StructuredHashToBins, we prove the total running time for doing R rounds StructuredHashToBins with different parameters.

Lemma B.6. *Let $R_1 := O(\log k/\log\log k)$. It takes $\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon}\log(n/\delta), d^2, 1)$ time to run R_1 rounds of StructuredHashToBins , where $B_r := O(\frac{k\alpha_r}{\epsilon r^2})$ and $\alpha_r := \Theta(\frac{1}{r^2})$ in each round $r \in R_1$.*

Proof. By lemma B.5, the running time of StructuredHashToBins in each round r is

$$\begin{aligned}
&O(\mathcal{T}_{\mathrm{mat}}(\frac{B_r}{\alpha_r}\log(n/\delta), d^2, 1) + \|\widehat{z}^{(r)}\|_0(1 + \alpha_r\log(n/\delta))) \\
=\ &O(\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon r^2}\log(n/\delta), d^2, 1) + k(1 + \frac{1}{r^2}\log(n/\delta)))
\end{aligned}$$

$$= O(\frac{1}{r^2}(\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon}\log(n/\delta), d^2, 1) + k\log(n/\delta)) + k)$$

The total running time in R_1 rounds is

$$\begin{aligned}
&\sum_{r=1}^{R_1} O(\frac{1}{r^2}(\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon}\log(n/\delta), d^2, 1) + k\log(n/\delta)) + k) \\
&= O(\mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon}\log(n/\delta), d^2, 1) + k\log(n/\delta) + k\log k) \\
&= \mathcal{T}_{\mathrm{mat}}(\frac{k}{\epsilon}\log(n/\delta), d^2, 1)
\end{aligned}$$

where the second step follows from $\sum_{r=1}^{R_1} \frac{1}{r^2} = O(1)$.

Thus we complete the proof. □

B.5. Running Time of Verification

The goal of this section is to prove the running time of VERIFICATION.

Lemma B.7 (Naive Running)**.** *Given $L \subset [n]$, procedure VERIFY runs in $O(|L| \cdot d^2)$ time.*

Proof. For each $l \in L$, it takes d^2 time to compute $(Ud)_l$. Thus the total running time is $|L| \cdot d^2$. □

We show a naive running time in the lemma above. Actually, we can do better by batching vectors into matrix and applying fast matrix computing technique.

Fact B.8. *For any matrices in $\mathbb{R}^{a\times b}$ and $\mathbb{R}^{b\times c}$, the time of multiplying them is $\mathcal{T}_{mat}(a, b, c)$. In particular, we use ω to denote the exponent of matrix multiplication, which means $\mathcal{T}_{mat}(n, n, n) = n^\omega$. Currently, $\omega \approx 2.373$.*

We apply this fact to all analysis of matrix computation time.

Lemma B.9 (Improved Running Time For Verification)**.** *We can compute $(Uh)_L$ in the following time:*

- *If $|L| \leq d$, it takes $\mathcal{T}_{\mathrm{mat}}(d, |L|, d)$ time.*
- *If $|L| > d$, it takes $(|L|/d) \cdot \mathcal{T}_{\mathrm{mat}}(d, d, d)$ time.*

Proof. We divide our proof into three cases: $|L| < d$, $|L| = d$ and $|L| > d$.

For the case $|L| = d$, we can compute it in $\mathcal{T}_{\mathrm{mat}}(d, d, d) = d^\omega$ time.

Here is the reason. Without of loss generality, assume $L = \{1, 2, \cdots, d\}$. Let V in $\mathbb{R}^{d\times d}$ denote the matrix whose i-th row is v_i. Then $(VHV^\top)_{i,i} = (Uh)_i$

Say matrix H in $\mathbb{R}^{d\times d}$ is the matrix version of h in $\mathbb{R}^{d^2}$

Using the above equation to compute $(Uh)_L$, we can first compute $VHV^\top$, and then take the diagonal entries. Thus, if $|L| = d$, we can compute $(Uh)_L$ in d^ω which is faster than d^3 time.

Similarly, for any $|L| < d$, we can compute it in $\mathcal{T}_{\mathrm{mat}}(d, |L|, d)$ time.

For $|L| > d$, we can divide L into $|L|/d$ groups then we reduces the problem to the $|L| = d$ case. Thus it computes in $|L|/d \cdot \mathcal{T}_{\mathrm{mat}}(d, d, d)$ time. □

With running time of one time VERIFICATION, we prove the total running time for VERIFICATION through R iterations with decaying parameters.

Lemma B.10. *Let $R_1 := O(\log k/\log\log k)$. It takes $\mathcal{T}_{\mathrm{mat}}(k, d, d)$ time to run R rounds of VERIFICATION , where B_r is chosen to be $O(\frac{k}{\epsilon r^6})$ in each round $r \in R_1$.*

Proof. In total, VERIFICATION verify k locations, which takes $\mathcal{T}_{\mathrm{mat}}(k, d, d)$ time. □

B.6. Running Time of Recover

The goal of this section is to prove the running time of RECOVER. This is the time of FINDLARGEINDEX (Algorithm 3) that without considering StructuredHashToBins time.

Lemma B.11. *Suppose $B = \frac{Ck}{\alpha^2\epsilon}$ for C larger than some fixed constant. The procedure FINDLARGEINDEX runs in expected time*

$$O(B \log n \cdot \log(n/B))$$

Proof. Set $t = \log n, t' = t/4$ and $R_3 = O(\log_{1/\alpha}(t/\alpha))$. Let $\mathsf{width}_0 = n/B$ and $\mathsf{width}_{r_2} = \mathsf{width}_0/(t')^{r_2-1}$, so $\mathsf{width}_{R_2+1} < 1$ for $R_2 = \log_{t'}(\mathsf{width}_0 + 1) < t$.

The running time of FINDLARGEINDEX mainly comes from the forloop in line 5, which has R_2 iterations. In each iteration, it invokes FINDLARGEINDEXINNER.

The running time of FINDLARGEINDEXINNER mainly comes from three recursive forloop (line 14 - 29). Notice that all operations in the forloop takes $O(1)$ time. Thus the running time for FINDLARGEINDEXINNER is $O(R_3 \cdot B \cdot t)$.

Since $R_3R_2 = O(\log_{1/\alpha}(t/\alpha)\log_t(n/B)) = O(\log(n/B))$, the running time for FINDLARGEINDEX is

$$\begin{aligned} &O(R_2 \cdot R_3 \cdot B \cdot t) \\ =&O(B \log n \cdot \log(n/B)) \end{aligned}$$

Thus we complete the proof. □

Using above result, we are able to prove the total running time for FINDLARGEINDEX that runs R rounds with different parameters.

Lemma B.12. *Let $R_1 := O(\log k/\log\log k)$. It takes $O(\frac{k}{\epsilon}\log n \cdot \log(n/k))$ time to run R_1 rounds of FINDLARGEINDEX , where B_r is chosen to be $O(\frac{k}{\epsilon r^6})$ in each round $r \in R_1$.*

Proof. By lemma B.11, the running time in each round is

$$\begin{aligned} &O(B_r \log n \cdot \log(n/B_r)) \\ =&O(\frac{k}{\epsilon r^6}\log n)(\log(n/k) + \log(\epsilon) + \log(r))) \\ =&O(\frac{k}{\epsilon r^2}\log n)\log(n/k)) \end{aligned}$$

The total running time in R_1 rounds is

$$\begin{aligned} &\sum_{r=1}^{R_1} O(\frac{k}{\epsilon r^2}\log n \log(n/k)) \\ =&O(\frac{k}{\epsilon}\log n \cdot \log(n/k)) \end{aligned}$$

where the last step follows from $\sum_{r=1}^{R_1}\frac{1}{r^2} = O(1)$.

Thus, we complete the proof. □

C. Correctness Proof

This section is organized as follows: In section C.1, we first bound the error of QUERY in each iteration up to a $(1 \pm \epsilon)$ multiplicative error , which is based on the results in Section C.3, Section C.4 and Section C.5. We demonstrate the correctness of QUERY in section C.2. In QUERY procedure, we

run several rounds. Using the result in Section C.1, we show that the cumulative error after several iterations is still bounded by a $(1+\epsilon)$ multiplicative error, which is due to the geometric decay of errors in each round. Section C.3 presents the correctness of StructuredHashToBin. We state that the result of StructuredHashToBin is close to the large coefficient. We begin with the correctness of FindLargeIndexInner in section C.4. We show that it finds out intervals that contain large coefficients with high probability. Next, we demonstrate the correctness of FindLargeIndex in Section C.5 combining result in Section C.4. We show how FindLargeIndex finds index set of large coefficient by setting appropriate parameters and analyze the time complexity.

C.1. Correctness of Query in each round

From now on, we treat vector $Uh \in \mathbb{R}^n$ as a whole. We will show that $Uh - \widehat{z}^{(r)}$ gets sparser as r increases, with only a mild increase in the error.

Theorem C.1. *Define $Uh^{(r)} = Uh - \widehat{z}^{(r)}$. Consider any one loop r of Query, running with parameters $(B, k, \alpha) = (B_r, k_r, \alpha_r)$ such that $B \geq \frac{Ck}{\alpha^2\epsilon}$ for some C larger than some fixed constant. Then for any $\xi > 0$,*

$$\operatorname{error}^2(Uh^{(r+1)}, \xi k) \leq (1+\epsilon)\operatorname{error}^2(Uh^{(r)}, k) + O(\epsilon\delta^2 n\|Uh\|_1^2)$$

with probability $1 - O(\alpha/\xi)$

Proof. Recall that in round r, $\mu^2 = \frac{\epsilon}{k}(\operatorname{error}^2(Uh^{(r)}, k) + \delta^2 n\|Uh\|_1^2)$ and $S = \{i \in [n] | Uh_i^{(r)}|^2 > \mu^2\}$. By Lemma C.5 each $i \in S$ lies in L_r with probability at least $1 - O(\alpha)$. Hence $|S \backslash L| < \xi k$ with probability at least $1 - O(\alpha/\xi)$. Then

$$\begin{aligned}
\operatorname{error}^2(Uh^{(r)}_{[n]\backslash L}, \xi k) &\leq \|Uh^{(r)}_{[n]\backslash(L\cup S)}\|_2^2 \\
&\leq \operatorname{error}^2(Uh^{(r)}_{[n]\backslash(L\cup S)}, k) + k\|Uh^{(r)}_{[n]\backslash(L\cup S)}\|_\infty^2 \\
&\leq \operatorname{error}^2(Uh^{(r)}_{[n]\backslash L}, k) + k\mu^2 \qquad (2)
\end{aligned}$$

Let $\widehat{w} = \widehat{z}^{(r+1)} - \widehat{z}^{(r)} = Uh^{(r)} - Uh^{(r+1)}$ by the vector recovered by Verified. Then $\operatorname{supp}(\widehat{w}) = L$, so

$$\begin{aligned}
\operatorname{error}^2(Uh^{(r+1)}, 2\xi k) &= \operatorname{error}^2(Uh^{(r)} - \widehat{w}, 2\xi k) \\
&\leq \operatorname{error}^2(Uh^{(r)}_{[n]\backslash L}, \xi k)
\end{aligned}$$

But by Eq. (2), this gives

$$\begin{aligned}
\operatorname{error}^2(Uh^{(r+1)}, 2\xi k) &\leq \operatorname{error}^2(Uh^{(r)}_{[n]\backslash L}, k) + O(k\mu^2) \\
&\leq \operatorname{error}^2(Uh^{(r)}, k) + O(k\mu^2) \\
&\leq (1 + O(\epsilon))\operatorname{error}^2(Uh^{(r)}, k) + O(\epsilon\delta^2 n\|Uh\|_1^2)
\end{aligned}$$

The result follows from rescaling ξ and ϵ by constant factors.

□

C.2. Correctness of Query

The goal of this section is to prove the correctness of Query. Again we define $Uh^{(r)} = Uh - \widehat{z}^{(r)}$.

Lemma C.2 (Correctness of Query)**.** *Given query $h \in \mathbb{R}^{d^2}$, procedure Query in Algorithm 1 takes h as input and outputs a $k-$sparse vector $y \in \mathbb{R}^n$ satisfying*

$$\|y - Uh\| \leq \min_{k-\text{sparse } z\in\mathbb{R}^n} (1+\epsilon)\|z - Uh\| + \delta\|Uh\|_2$$

Proof. Define $\xi_r = O(1/r^2)$ so $\sum \xi_r < 1/4$. Choose R so $\prod_{r=1}^R \xi_r < 1/k \leq \prod_{r=1}^{R-1} \xi_r$. Then $R = O(\log k/\log\log k)$, since $\prod_{r=1}^R \xi_r < (\xi_{R/2})^{R/2} = (2/R)^R$.

Set $\epsilon_r = \xi_r\epsilon, \alpha_r = \Theta(\xi_r^2), k_r = k\prod_{i=1}^{r-1}\xi_i, B_r = O(\frac{k}{\epsilon}\alpha_r\xi_r)$. Then $B_r = \omega(\frac{k_r}{\alpha_r^2\epsilon_r})$, so for sufficiently large constant the constraint of Theorem C.1 is satisfied. For appropriate constants, Theorem C.1 says that in each round r,

$$\begin{aligned}\operatorname{error}^2(Uh^{(r+1)}, k_{r+1}) &= \operatorname{error}^2(Uh^{(r+1)}, \xi_r k_r)\\ &\le (1+\xi_r\epsilon)\operatorname{error}^2(Uh^{(r)}, k_r) + O(\xi_r\epsilon\delta^2 n\|Uh\|_1^2)\end{aligned}$$

with probability at least $1-\xi_r$. The error accumulates, so in round r we have

$$\operatorname{error}^2(Uh^{(r)}, k_r) \le \operatorname{error}^2(Uh, k)\prod_{i=1}^{r-1}(1+\xi_i\epsilon) + \sum_{i=1}^{r-1} O(\xi_r\epsilon\delta^2 n\|Uh\|_1^2)\prod_{j=i+1}^{r-1}(1+\xi_j\epsilon)$$

with probability at least $1-\sum_{i=1}^{r-1}\xi_i > 3/4$. Hence in the end, since $k_{R+1} = k\prod_{i=1}^{R}\xi_i < 1$,

$$\begin{aligned}\|Uh^{(R+1)}\|_2^2 &= \operatorname{error}^2(Uh^{(R+1)}, k_{R+1})\\ &\le \operatorname{error}^2(Uh, k)\prod_{i=1}^{R}(1+\xi_i\epsilon) + O(R\epsilon\delta^2 n\|Uh\|_1^2)\prod_{i=1}^{R}(1+\xi_i\epsilon)\end{aligned}$$

with probability at least $3/4$. We also have

$$\prod_{i=1}^{R}(1+\xi_i\epsilon) \le e^{\epsilon\sum_{i=1}^{R}\xi_i} \le e$$

making

$$\prod_{i=1}^{R}(1+\xi_i\epsilon) \le 1 + e\sum_{i=1}^{R}\xi_i\epsilon < 1+2\epsilon$$

Thus we get the approximation factor

$$\|Uh - \widehat{z}^{(R+1)}\|_2^2 \le (1+2\epsilon)\operatorname{error}^2(Uh, k) + O((\log k)\epsilon\delta^2 n\|Uh\|_1^2)$$

with at least $3/4$ probability. Rescaling δ by poly (n), using $\|Uh\|_1^2 \le n\|Uh\|_2$, and taking the square root gives the desired

$$\|Uh - \widehat{z}^{(R+1)}\|_2 \le (1+\epsilon)\operatorname{error}(Uh, k) + \delta\|Uh\|_2.$$

□

C.3. Correctness of StructuredHashToBin

The goal of this section is to prove the correctness of StructuredHashToBin.

Lemma C.3 (Correctness of StructuredHashToBin)**.** *Let $a \in [n]$ uniformly at random, B divide n, and the other parameters be arbitrary in StructuredHashToBin (Algorithm 4).*

Let

$$\mathsf{u} = \textit{StructuredHashToBin}(h, \widehat{z}, a, r)$$

Then for any $i \in [n]$ with $j = h_{\sigma[r],b[r]}(i)$ and none of $E_{\mathrm{coll}}(i), E_{\mathrm{off}}(i)$, or $E_{\mathrm{noise}}(i)$ holding,

$$\mathbb{E}[|\mathsf{u}_j - Uh_i'\omega^{a\sigma[r]i}|^2] \le 2\frac{\rho^2}{\alpha[r]B[r]}$$

Proof. Let $\widehat{G'} = \widehat{G'}_{B[r],\delta,\alpha[r]}$. Let $T = h^{-1}_{\sigma[r],b[r]}(j)\backslash\{i\}$. We have that $T\cap S = \emptyset$ and $\widehat{G'} - o_{\sigma[r],b[r]}(i) = 1$. By Lemma A.6

$$\mathsf{u}_j - Uh'_i\omega^{a\sigma[r]i} = \sum_{i'\in T} \widehat{G}'_{-o_\sigma[r](i')} Uh'_{i'}\omega^{a\sigma[r]i'} \pm \delta\|Uh\|_1$$

Because the $\sigma i'$ are distinct for $i' \in T$, we have by Parseval's theorem

$$\mathop{\mathbb{E}}_a[|\sum_{i'\in T} \widehat{G'} - o_{\sigma[r]}(i')Uh'_{i'}\omega^{a\sigma[r]i'}|^2] = \sum_{i'\in T}(\widehat{G'} - o_{\sigma[r]}(i')Uh'_{i'})^2 \le \|Uh'_T\|_2^2$$

Since $|X+Y|^2 \le 2|X|^2 + 2|Y|^2$ for any X, Y, we get

$$\begin{aligned}\mathop{\mathbb{E}}_a[|\mathsf{u}_j - Uh'_i\omega^{a\sigma[r]i}|^2] &\le 2\|Uh'_T\|_2^2 + 2\delta^2\|Uh\|_1^2 \\ &\le 2\,\mathrm{error}^2(Uh', k)/(\alpha[r]B[r]) + 2\delta^2\|Uh\|_1^2 \\ &\le 2\rho^2/(\alpha[r]B[r])\end{aligned}$$

□

C.4. Correctness of FindLargeIndexInner

The goal of this section is to prove Lemma C.4.

The proof of this lemma is standard in literature, such as [2, 6, 7].

Lemma C.4 (Correctness of FindLargeIndexInner)**.** *Let $i \in S$. Suppose none of $E_{\mathrm{coll}}(i), E_{\mathrm{off}}(i)$, and $E_{\mathrm{noise}}(i)$ hold, and let $j = h_{\sigma,b}(i)$. Consider any run of FindLargeIndexInner (Algorithm 3) with $\pi_{\sigma,b}(i) \in [\mathsf{left}_j, \mathsf{left}_j + \mathsf{width}]$. Let $\xi > 0$ be a parameter such that*

$$B = \frac{Ck}{\alpha\xi\epsilon}$$

for C larger than some fixed constant. Then $\pi_{\sigma,b}(i) \in [\mathsf{left}'_j, \mathsf{left}'_j + 4\mathsf{width}/t]$ with probability at least $1 - t\xi^{\Omega(R_3)}$.

Proof. Let $\tau = \pi_{\sigma,b}(i) \equiv \sigma(i-b) (\mathrm{mod} n)$, and for any $j \in [n]$ define

$$\theta^*_j = \frac{2\pi}{n}(j + \sigma b) \pmod{2\pi}$$

so $\theta^*_\tau = \frac{2\pi}{n}\sigma i$. Let $g = \Theta(\xi^{1/3})$, and $C' = \frac{B\alpha\epsilon}{k} = \Theta(1/g^3)$.

To get the result, we divide $[\mathsf{left}_j, \mathsf{left}_j + \mathsf{width}]$ into t "regions", $Q_q = [\mathsf{left}_j + \frac{q-1}{t}\mathsf{width}, \mathsf{left}_j + \frac{q}{t}\mathsf{width}]$ for $q \in [t]$. We will first show that in each round r, c_j is close to $\beta\theta^*_\tau$ with $1-g$ probability. This will imply that Q_q gets a "vote," meaning $\mathrm{vote}[j][q]$ increases, with $1-g$ probability for the q' with $\tau \in Q_{q'}$. It will also imply that $\mathrm{vote}[j][q]$ increases with only g probability when $|q - q'| > 3$. Then R_3 rounds will suffice to separate the two with $1 - \xi^{-\Omega(R_3)}$ probability. We get that with $1 - t\xi^{-\Omega(R_3)}$ probability, the recovered Q^* has $|q - q'| \le 3$ for all $q \in Q^*$. If we take the minimum $q \in Q^*$ and the next three sub-regions, we find τ to within 4 regions, or $4\mathsf{width}/t$ locations, as desired.

In any round r, define $\mathsf{u} = \mathsf{u}^{(r)}$ and $a = a_r$. We have by Lemma C.3 and that $i \in S$ that

$$\begin{aligned}\mathbb{E}[|\mathsf{u}_j - \omega^{a\sigma i}Uh'_i|^2] &\le 2\frac{\rho^2}{\alpha B} \\ &= \frac{2k}{B\alpha\epsilon}\mu^2 \\ &= \frac{2}{C'}\mu^2 \\ &\le \frac{2}{C'}|Uh'_i|^2.\end{aligned}$$

where the first step follows from definition of Lemma C.3, the second step follows from definition of μ, the third step follows from definition of C', and the last step follows from definition of S.

Note that $\phi(\omega^{a\sigma i}) = -a\theta^*_\tau$. Thus for any $p > 0$, with probability $1 - p$ we have

$$|\mathsf{u}_j - \omega^{a\sigma i} U h_i| \le \sqrt{\frac{2}{C'p}} |U h'_i| \tag{3}$$

$$\|\phi(\mathsf{u}_j) - (\phi(U h'_i) - a\theta^*_\tau)\|_\bigcirc \le \sin^{-1}(\sqrt{\frac{2}{C'p}}) \tag{4}$$

where $\|x - y\|_\bigcirc = \min_{\gamma \in \mathbb{Z}} |x - y + 2\pi\gamma|$ denotes the "circular distance" between x and y. The analogous fact holds for $\phi(\mathsf{v}_j)$ relative to $\phi(U h'_i) - (a+\beta)\theta^*_\tau$. Therefore with at least $1 - 2p$ probability,

$$\begin{aligned}
\|c_j - \beta\theta^*_\tau\|_\bigcirc &= \|\phi(\mathsf{u}_j) - \phi(\mathsf{v}_j) - \beta\theta^*_\tau\|_\bigcirc \\
&= \|(\phi(\mathsf{u}_j) - (\phi(U h'_i) - a\theta^*_\tau)) - (\phi(\mathsf{v}_j) - (\phi(U h'_i) - (a+\beta)\theta^*_\tau))\|_\bigcirc \\
&\le \|\phi(\mathsf{u}_j) - (\phi(U h'_i) - a\theta^*_\tau)\|_\bigcirc + \|\phi(\mathsf{v}_j) - (\phi(U h'_i) - (a+\beta)\theta^*_\tau)\|_\bigcirc \\
&\le 2\sin^{-1}(\sqrt{\frac{2}{C'p}})
\end{aligned}$$

where the first step follows from definition of c_j, the second step follows from interpolating $\phi(U h'_i) - a\theta^*_\tau$, the third step follows from triangular inequality, and the last step follows from Eq. (4).

by the triangle inequality. Thus for any $s = \Theta(g)$ and $p = \Theta(g)$, we can set $C' = \frac{2}{p \sin^2(\overline{\alpha}\pi/4)} = \Theta(1/g^3)$ so that

$$\|c_j - \beta\theta^*_\tau\|_\bigcirc < \overline{\alpha}\pi/2 \tag{5}$$

with probability at least $1 - 2p$.

Eq. (5) shows that c_j is a good estimate for i with good probability. We will now show that this means the approprate "region" $Q_{q'}$ gets a "vote" with "large" probability.

For the q' with $\tau \in [\mathsf{left}_j + \frac{q'-1}{t}\mathsf{width}, \mathsf{left}_j + \frac{q'}{t}\mathsf{width}]$, we have that $\mathsf{mid}_{j,q} = \mathsf{left}_j + \frac{q'-1/2}{t}\mathsf{width}$ satisfies

$$|\tau - \mathsf{mid}_{j,q}| \le \frac{\mathsf{width}}{2t}$$

so

$$|\theta^*_\tau - \theta_{j,q'}| \le \frac{2\pi}{n}\frac{\mathsf{width}}{2t} \tag{6}$$

Hence by Eq. (5), the triangle inequality, and the choice of $B \le \frac{snt}{2w}$,

$$\begin{aligned}
\|c_j - \beta\theta_{j,q'}\|_\bigcirc &\le \|c_j - \beta\theta^*_\tau\|_\bigcirc + \|\beta\theta^*_\tau - \beta\theta_{j,q'}\|_\bigcirc \\
&< \frac{\overline{\alpha}\pi}{2} + \frac{\beta\pi\mathsf{width}}{nt} \\
&\le \frac{\overline{\alpha}\pi}{2} + \frac{\overline{\alpha}\pi}{2} \\
&= \overline{\alpha}\pi
\end{aligned}$$

where the first step follows from triangular inequality, the second step follows from Eq. (5) and Eq. (6), and the third step follows from $\beta \le \frac{\overline{\alpha}nt}{2\mathsf{width}}$.

Thus, $\mathrm{vote}[j][q']$ will increase in each round with probability at least $1 - 2p$.

Now, consider q with $|q - q'| > 3$. Then $|\tau - \mathsf{mid}_{j,q}| \ge \frac{7\mathsf{width}}{2t}$, and (from the definition of $\beta > \frac{\overline{\alpha}nt}{4\mathsf{width}}$) we have

$$\beta|\tau - \mathsf{mid}_{j,q}| \ge \frac{7\overline{\alpha}n}{8} > \frac{3sn}{4} \tag{7}$$

We now consider two cases. First, suppose that $|\tau - \mathsf{mid}_{j,q}| \le \frac{\mathsf{width}}{\overline{\alpha}t}$. In this case, from the definition of β it follows that

$$\beta|\tau - \mathsf{mid}_{j,q}| \le n/2$$

Together with Eq. (7) this implies

$$\Pr[\beta(\tau - \mathsf{mid}_{j,q}) \bmod n \in [-3\overline{\alpha}n/4, 3\overline{\alpha}n/4]] = 0$$

On the other hand, suppose that $|\tau - \mathsf{mid}_{j,q}| > \frac{\mathsf{width}}{\overline{\alpha}t}$. In this case, we use Lemma A.1 with parameters $l = 3\overline{\alpha}n/2, m = \frac{\overline{\alpha}nt}{2\mathsf{width}}, t = \frac{\overline{\alpha}nt}{4\mathsf{width}}, i = (\tau - \mathsf{mid}_{j,q})$ and $n = n$, to conclude that

$$\begin{aligned}
&\Pr[\beta(\tau - \mathsf{mid}_{j,q}) \bmod n \in [-3\overline{\alpha}n/4, 3\overline{\alpha}n/4]] \\
&\le \frac{4\mathsf{width}}{\overline{\alpha}nt} + 2\frac{|\tau - \mathsf{mid}_{j,q}|}{n} + 3\overline{\alpha} + \frac{3\overline{\alpha}n}{2}\frac{\overline{\alpha}t}{\mathsf{width}}\frac{4\mathsf{width}}{\overline{\alpha}nt} \\
&\le \frac{4\mathsf{width}}{\overline{\alpha}nt} + \frac{2\mathsf{width}}{n} + 9\overline{\alpha} \\
&< \frac{6}{\overline{\alpha}B} + 9\overline{\alpha} \\
&< 10\overline{\alpha}
\end{aligned}$$

where the first step follows from Lemma A.1, the second step follows from $|i| = |\tau - \mathsf{mid}_{j,q}| \le \mathsf{width}$, the third step follows from assumption $\frac{\mathsf{width}}{\overline{\alpha}t} < |i| \le \mathsf{width} \le n/B$, and the last step follows from $6/B < \overline{\alpha}^2$.

Thus in either case, with probability at least $1 - 10\overline{\alpha}$ we have

$$\|\beta\theta_{j,q} - \beta\theta^*_\tau\|_\bigcirc = \|\frac{2\pi\beta(\mathsf{mid}_{j,q} - \tau)}{n}\|_\bigcirc > \frac{2\pi}{n}\frac{3\overline{\alpha}n}{4} = \frac{3}{2}\overline{\alpha}\pi$$

for any q with $|q - q'| > 3$. Therefore we have

$$\|c_j - \beta\theta_{j,q}\|_\bigcirc \ge \|\beta\theta_{j,q} - \beta\theta^*_\tau\|_\bigcirc - \|c_j - \beta\theta^*_\tau\|_\bigcirc > \overline{\alpha}\pi$$

with probability at least $1 - 10\overline{\alpha} - 2p$, and $\mathrm{vote}[j][q]$ is not incremented.

To summarize: in each round, $\mathrm{vote}[j][q']$ is incremented with probability at least $1-2p$ and $\mathrm{vote}[j][q]$ is incremented with probability at most $10\overline{\alpha} + 2p$ for $|q - q'| > 3$. The probabilities corresponding to different rounds are independent.

Set $\overline{\alpha} = g/20$ and $p = g/4$. Then $\mathrm{vote}[j][q']$ is incremented with probability at least $1-g$ and $\mathrm{vote}[j][q]$ is incremented with probability less than g. Then after R_3 rounds, if $|q - q'| > 3$,

$$\begin{aligned}
\Pr[\mathrm{vote}[j][q] > R_3/2] &\le \binom{R_3}{R_3/2} g^{R_3/2} \\
&\le (4g)^{R_3/2} \\
&= \xi^{\Omega(R_3)}
\end{aligned}$$

where the first step follows from $\sum_{i=R_3/2+1}^{R_3}\binom{R_3}{i} \le \binom{R_3}{R_3/2}$, the second step follows from $\binom{R_3}{R_3/2} \le 2^{R_3}$, and the last step follows from $g = \xi^{1/3}/4$.

$$\Pr[\mathrm{vote}[j][q'] < R_3/2] \le \xi^{\Omega(R_3)}$$

Hence with probability at least $1 - t\xi^{\Omega(R_3)}$ we have $q' \in Q^*$ and $|q - q'| \le 3$ for all $q \in Q^*$. But then $\tau - \mathsf{left}'_j \in [0, 4\mathsf{width}/t]$ as desired.

Because

$$\mathbb{E}[|\{i \in \mathrm{supp}(\widehat{z}) \mid E_{\mathrm{off}}(i)\}|] = \alpha\|\widehat{z}\|_0,$$

the expected running time is

$$O(R_3Bt + R_3\frac{B}{\alpha}\log(n/\delta) + R_3\|\widehat{z}\|_0(1 + \alpha\log(n/\delta))).$$

□

C.5. Correctness of FindLargeIndex

The goal of this section is to prove the correctness of FindLargeIndex.

The proof of this lemma is standard in literature, see Lemma 4.5 in [2].

Lemma C.5. *Suppose $B = \frac{Ck}{\alpha^2\epsilon}$ for C larger than some fixed constant. The procedure FindLargeIndex in Algorithm 2 returns a set L of size $|L| \le B$ such that for any $i \in S, \Pr[i \in L] \ge 1 - O(\alpha)$.*

Proof. Consider any $i \in S$ such that none of $E_{\text{coll}}(i), E_{\text{off}}(i)$, and $E_{\text{noise}}(i)$ hold, as happens with probability $1 - O(\alpha)$.

Set $t = \log n, t' = t/4$ and $R_3 = O(\log_{1/\alpha}(t/\alpha))$. Let $\mathsf{width}_0 = n/B$ and $\mathsf{width}_{r_2} = \mathsf{width}_0/(t')^{r_2-1}$, so $\mathsf{width}_{R_2+1} < 1$ for $R_2 = \log_{t'}(\mathsf{width}_0 + 1) < t$.

In each round r_2, Lemma C.4 implies that if

$$\pi_{\sigma,b}(i) \in [\mathsf{left}_j^{(r_2)}, \mathsf{left}_j^{(r_2)} + \mathsf{width}_{r_2}]$$

then

$$\pi_{\sigma,b}(i) \in [\mathsf{left}_j^{(r_2+1)}, \mathsf{left}_j^{(r_2+1)} + \mathsf{width}_{r_2+1}]$$

with probability at least $1 - \alpha^{\Omega(R_3)} = 1 - \alpha/t$.

By a union bound, with probability at least $1 - \alpha$ we have

$$\pi_{\sigma,b}(i) \in [\mathsf{left}_j^{(R_2+1)}, \mathsf{left}_j^{(R_2+1)} + \mathsf{width}_{R_2+1}] = \{\mathsf{left}_j^{(R_2+1)}\}.$$

Thus $i = \pi_{\sigma,b}^{-1}(\mathsf{left}_j^{(R_2+1)}) \in L$. □

D. Proof of Main Result

In this section, we combine results in Section B and Section C to state the proof for our main theorem.

Theorem D.1 (Restatement of Theorem 4.1)**.** *Given a collection of vectors $v_1, \cdots, v_n \in \mathbb{R}^d$. Let $U \in \mathbb{R}^{n \times d^2}$ denote the matrix where the i-th row is the vectorization of $v_i v_i^\top \in \mathbb{R}^{d \times d}$. There is a data structure that uses $O(md^2 + nd)$ space that supports the following operations:*

- *Init$(v_1, \cdots, v_n \subset \mathbb{R}^d, n \in \mathbb{N}_+, d \in \mathbb{N}_+, k \in \mathbb{N}_+)$, this operation takes $n(m + d^2)$ time. This operation takes $v_1, \cdots, v_n$ as inputs. It stores $V \in \mathbb{R}^{n \times d}$, and Fourier measurement $\Phi U \in \mathbb{R}^{m \times d^2}$ (where $\Phi = S \cdot F$, $S \in \mathbb{R}^{m \times n}$ is a subsampled matrix where each row only has a single nonzero entries, $F \in \mathbb{C}^{n \times n}$ is a discrete Fourier matrix)*
- *Update$(i \in [n], v_{\mathbf{new}} \in \mathbb{R}^d)$, it takes index i and vector v as input, runs in $O(md^2)$ time, and replaces v_i with $v_{\mathbf{new}}$.*
- *Query$(h \in \mathbb{R}^{d^2})$, it takes vector $h \in \mathbb{R}^{d^2}$ as input, it outputs k-sparse $y \in \mathbb{R}^n$ such that*

$$\|y - Uh\|_2 = \min_{k-\text{sparse } z} (1+\epsilon)\|z - Uh\|_2 + \delta\|Uh\|_2. \tag{8}$$

  This operation takes

$$O(\mathcal{T}_{\text{mat}}(\epsilon^{-1}k\log(n/\delta), d^2, 1) + k\log n \cdot \log(n/k))$$

Proof. We aggregate the proof for this theorem as follow:

- By Lemma B.1, the running time of Init is

$$O(n(m + d^2))$$

  In Init, it stores matrix $V \in \mathbb{R}^{n \times d}$ and $\Phi U \in \mathbb{R}^{m \times d^2}$, which takes $O(md^2 + nd)$ space.

  Thus, we prove the space storage of our data structure.

- By Lemma B.2, the running time of Update is

$$O(md^2)$$

- By Lemma B.3, the running time of Query is

$$O(\mathcal{T}_{\mathrm{mat}}(\epsilon^{-1}k\log(n/\delta), d^2, 1) + k\log n \cdot \log(n/k))$$

  By Lemma C.2, Query takes vector $h \in \mathbb{R}^{d^2}$ as input and outputs a k-sparse vector $y \in \mathbb{R}^n$ satisfying

$$\|y - Uh\| \leq \min_{k-\mathrm{sparse} z \in \mathbb{R}^n} (1+\epsilon)\|z - Uh\| + \delta\|Uh\|_2$$

  Thus, we prove the correctness of Query.

Thus, we complete the proof for the whole theorem. □

E. Discussion on SDP

One of the major motivation of our work is matrix sensing. In this section, we provide another motivation which is from rank-1 semidefinite programming. Semdefinite programming [68, 69] can be defined as follows

Definition E.1 (Semidefinite programming)**.** *Given a collection of symmetric matrices $C, A_1, \cdots, A_n \in \mathbb{R}^{d\times d}$ and a vector $n \in \mathbb{R}^n$, the goal is to solve*

$$\begin{aligned} \max_X \ & \langle C, X\rangle \\ \text{s.t.} \ & \langle A_i, X\rangle = b_i, \forall i \in [n] \\ & X \succeq 0 \end{aligned}$$

where $\langle Y, X\rangle$ denote the inner product between two matrices Y and X.

Using robust central path method [68, 69] to solve semidefine programming requires to form a Hessian matrix which can be written as

$$\mathsf{Hessian} = \mathsf{A}(S^{-1} \otimes S^{-1})\mathsf{A}^\top$$

where $S \in \mathbb{R}^{d\times d}$ is the slack matrix, $S^{-1}\otimes S^{-1}$ forms a matrix that has size $d^2 \times d^2$, and $\mathsf{A} \in \mathbb{R}^{n\times d^2}$ is matrix where i-th row of vectorization of A_i. If we impose the additional rank-1 constraint to each A_i, then A becomes our matrix U exactly.

In each iteration of robust central path method [68, 69], we need to compute $\mathsf{Hessian} \cdot v$ for some $v \in \mathbb{R}^n$ which can be viewed as $\mathsf{A}\cdot h$ where $h = (S^{-1}\otimes S^{-1})\mathsf{A}\cdot v \in \mathbb{R}^{d^2}$.

F. Exact Recovery

We provide a case where Query procedure could recover k-sparse vector exactly.

Lemma F.1. *For $|S| = k$ where $S = \{i \in [n] | |(Uh)_i| > 1\}$ and $|(Uh)_i| < 1/\mathrm{poly}(n)$ for $i \in [n]\setminus S$, our Query procedure can recover k-sparse $y \in \mathbb{R}^d$ exactly, i.e.*

$$\|y - Uh\|_2 \leq \min_{k-\mathrm{sparse}\ z} \|z - Uh\|_2.$$

Proof. In this case, all the index sets that FindLargeIndex returns will cover the top-k coordinate. By our Verification procedure, we find their magnitudes directly. Thus, we obtain a k-sparse recovery exactly. □

G. Limitations

While our research proposes a method to separate signal estimation from frequency estimation for clarity and specialized focus, it might also limit the integration and holistic understanding of how these components interact within the broader scope of Sparse Fourier Transform applications. Also, our work is based on some assumptions such as the independence of inputs, the noiselessness of settings, and the well-specification of the model. These assumptions may not hold in practical scenarios.

H. Impact Statement

Our research suggested an efficient algorithm to compute Sparse Fourier Transform. Future researchers could potentially reduce the use of power and electricity when running real-world coding experiments. Since our research is purely theoretical, there is no negative impacts on society.