Dimension-independent Sparse Fourier Transform

Michael Kapralov
EPFL
michael.kapralov@epfl.ch

Ameya Velingker
EPFL
ameyav@google.com

Amir Zandieh
EPFL
amir.zandieh@epfl.ch

February 19, 2019

Abstract

The Discrete Fourier Transform (DFT) is a fundamental computational primitive, and the fastest known algorithm for computing the DFT is the FFT (Fast Fourier Transform) algorithm. One remarkable feature of FFT is the fact that its runtime depends only on the size N of the input vector, but *not on the dimensionality of the input domain*: FFT runs in time $O(N \log N)$ irrespective of whether the DFT in question is on $\mathbb{Z}_N$ or $\mathbb{Z}_n^d$ for some $d > 1$, where $N = n^d$.

The state of the art for Sparse FFT, i.e. the problem of computing the DFT of a signal that has at most k nonzeros in Fourier domain, is very different: all current techniques for sublinear time computation of Sparse FFT incur an exponential dependence on the dimension d in the runtime. In this paper we give the first algorithm that computes the DFT of a k-sparse signal in time $\mathrm{poly}(k, \log N)$ *in any dimension* d, avoiding the curse of dimensionality inherent in all previously known techniques. Our main tool is a new class of filters that we refer to as *adaptive aliasing filters*: these filters allow isolating frequencies of a k-Fourier sparse signal using $O(k)$ samples in time domain and $O(k \log N)$ runtime per frequency, in any dimension d.

We also investigate natural average case models of the input signal: **(1)** worst case support in Fourier domain with randomized coefficients and **(2)** random locations in Fourier domain with worst case coefficients. Our techniques lead to an $\widetilde{O}(k^2)$ time algorithm for the former and an $\widetilde{O}(k)$ time algorithm for the latter.

1 Introduction

The Discrete Fourier Transform (DFT) is one of the most widely used computational primitives in modern computing, with numerous applications in data analysis, signal processing, and machine learning. The fastest algorithm for computing the DFT is the Fast Fourier Transform (FFT) algorithm of Cooley and Tukey, which has been recognized as one of the 10 most important algorithms of the 20th century [Cip00]. The FFT algorithm is very efficient: it computes the Discrete Fourier Transform of a length N complex-valued signal in time $O(N \log N)$. This applies to vectors in any dimension: FFT works in $O(N \log N)$ time irrespective of whether the DFT is on the line, on a $\sqrt{N} \times \sqrt{N}$ grid, or is in fact the Hadamard transform on $\{0,1\}^d$, with $d = \log_2 N$.

In any applications of the Discrete Fourier Transform, the input signal $x \in \mathbb{C}^N$ often satisfies *sparsity* or *approximate sparsity* constraints: the Fourier transform $\widehat{x}$ of x has a small number of coefficients k or is close to a signal with a small number of coefficients (e.g., this phenomenon is the motivation for compression schemes such as JPEG and MPEG). This has motivated a rich line of work on the *Sparse FFT* problem: given access to a signal $x \in \mathbb{C}^N$ in time domain that is sparse in Fourier domain, compute the k nonzero coefficients in *sublinear* (i.e., $o(N)$) time.

Very efficient algorithms for the Sparse FFT problem have been developed in the literature [GL89, KM91, Man92, GGI+02, AGS03, GMS05, Iwe10a, Aka10, HIKP12b, HIKP12a, LWC12, BCG+12, HAKI12, PR13, HKPV13, IKP14, IK14, Kap16, PS15, CKPS16, Kap17]. The state-of-the-art approach, due to [HIKP12a], yields an $O(k \log N)$ runtime algorithm for the following exact k-sparse Fourier transform problem: given access to an input signal of length N whose Fourier transform has at most k nonzeros, output the nonzero coefficients and their values. This highly efficient algorithm comes with a caveat, however: the runtime of $O(k \log N)$ only holds for the Fourier transform on the line, namely, $\mathbb{Z}_N$. The algorithm naturally extends to higher dimensions, namely, $\mathbb{Z}_n^d$, where $N = n^d$, but with an exponential loss in runtime; the runtime becomes $O(k \log^d N)$ as opposed to $O(k \log N)$. Interestingly, the other extreme of $d = \log_2 N$, i.e., the Hadamard transform, has been known to admit an $O(k \log N)$ algorithm since the seminal work of Goldreich and Levin [GL89]. However, all intermediate values of d exhibit a *curse of dimensionality*. This is in sharp contrast with FFT itself, which runs in time $O(N \log N)$, where $N = n^d$ is the length of the input signal, in *any dimension* d. The focus of our work is to design sublinear time algorithms for Sparse FFT that avoid this curse of dimensionality. Our main point of attention is the Sparse FFT problem:

$$\begin{aligned} \textbf{Input:} \quad & \text{access to } x : [n]^d \to \mathbb{C}, \\ & \text{integer } k \geq 1 \text{ such that } |\operatorname{supp} \widehat{x}| \leq k \\ \textbf{Output:} \quad & \text{nonzero elements of } \widehat{x} \text{ and their coefficients} \end{aligned} \tag{1}$$

Our main result is the first sublinear algorithm for exact Sparse FFT (1), as stated in the following theorem.

Theorem 1 (Main result, informal version of Theorem 3 in Section 2.1)**.** *For any integer n that is a power of two and any positive integer d, there exists a deterministic algorithm that, given access to a signal $x \in \mathbb{C}^{n^d}$ with $\|\widehat{x}\|_0 \leq k$, recovers $\widehat{x}$ in time* $\operatorname{poly}(k, \log N)$.

We note that this is the first sublinear time Sparse FFT algorithm that avoids an exponential dependence on the dimension d. One should note that the runtime still depends on d, since $\log_2 N = d \log_2 n$ is lower bounded by d, but this dependence is polynomial as opposed to exponential.

1.1 Significance of our results and related work

Significance of our results. The state of the art in high dimensional Sparse Fourier Transforms presents an interesting conundrum: algorithms with runtime $O(k \log N)$ are known for $d = 1$ (Discrete Fourier Transform on the line, see [HIKP12a]) and $d = \log_2 N$ (the Hadamard transform, see [GL89]), but for all intermediate values of d the runtime scales exponentially in d. Given that FFT itself is dimension-insensitive, this strongly suggests that exciting new algorithmic techniques can be developed for the high-dimensional version of the problem. Our paper designs the first approach to high dimensional Sparse FFT that does not suffer from the curse of dimensionality, and naturally leads to several exciting open problems that we hope will spur further progress in this area.

In addition, we note that rather high-dimensional versions of the Fourier transform arise in applications (e.g., 2D, 3D and 4D-NMR in medical imaging), and designing practical Sparse FFT algorithms for this regime is an important problem. We hope that new techniques for dimension-independent Sparse FFT will lead to progress in this direction as well.

Sample complexity of high-dimensional Sparse FFT. We note that, besides runtime, another very important parameter of a Sparse FFT algorithm is *sample complexity*, i.e., the number of samples that an algorithm needs to access in time domain in order to compute the top few coefficients of the Fourier transform. The sample complexity of Sparse FFT, unlike runtime, does not suffer from a curse of dimensionality. Indeed, there exist several algorithms with $\widetilde{O}(N)$ runtime that can recover the top k coefficients of $\widehat{x}$ using only $k \operatorname{poly}(\log N)$ accesses in time domain, irrespective of the dimensionality of the problem. This can be achieved, for example, using either results on the restricted isometry property (RIP) [CT06, RV08, Bou14, CGV12, HR17], or using the filtering approach developed in the Sparse FFT literature, with $\widetilde{O}(N)$ decoding time. Thus, the challenge is to achieve sublinear *runtime* without an exponential dependence on the dimension.

We now outline existing approaches to Sparse FFT and explain why they fail to scale well in high dimensions:

State-of-the-art approaches to Sparse FFT and their lack of scalability in high dimensions. The main idea behind many recently developed algorithms for the Sparse FFT problem is the "hashing" approach inherited from sparse recovery with arbitrary linear measurements. Given access to a signal $x : [n]^d \to \mathbb{C}$, one designs linear measurements of x that allow one to "hash" the nonzero positions of $\widehat{x}$ into a number of "buckets." The number of buckets $B = b^d$ is chosen to be a constant factor larger than the sparsity k to ensure that a large constant fraction of the nonzero positions of $\widehat{x}$ are isolated in their buckets. Every isolated element can be recovered and subtracted from x for future iterations of the same hashing scheme, thereby ensuring convergence. The idea of hashing is implemented via filtering: one designs a filter $G : [n]^d \to \mathbb{C}$ such that $\widehat{G}$ approximates a "bucket," i.e., $\widehat{G}$ is close to 1 on an ℓ_∞ ball of side length $\approx (N/B)^{1/d} = n/b$ in dimension d. The content of the $\boldsymbol{j}$-th 'bucket', for $\boldsymbol{j} \in [\boldsymbol{B}]$, is then

$$\widehat{(x_{\cdot - a} \cdot G)}_{\boldsymbol{j} \cdot n/b} = \sum_{\boldsymbol{f} \in [n]^d} \widehat{x}_{\boldsymbol{f}} e^{2\pi \boldsymbol{f}^T a/n} \cdot \widehat{G}_{\boldsymbol{j} \cdot n/b - \boldsymbol{f}}. \tag{2}$$

Since $\widehat{G}$ is essentially 1 on the ℓ_∞ ball around the center $\boldsymbol{j} \cdot n/b$ of the 'bucket' and essentially zero outside, (2) gives the algorithm time domain access to the restriction of $\widehat{x}$ to the "bucket,"

i.e., the essential support of $\widehat{G}$, where $a \in [n]^d$ is the location in time domain at which the signal is being accessed. A pseudorandom permutation of the frequency space ensures that such a bucket is likely to contain just a single element of the support, which enables the algorithm to recover at least a constant fraction of elements in a single round and perform iterative recovery. Furthermore, if the (essential) support of G in time domain is small, one obtains an efficient algorithm.

The difficulty that arises in using (2) in high dimensions is the fact that it is not known how to ensure that $\widehat{G}$ is close to 1 in an appropriately defined "bucket" while simultaneously ensuring that |supp G| is small. For example, the filters constructed in [HIKP12a] ensure that $\widehat{G}$ is polynomially close to 1 in Fourier domain, but this comes at the expense of |supp G| being larger than k (the ideal support size) by a factor of $\Theta(\log n)$, and this effect is even more pronounced in higher dimensions, resulting in a $\log^d n$ loss in runtime. The other extreme would be to choose G to be equal to 1 on an ℓ_∞ ball with k points around the origin, but in that case, its Fourier transform $\widehat{G}$ is the sinc function, which is only a constant factor approximation to the indicator of the corresponding ℓ_∞ box in Fourier domain (i.e., the ideal "bucket"). In dimension d, the approximation degrades to c^d for some constant $c \in (0, 1)$, leading to exponential loss in runtime. Indeed, suppose that all elements of $\widehat{x}$ have roughly the same value. Then for a given element $\boldsymbol{f} \in$ supp $\widehat{x}$, the expected contribution of other elements to the noise in the "bucket" that $\boldsymbol{f}$ is hashed to is $||\widehat{x}||_2^2/B$, but the contribution of $\widehat{x}_{\boldsymbol{f}}$ to its own bucket is (most of the time) only c^d of its value, and, hence, only an exponentially small fraction of coefficients can be recovered in a given round of hashing. [1]

Related work. In [CI17], the authors presented a deterministic Sparse Fourier transform algorithm for the Hadamard transform, i.e., $d = \log_2 N$, that runs in nearly linear time in the sparsity parameter k, but it is not known how this extends to lower dimensions. In [Iwe10b, Iwe12] the author gives a $\widetilde{O}(k^2)$ time deterministic algorithm for the Sparse Fourier Transform, but the algorithm only applies to a related but distinctly easier problem. Specifically, the problem considers a continuous function on $[0, 2\pi)$ whose Fourier transform is bandlimited and sparse. The presented algorithm requires sampling the signal at arbitrary locations in $[0, 2\pi)$. A natural approach is to emulate sampling off-grid (i.e., at arbitrary points in $[0, 2\pi)$) given discrete samples that we have access to, which is achieved in [MZIC17] giving an $\widetilde{O}(k^2)$ time deterministic algorithm for one dimensional sparse FFT. But this is a challenging task in multi-dimensional setting for several reasons. First, we are operating under the sparsity assumption alone, and no powerful general interpolation techniques that work under the sparsity assumption alone are available, to the best of our knowledge. Furthermore, even if the function were bandlimited, a natural approach to interpolation would involve some form of Taylor expansion or semi-equispaced Fourier Transform, however, both approaches incur a $\log^d N$ loss in dimension d. Indeed, similar exponential dependence on the dimensionality of the problem manifests itself in Fast Multipole Methods [GR87b, BG97] and the Sparse FFT algorithms mentioned above. Finally, one should also note that whereas the problem of computing the Fourier transform on a $p \times q$ grid with p mutually prime with q is equivalent to a one-dimensional Fourier transform on $\mathbb{Z}_{pq}$, the standard case of side lengths that are powers of two (for which we have the most efficient FFT algorithms) does not admit such a reduction. Furthermore, such a reduction appears to be quite challenging

[1] In addition, the discussion above assumes the presence of an approximate pairwise hashing lemma for high dimensions that does not lose an exponential factor in the dimension (it is known that such a lemma holds with at most about a factor of 2^d loss [IK14], but no dimension-independent version is available in the literature).

in high dimensions for reasons outlined above, and even more so for highly oscillatory functions that Sparse FFT algorithms need to handle.

2 Overview of our results and techniques

Prior works on Sparse FFT have primarily focused on efficiently implementing hashing-based ideas developed in the extensive literature on sparse recovery using general linear measurements (e.g., [GHI+13]), which meets with several difficulties. In particular, the presence of multiplicative subgroups in $\mathbb{Z}_n^d$ has been a hurdle in analyzing Sparse FFT algorithms: while aliasing filters have optimal performance from the point of view of the uncertainty principle, their applications have been limited due to the fact that frequencies that belong to the same subgroup get hashed together if such filters are used, making it impossible to reason about isolation of individual frequencies. At the same time, FFT itself owes much of its efficiency to the very same multiplicative subgroups of $\mathbb{Z}_n^d$, and a natural question is whether one can design a Sparse FFT algorithm that operates on similar principles. This is precisely the approach that we take.

Adaptive aliasing filters. The main technical innovation that allows us to avoid exponential dependence on the dimension and obtain Theorem 1 is a new family of filters for isolating a subset of frequencies in Fourier domain in a sparse signal $\widehat{x}$ using few samples in time domain. We refer to the family of filters as *adaptive aliasing filters.*

Definition 1 ($(\boldsymbol{f}, S)$-isolating filter, informal version of Definition 21, see Section 4)**.** Suppose n is a power of two integer and $S \subseteq [n]^d$ for a positive integer d. Then, for any frequency $\boldsymbol{f} \in S$, a filter $G : [n]^d \to \mathbb{C}$ is called $(\boldsymbol{f}, S)$-*isolating* if $\widehat{G}_{\boldsymbol{f}} = 1$ and $\widehat{G}_{\boldsymbol{f}'} = 0$ for every $\boldsymbol{f}' \in S \setminus \{\boldsymbol{f}\}$.

We explain the intuition behind the construction of the filter in Section 2.1 below and provide the details later in Section 4.

The reason why an $(\boldsymbol{f}, S)$-isolating filter G is useful lies in the fact that for every signal $x \in \mathbb{C}^{n^d}$ with supp $\widehat{x} \subseteq S$ we have, for all $\boldsymbol{t} \in [n]^d$

$$\sum_{\boldsymbol{j} \in [n]^d} x_{\boldsymbol{j}} G_{\boldsymbol{t}-\boldsymbol{j}} = (x * G)_{\boldsymbol{t}} = \frac{1}{N} \sum_{\boldsymbol{j} \in [n]^d} \widehat{x}_{\boldsymbol{j}} \cdot \widehat{G}_{\boldsymbol{j}} \cdot e^{2\pi i \frac{\boldsymbol{j}^T \boldsymbol{t}}{n}} = \frac{1}{N} \widehat{x}_{\boldsymbol{f}} e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{t}}{n}}$$

Thus, the filter G enables access to the time domain representation of the restriction of $\widehat{x}$ to $\boldsymbol{f}$ in time proportional to $|\text{supp } G|$, at any point $\boldsymbol{t}$. Of course, this is only useful if the support of G is small. The main technical lemma of our paper shows that for every support set $S \subseteq \widehat{x}$, there exists an $\boldsymbol{f} \in S$ that can be isolated efficiently:

Lemma 1 (Informal version of Corollary 2 in Section 4)**.** *For every power of two $n \geq 1$, positive integer d, and set $S \subseteq [n]^d$, there exists an $\boldsymbol{f} \in S$ and an $(\boldsymbol{f}, S)$-isolating filter G such that* $|\text{supp } G| \leq |S|$.

The proof of the lemma is given in Section 4.

Accessing the residual signal. Lemma 1 suggests a natural approach to the estimation problem with Fourier measurements in high dimensions: iteratively construct an $(\boldsymbol{f}, S)$-isolating filter G, estimate $\boldsymbol{f}$, remove $\boldsymbol{f}$ from S, and proceed. The hope is that we can essentially assume that we are given access to $\mathcal{F}^{-1}(\widehat{x}_{S \setminus \{\boldsymbol{f}\}})$ once we have estimated $\boldsymbol{f}$. In general, if we have been

able to estimate the values of $\widehat{x}_{\boldsymbol{f}}$ for all $\boldsymbol{f} \in C$ with some $C \subseteq S$, then we would like to obtain access to

$$x_{\boldsymbol{t}} - \sum_{\boldsymbol{f} \in C} \widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \boldsymbol{f}^T \boldsymbol{t}}.$$

Note that we would need $x_{\boldsymbol{t}}$ for $\boldsymbol{t}$ in the support of G at the next iteration, and this support is generally a rather complicated set of size $\Omega(k)$, from which we need to subtract the inverse Fourier transform of the signal estimated so far. This problem is the non-uniform Fourier transform problem, and no subquadratic methods for subtraction are known even in dimension $d = 1$ when the set in time domain that we want to compute the inverse Fourier transform on is arbitrary. Even if the target set is an ℓ_∞-box, the best known algorithms for this problem run in time $\Omega(k \log^d(1/\epsilon))$, where $\epsilon > 0$ is the precision parameter of the computation—this reduces to quadratic time even when $d = \Omega(\log k / \log\log k)$ and inverse polynomial in k precision is desired. Thus, subtracting from time domain would result in at least cubic runtime in k. Instead, we subtract the influence of the residual in frequency domain, which requires $O(k)$ evaluations of $\widehat{G}$ (as we show, $\widehat{G}$ can be evaluated at a cost of just $O(\log N)$). Note that it is crucial here that we peel off one coefficient at at time. Any improvements to this process, if they were to achieve $k^{2-\Omega(1)}$ runtime overall, would likely also imply improvements in the computation of approximate *non-uniform* Fourier transform: given a k-sparse signal $\widehat{x}$ and a set $T \subseteq [n]^d$ with $|T| \leq k$, output $y : [n]^d \to \mathbb{C}$ such that $||(x-y)_T||_2^2 \leq \epsilon ||x||_2^2$. However, it seems plausible that quadratic runtime in k is essentially optimal for the non-uniform Fourier transform problem: specifically, that under natural complexity theoretic assumptions there exists no algorithm for the ϵ-approximate non-uniform Fourier transform problem with runtime $k^{2-\Omega(1)}$ when $d = \Omega(\log k)$ and $\epsilon < 1/k^C$ for sufficiently large constant C. We note that current techniques do not provide a subquadratic algorithm even for simple sets T such as the ℓ_∞ box with k points in dimension $d = \Omega(\log k / \log\log k)$ (due to the $k \log^d(1/\epsilon)$ dependence mentioned above; a similar exponential dependence on the dimension is present in Fast Multipole Methods [GR87a, BG]). For an arbitrary set T no subquadratic algorithm is known even when $d = 1$.

Putting it together: estimation with Fourier measurements Combining the aforementioned ideas, we are able to develop a deterministic algorithm for the *estimation problem with Fourier measurements* in high dimensions:

$$\begin{aligned} \textbf{Input:} \quad & \text{access to } x : [n]^d \to \mathbb{C}, \\ & \text{subset } S \subseteq [n]^d \text{ such that supp } \widehat{x} \subseteq S \\ \textbf{Output:} \quad & \widehat{x}_S \end{aligned} \tag{3}$$

For the estimation problem (3) we obtain the following result.

Theorem 2 (Estimation guarantee, informal version of Theorem 8 in Section 5)**.** *Suppose n is a power of two integer, d is a positive integer, and $S \subseteq [n]^d$. Then, for any signal $x \in \mathbb{C}^{n^d}$ with* supp $\widehat{x} \subseteq S$*, the procedure* Estimate(x, S, n, d) *(see Algorithm 2) recovers $\widehat{x}$. Moreover, the sample complexity of this procedure is $O(|S|^2)$ and its runtime is $O(|S|^2 \cdot \log N)$. Furthermore, the procedure* Estimate *is deterministic.*

In the rest of this section, we give an overview of our techniques. Throughout the section, we present our results for the one-dimensional setting, as this makes notation simpler. All our results translate to the high-dimensional setting without any loss—see Section 4.2 for details.

2.1 Recovery via adaptive aliasing filters

Our main theorem is the following, which presents an algorithm for problem (1) for worst-case signals.

Theorem 3 (Sparse FFT for worst-case signals)**.** *For any power of two integer n and any positive integer d and any signal $x \in \mathbb{C}^{n^d}$ with $\|\widehat{x}\|_0 = k$, the procedure* SPARSEFFT(x, n, d, k) *in Algorithm 4 recovers $\widehat{x}$. Moreover, the sample complexity of this procedure is $O(k^3 \log^2 k \log^2 N)$ and its runtime is $O(k^3 \log^2 k \log^2 N)$.*

The major difference between estimation and recovery (i.e., problem (3) vs. (1)) is the fact in the latter problem, the set S of frequencies is unknown to us: the algorithm is only given access to x and an upper bound on the sparsity of $\widehat{x}$. Since our (f, S)-separating filter is adaptive, i.e., depends on S, this appears to present a challenge. However, we circumvent this challenge by constructing a sequence of successive approximations to the set S. In dimension 1, these approximations amount to reducing S modulo 2^j for all $j = 1, \ldots, \log_2 n$, and adaptively probing to learn which of the residue classes are nonzero. As before, our approach extends seamlessly to high dimensions by simply concatenating the d coordinates into a single vector. Note that this is in sharp contrast to all previously known approaches, which are more efficient in low dimensions, but incur an exponential loss overall. We would like to note that at a high level one can view our filtering approach as a way to prune the FFT computation graph in a way that suffices for recovery of a k-Fourier sparse vector.

We outline the main ideas in one-dimensional setting here to simplify the presentation (see Section 4.2 for the high-dimensional version of the argument). Let $N = n$ be the length of the signal and $d = 1$ be the dimension for n a power of two. We define T_n^{full} to be a full binary tree of height $\log_2 n$ and define a labelling scheme on the vertices as follows.

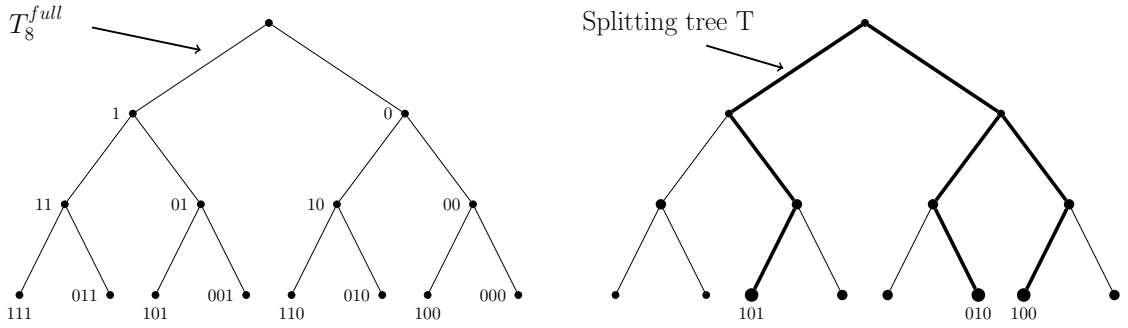
Definition 2. Suppose n is a power of two. Let T_n^{full} be a full binary tree of height $\log_2 n$, where for every $j \in \{0, 1, \ldots, \log_2 n\}$, the nodes at level j (i.e., at distance j from the root) are labeled with integers in $[2^j]$. For a node $v \in T_n^{\text{full}}$, we let f_v be its label. The label of the root is $f_{root} = 0$. The labelling of T_n^{full} satisfies the condition that for every $j \in [\log_2 n]$ and every v at level j, the right and left children of v have labels f_v and $f_v + 2^j$, respectively. Note that the root of T_n^{full} is at level 0, while the leaves are at level $\log_2 n$.

The tree captures the computation graph of FFT algorithm, where leaves correspond to frequencies in $[n]$ (given by the label), and for any $j \in \{0, 1, \ldots, \log_2 n\}$, the nodes at level j (i.e., at distance j from the root) correspond to congruence classes of frequencies modulo 2^j, as specified by the labelling (see Figure 1a).

Note that the full FFT algorithm starts from the root of T_n^{full} and computes the congruence classes of the Fourier transform of signal x at each level of this tree iteratively. Because it can reuse the computations from each level for computing the next levels, the total runtime of FFT is $O(n \log_2 n)$.

In order to speed up the computation for sparse signals, we introduce the notion of a *splitting tree*, which is nothing but the subtree of T_n^{full} that contains the nonzero locations of $\widehat{x}$ together with paths connecting them to the root. Given a set $S \subseteq [n]$ (the support of $\widehat{x}$ in Fourier domain), we define the *splitting tree* of the set S as follows:

Definition 3 (Splitting tree)**.** Let n be a power of two. For every $S \subseteq [n]$, the *splitting tree* $T = \text{Tree}(S, n)$ of a set S is a binary tree that is the subtree of T_n^{full} that contains, for every $j \in [\log_2 n]$, all nodes $v \in T_n^{\text{full}}$ at level j such that $\{f \in S : f \equiv f_v \pmod{2^j}\} \neq \emptyset$.



(a) T_8^{full} with binary labelling.

(b) A splitting tree of depth 3 with three leaves. Set of leaves $S = \{2, 4, 5\}$.

Figure 1: An example of T_n^{full} and a splitting tree with $n = 8$ and binary labelling.

An illustration of such a tree is given in Figure 1b. In order to recover the identities of the elements in S, our algorithm performs an exploration of this tree. At every point, the algorithm constructs a filter G that isolates *frequencies in a given subtree* and tests whether that subtree contains a nonzero signal. In order to make this work, we need a construction of filters that isolates the entire subtree as opposed to only one element, as Definition 1 does. Fortunately, the actual (f, S)-isolating filters G constructed in Lemma 1 satisfy precisely this property. The stronger isolation properties are captured by the following definition:

Definition 4 (Frequency cone of a leaf of T)**.** For every power of two n, subtree T of T_n^{full}, and vertex $v \in T$ which is at level $l_T(v)$ from the root, define the *frequency cone of v with respect to T* as

$$\text{FrequencyCone}_T(v) := \left\{ f \in [n] : f \equiv f_v \pmod{2^{l_T(v)}} \right\}.$$

Intuitively, the frequency cone of a node v in T captures all potential nonzeros of $\widehat{x}$ that belong to the subtree of v in T (see Figure 2). Our adaptive filter construction lets us obtain time domain access to the corresponding part of the frequency space:

Definition 5 ((v, T)-isolating filter)**.** For every integer n, subtree T of T_n^{full}, and leaf v of T, a filter $G \in \mathbb{C}^n$ is called (v, T)-*isolating* if the following conditions hold:

- For all $f \in \text{FrequencyCone}_T(v)$, we have $\widehat{G}_f = 1$.
- For every $f' \in \bigcup_{\substack{u \neq v \\ u\text{: leaf of } T}} \text{FrequencyCone}_T(u)$, we have $\widehat{G}_{f'} = 0$.

Note that for all signals $x \in \mathbb{C}^n$ with $\operatorname{supp} \widehat{x} \subseteq \bigcup_{u\text{: leaf of } T} \text{FrequencyCone}_T(u)$ and $t \in [n]$,

$$\sum_{j \in [n]} x_j G_{t-j} = \frac{1}{n} \sum_{f \in \text{FrequencyCone}_T(v)} \widehat{x}_f e^{2\pi i \frac{ft}{n}}.$$

Iterative tree exploration process leading to an algorithm with $\widetilde{O}(k^3)$ runtime. Now that we have defined the framework for our algorithm, we need to specify the order in which the algorithm will be accessing the leaves of the tree in order to minimize runtime. This is governed by the cost of constructing and using a (v, T)-isolating filter for various nodes v in T. To quantify cost, we introduce the notion of a *weight* of a leaf in the tree.

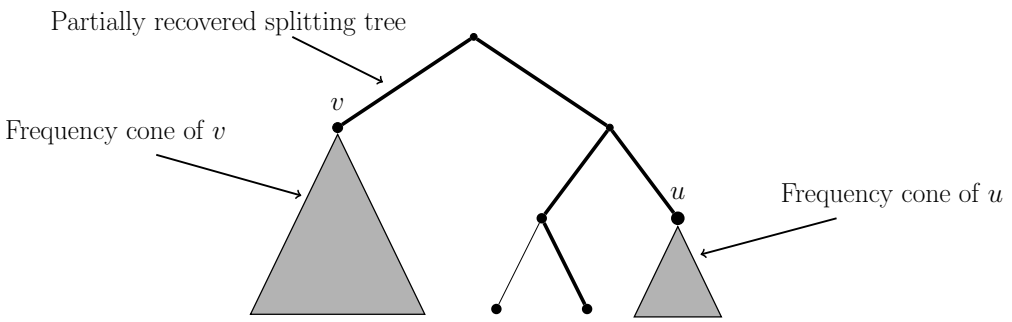


Figure 2: A partially recovered splitting tree (shown in bold). Frequency cones of u and v correspond to the subtrees rooted at nodes u and v, respectively, which have not been discovered yet.

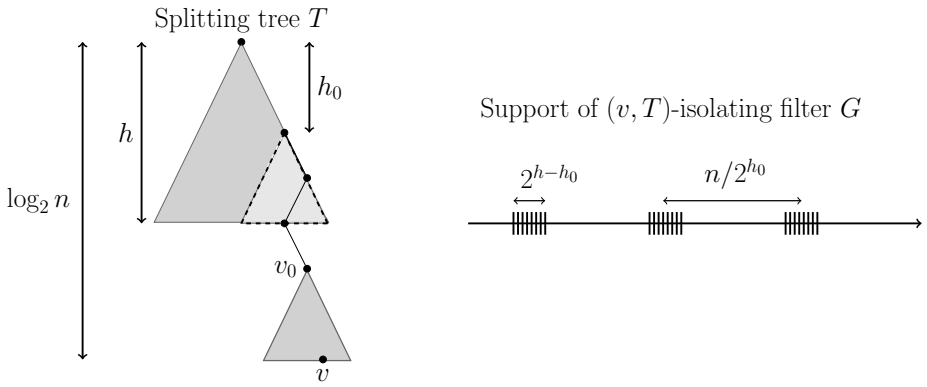


Figure 3: An instance of a (v, T)-isolating filter G, where the weight of leaf v is h and hence the filter G satisfies $|\text{supp } G| = 2^{w_T(v)} = 2^h$.

Definition 6 (Weight of a leaf)**.** Suppose n is a power of two. Let T be a subtree of T_n^{full}. Then for any leaf $v \in T$, we define its *weight* $w_T(v)$ *with respect to* T to be the number of ancestors of v in tree T with two children.

It turns out that the techniques from Lemma 1 also yield the following.

Lemma 2 (Informal version of Lemma 3 in Section 4)**.** *Suppose n is a power of two. Let T be a subtree of T_n^{full}. Then for any leaf $v \in T$, there exists a (v, T)-isolating filter G with $|supp\ G| \le 2^{w_T(v)}$ such that G and $\widehat{G}$ can be evaluated at $O(\log N)$ cost per point.*

Before describing the algorithm we give an example illustrating filter support in time domain. Consider a complete binary tree T of height $h \ll \log_2 n$. Suppose that v_0 is some vertex at level $h_0 < h$ of this tree. Now we take the subtree rooted at v_0 and add an appendage of length $\log_2 n - h$ to v_0. The appendage is a path of $\log_2 n - h$ nodes each of which has a single child. This doesn't change the weight of any of the leafs of the original tree because every node on the appendage has exactly one child. One can see an example of such tree in Fig. 3. Suppose that the leaf v is a leaf of the subtree rooted at v_0, which is moved far from the root by the appendage. In order to isolate v from the elements that are not in the subtree of v_0 we need a filter which is $(n/2^{h_0})$-periodic in time domain and in order to isolate from the rest of the elements in subtree of v_0 the filter needs to sample the signal at a fine grid of length 2^{h-h_0}. Note

that the support of a (v, T)-isolating filter G is $\operatorname{supp} G = \left\{ i + (n/2^{h_0}) \cdot j;\, j \in [2^{h_0}], i \in [2^{h-h_0}] \right\}$. In Fig. 3 we exhibit a (v, T)-isolating filter G which is constructed based on Lemma 3, where v and T correspond to this instance of splitting tree.

Given Lemma 2, our algorithm is natural. We find the vertex $v^* = \operatorname{argmin}_{v \in T} w_T(v)$, which, by Kraft's inequality, satisfies $w_T(v^*) \leq \log_2 k$. We then define an auxiliary tree T' by appending a left a and a right child b to v. Then for each of the children a, b, we, in turn, construct a filter G that isolates them from the rest of T (i.e., from the frequency cones of other nodes in T) and check whether the corresponding restricted signals are nonzero. The latter is unfortunately a nontrivial task, since the sparsity of these signals can be as high as k, and detecting whether a k-sparse signal is nonzero requires $\Omega(k)$ samples. However, a fixed set of $k \log^3 N$ locations that satisfies the restricted isometry property (RIP) can be selected, and accessing the signal on those values suffices to test whether it is nonzero. The overall runtime becomes $\widetilde{O}(k^3)$: the isolating filter has support at most $2k$, while the number of samples needed to test whether the two subtrees of v are nonempty is $\widetilde{O}(k)$, so peeling off $\leq k$ elements takes $\widetilde{O}(k^3)$ time overall. This results in Theorem 3 (the algorithm is presented as Algorithm 4).

$\widetilde{O}(k^2)$ runtime under random phase assumption. We note that the runtime can be easily reduced to $\widetilde{O}(k^2)$ if assumptions are made on the signal that ensure that its energy is evenly spread across time domain, making $\widetilde{O}(1)$ samples sufficient to detect whether the signal is zero or not. This occurs, for instance, if a signal's frequencies satisfy distributional assumptions (e.g., the values have random phases). We present such a result in Section 7. It seems that even under this assumption on the values of the signal, since the support of the signal in Fourier domain is worst case, reducing the runtime below k^2 likely requires a major advance in techniques for non-uniform Fourier transform computation.

More formally, we introduce the notion of a *worst-case signal with random phase* as follows:

Definition 7 (Worst-case signal with random phase)**.** For any positive integer d and power of two n, we define x to be a *worst-case signal with random phase* having values $\{\beta_{\boldsymbol{f}}\}_{\boldsymbol{f} \in [n]^d}$ if

$$\widehat{x}_{\boldsymbol{f}} = \beta_{\boldsymbol{f}} e^{2\pi i \theta} \quad \text{for uniformly random } \theta \in [0, 2\pi),$$

independently for every $\boldsymbol{f} \in [n]^d$. Furthermore, if k of the values $\{\beta_{\boldsymbol{f}}\}_{\boldsymbol{f} \in [n]^d}$ are nonzero, then x is said to be a *worst-case k-sparse signal with random phase* and is guaranteed to have sparsity k.

Note that "worst-case" in the above definition signifies the fact that the *support* of the signal is arbitrary (having no distributional assumptions), subject to a potential sparsity constraint. We then present the following theorem:

Theorem 4 (Sparse FFT for worst-case signals with random support)**.** *For any power of two integer n, positive integer d, and worst-case k-sparse signal with random phase $x \in \mathbb{C}^{n^d}$, the procedure* SparseFFT-RandomPhase(x, n, d, k) *in Algorithm 5 recovers $\widehat{x}$ with probability $1 - \frac{1}{N^2}$. Moreover, the sample complexity of this procedure is $O(k^2 \log^4 N)$ and its runtime is $O(k^2 \log^4 N)$.*

Impossibility of reducing the number of iterations (rounds of adaptivity): signals with low Hamming weight support. We note that our algorithm differs from all prior

works in that it uses many rounds of adaptivity. Indeed, the samples that our algorithm takes are guided by values of the signal that have been read in previously queried locations, which is in contrast to most prior Sparse Fourier Transform algorithms. Two notable exceptions in recent literature include the adaptive block Sparse FFT algorithms of [CKSZ17] and [CKPS16].

Our algorithm uses k rounds of adaptivity, peeling off one element at a time. It would be desirable to reduce the number of rounds of adaptivity by perhaps peeling off many elements in one batch as opposed to one at a time. For example, if the locations of the nonzeros of $\widehat{x}$ are uniformly random in $[n]^d$, then the splitting tree of x is likely to be rather balanced (see, e.g. Fig. 5 for an illustration), so perhaps one can find a filter G that has small support and can be efficiently used to isolate many coefficients at once? Indeed, this intuition turns out to be correct for signals with uniformly random supports—we show in Section 2.2 below (with details presented in Section 8) that this idea yields a $\widetilde{O}(k)$ time algorithm. However, rather surprisingly, adversarial instances exist that force the peeling process to use $k^{1-o(1)}$ rounds of adaptivity in the worst case, making our analysis essentially tight. We now present this adversarial instance.

Definition 8 (Hamming ball)**.** For any power of two integer n any integer $0 \le c \le \log_2 n$, we define H_c^n to be the *closed Hamming ball* of radius c centered at 0:

$$H_c^n = \{f \in [n] : w(f) \le c\},$$

where $w(f)$ is the Hamming weight of the binary representation of f, i.e., $w(f)$ is the number of ones in the binary representation of f.

Note that $|H_c^n| = \sum_{j=0}^{c} \binom{\log_2 n}{j}$.

Definition 9 (Class of signals with low Hamming support)**.** For any power of two integer n and any integer c, Let $\mathcal{X}_c^n$ denote the class of signals in $\mathbb{C}^n$ with support H_c^n as in Definition 8,

$$\mathcal{X}_c^n = \{x \in \mathbb{C}^n : \operatorname{supp} x \subseteq H_c^n\}$$

Note that for any $x \in \mathcal{X}_c^n$ we have that $\|x\|_0 = \sum_{i=0}^{c} \binom{\log_2 n}{i}$, so for any $c \le (\frac{1}{2} - \epsilon)\log_2 n$, the signals that are contained in class $\mathcal{X}_c^n$ are $\Theta\Big(\binom{\log_2 n}{c}\Big)$-sparse.

Definition 10 (Low Hamming weight binary trees)**.** Suppose n is a power of two integer. Then, we define a *low Hamming weight binary tree* T_c^n inductively for $c = 0, 1, \ldots, \log_2 n$:

1. T_0^n is defined to be the unique tree of depth $\log_2 n$ that has a single leaf node and satisfies the property that each non-leaf node has a single right child only. Thus, T_0^n has $\log_2 n + 1$ nodes.

2. For any $c > 0$, T_c^n is constructed as follows: Take T_0^n and label the nodes in order from the root to the leaf as $0, 1, \ldots, \log_2 n$. Then, for each node $0 \le j < \log_2 n$, take a copy of $T_{c-1}^{n/2^{j+1}}$ and let its root be the left child of node j. The resulting tree defines T_c^n.

Note that all the leaves of T_c^n are at level $\log_2 n$.

It is not hard to see that T_c^n is in fact the splitting tree for the set H_c^n and, hence, the number of its leaves is $\sum_{i=0}^{c} \binom{\log_2 n}{i}$. An illustration of the tree T_c^n for $c = 2$ and $n = 32$ is shown in Figure 4.

We prove the following theorem in Section 6 (see Theorem 9):

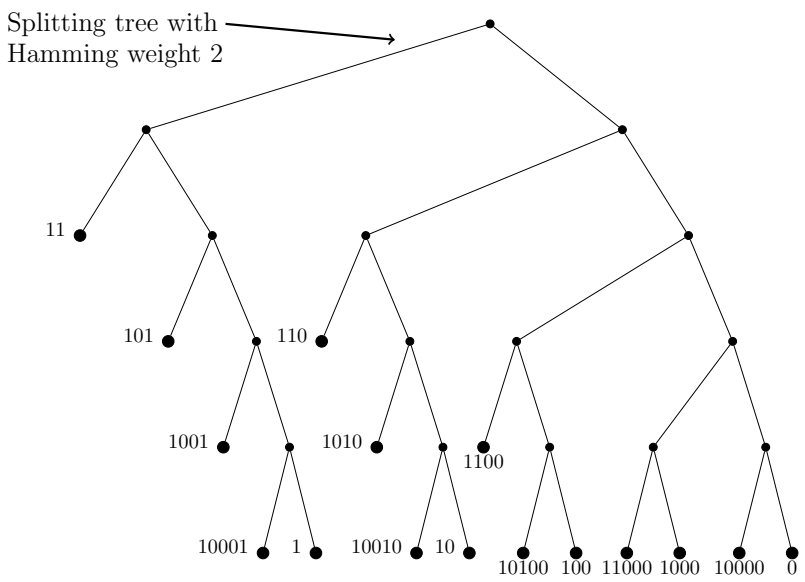


Figure 4: Illustration of T_2^n, the splitting tree corresponding to a family of signals with Hamming weight 2. For simplicity, we truncate terminal rightward paths from leaves to the bottom level of the tree from the picture. The corresponding support set of this tree is $S = \{0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 16, 17, 18, 20, 24\}$. Note that all the elements of S have binary representations with Hamming weight at most 2.

Theorem 5 (Informal version of Theorem 9)**.** *A peeling process with threshold $\tau \leq \log_2 k + O(1)$ (i.e. any threshold that allows isolation of an element at cost bounded by $O(k)$) must take $k^{1-o(1)}$ iterations to terminate.*

To add to the result above, we note that the lower bound on the number of rounds of adaptivity is not the only cause for quadratic runtime in our algorithm. The other cause is the necessity to update the residual signal as more and more elements are recovered, i.e. perform non-uniform Fourier transform computations. Since no subquadratic approach to this problem are known in high dimensions, it seems plausible that a $k^{2-\Omega(1)}$ runtime algorithm for high-dimensional FFT would also shed light on the complexity of this intriguing problem.

2.2 Runtime $\widetilde{O}(k)$ for random supports through a batched peeling process

To complement our lower bound of $k^{1-o(1)}$ rounds of adaptive pruning for *worst-case* signals using our adaptive aliasing filters, we show that if the support of the signal is uniformly random, adaptive aliasing filters can be used to achieve an algorithm with $\widetilde{O}(k)$ runtime. A beautiful $\widetilde{O}(k)$ runtime and optimal $O(k)$ sample complexity algorithm for this model was given in [GHI+13]. The algorithm was stated for $d = 2$ but readily extends to high dimensions. Unfortunately, it comes with a major restriction, namely, the sparsity k must be $o(N^{1/d})$. Our approach is different and extends to all $k \leq N$.

We now introduce the notion of a Fourier sparse *random support signal*:

Definition 11 (Random support signal)**.** For any positive integer d, power of two n, and arbitrary $\beta : [n]^d \to \mathbb{C}$, we define $x : [n]^d \to \mathbb{C}$ to be a *random support signal* of Fourier sparsity

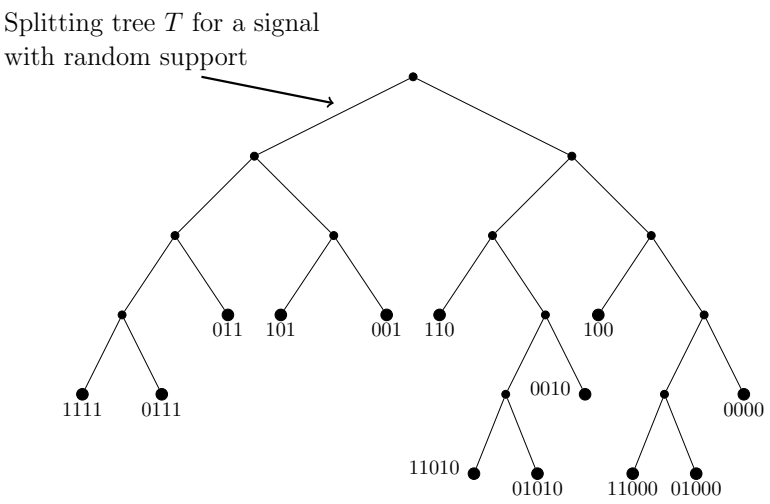


Figure 5: An example of a splitting tree for a signal with uniformly random support (the nodes are labelled in binary). For simplicity, we truncate terminal rightward paths from leaves to the bottom level of the tree from the picture. The corresponding support set of this tree is $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 15, 24, 26\}$

.

k (with values given by β) if $\widehat{x}$ is the signal defined by

$$\widehat{x}_{\boldsymbol{f}} = \begin{cases} \beta_{\boldsymbol{f}} & \text{with probability } k/n^d \\ 0 & \text{with probability } 1 - k/n^d \end{cases},$$

where the $x_{\boldsymbol{f}}$ are independently chosen for the various $\boldsymbol{f} \in [n]^d$.

In other words, we assume a Bernoulli model for supp $\widehat{x}$, while the values at the frequencies that are chosen to be in the support are arbitrary.

Our algorithmic result for such signals is stated below.

Theorem 6 (Sparse FFT algorithm for random support signals)**.** *Suppose d is a positive integer and n and k are powers of two. For any signal $x \in \mathbb{C}^{n^d}$ such that x is a random support signal of Fourier sparsity k, the procedure* SparseFFT *(x, n, d, k) (see Algorithm 8) returns $\widehat{x}$ with probability $9/10$. Moreover, the runtime and sample complexity of this procedure are $\widetilde{O}(k)$.*

The algorithm is motivated by the idea of speeding up our algorithm for worst-case signals (Algorithm 4, also see Theorem 3) by reducing the number of iterations of the process from $\Theta(k)$ down to $O(\log k)$. Such a reduction (which we show to be impossible for worst-case signals in Section 6) requires the ability to peel off many elements of the residual in a single phase of the algorithm, which turns out to be possible if the support of $\widehat{x}$ is chosen uniformly at random as in Definition 11. Indeed, if one considers the splitting tree T of a signal with uniformly random support (see Fig. 5 for an illustration), one sees that

(a) a large constant fraction of nodes $v \in T$ satisfy $w_T(v) \leq \log_2 k + O(1)$;

(b) the adaptive aliasing filters G constructed for such nodes will have significantly overlapping support in time domain.

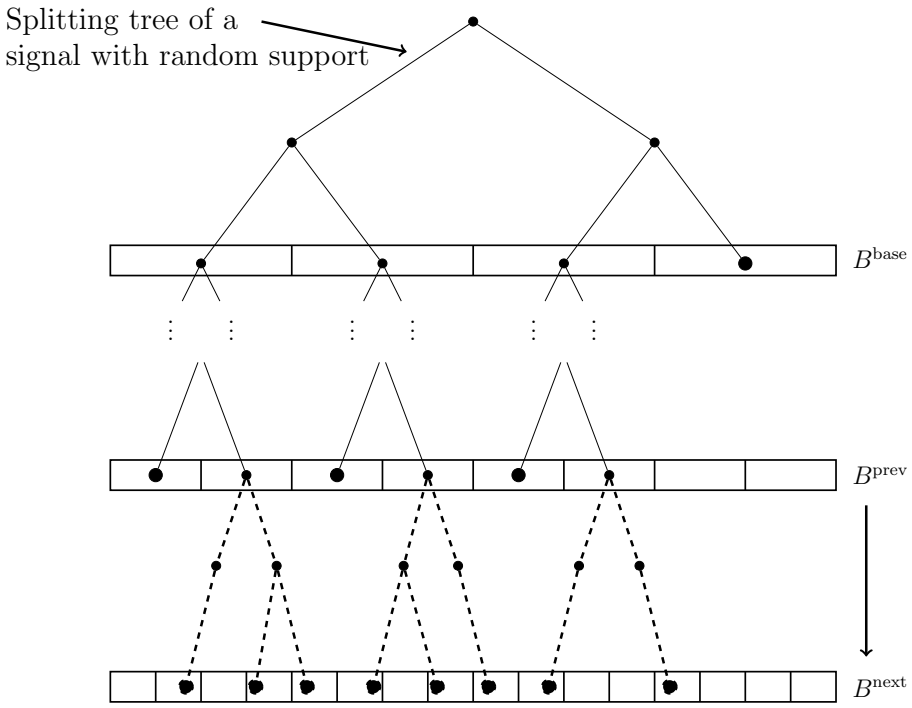


Figure 6: The illustration of the transition from B^{prev}-bucketing to B^{next}-bucketing on the splitting tree of a signal with random support.

We provide the intuition for this for the one-dimensional setting ($d = 1$) to simplify notation (changes required in higher dimensions are minor). In this setting, property **(b)** above is simply a manifestation of the fact that since the support is uniformly random, any given congruence classes modulo $B' = Ck$ for a large enough constant $C > 1$ is likely to contain only a single element of the support of $\widehat{x}$. Our adaptive aliasing filters provide a way to only partition frequency space along a carefully selected subset of bits in $[\log_2 N]$, but due to the randomness assumption, one can isolate most of the elements by simply partitioning using the bottom $\log_2 k + O(1)$ bits. This essentially corresponds to hashing $\widehat{x}$ into $B = Ck$ buckets at computational cost $O(B' \log B') = O(k \log k)$. While this scheme is efficient, it unfortunately only recovers a constant fraction of coefficients. One solution would be to hash into $B = Ck^2$ buckets (i.e., consider congruence classes modulo Ck^2), which would result in perfect hashing with good constant probability, allowing us to recover the entire signal in a single round. However, this hashing scheme would result in a runtime of $\Omega(k^2 \log k)$ and is, hence, not satisfactory. On the other hand, hashing into Ck^2 buckets is clearly wasteful, as most buckets would be empty. Our main algorithmic contribution is a way of "implicitly" hashing into Ck^2 buckets, i.e., getting access to the nonempty buckets, at an improved cost of $\widetilde{O}(k)$.

Our algorithm uses an iterative approach, and the main underlying observation is very simple. Suppose that we are given the ability to "implicitly" hash into B buckets for some B, namely, get access to the nonempty buckets. If B is at least $\min(Ck^2, N)$, we know that there are no collisions with high probability and we are done. If not, then we show that, given access to nonempty buckets in the B-hashing (i.e. a hashing into B buckets), we can get access to the nonempty buckets of a (ΓB)-hashing for some appropriately chosen constant $\Gamma > 1$ at a polylogarithmic cost in the size of each nonempty bucket of the B-bucketing by essentially

computing the Fourier transform of the signal restricted to nonempty buckets in the B-bucketing. We then proceed iteratively in this manner, starting with $B = Ck$, for which we can perform the hashing explicitly. Since the number of nonzero frequencies remaining in the residual after t iterations of this process decays geometrically in t, we can also afford to use a smaller number of buckets B' in the hashing that we construct explicitly, ensuring that the runtime is dominated by the first iteration.

Ultimately, the algorithm takes the following form. At every iteration, we explicitly compute a hashing into $B^{\text{base}} \leq Ck$ buckets explicitly. Then, using a list of nonempty buckets in a B^{prev}-hashing from the previous iteration, we extend this list to a list of nonempty buckets in a B^{next}-bucketing at polylogarithmic cost per bucket (by solving a well-conditioned linear system, see Algorithm 6), where $B^{\text{next}} = \Gamma \cdot B^{\text{prev}}$ for some large enough constant $\Gamma > 1$. Meanwhile, we reduce B^{base} by a factor of Γ, thus maintaining the invariant $B^{\text{base}} \cdot B^{\text{next}} \approx k^2$ at all times (note that this is satisfied at the start, when $B^{\text{base}} = B^{\text{prev}} \approx k$, and $B^{\text{base}} \cdot B^{\text{next}}$ remains invariant at each iteration). Therefore, after a logarithmic number of iterations, we have effectively emulated hashing into $\approx k^2$ but at a total cost of roughly one hashing computation into $\approx k$ buckets (see Figure 6 for an illustration).

Organization. In Section 3, we introduce basic definitions and notation that will be used throughout the paper. Section 4 introduces our main technical tool of adaptive aliasing filter, which are used in the various algorithms found in this paper. Section 5 shows how to use the adaptive aliasing filters to solve the problem of estimation for Fourier measurements for worst-case signals, i.e., problem (3), thereby proving Theorem 2. Section 6 then shows that the inherent tree pruning process used to subtract off recovered frequencies and access residual signals in the estimation algorithm is essentially optimal.

Section 7 proves our main theorem, Theorem 3, for problem (1) on worst-case signals. Additionally, it shows how to improve on the runtime under the assumption that the signal is a worst-case signal with random phase, thereby proving Theorem 4.

Finally, Section 8 discusses how to obtain an algorithm for problem (1) on random support signals and proves Theorem 6.

3 Preliminaries and notation

In this section, we introduce some notation and basic definitions that we will use in the paper.

For any positive integer n, we use the notation $[n]$ to denote the set of integer numbers $\{0, 1, \ldots, n-1\}$. We are interested in computing the Fourier transform of discrete signals of size N in dimension d, where $N = n^d$ for some $n \geq 2$. Such a signal will be a function $[n]^d \to \mathbb{C}$. However, we will often identify $[n]^d \to \mathbb{C}$ with $\mathbb{C}^{n^d}$ for convenience (and often use the two interchangably depending on the context). This correspondence is formally defined later in Definition 19. We first need the notion of an inner product.

Definition 12 (Inner product)**.** Let $\boldsymbol{t}$ and $\boldsymbol{f}$ be two vectors in dimension d. We denote the inner product of $\boldsymbol{t}$ and $\boldsymbol{f}$ by $\boldsymbol{f}^T\boldsymbol{t} = \sum_{q=1}^{d} f_q t_q$.

Let us define the *Fourier transform* of a signal.

Definition 13 (Fourier transform)**.** For any positive integers d and n, the *Fourier transform* of a signal $x \in \mathbb{C}^{n^d}$ is denoted by $\widehat{x}$, where for any $\boldsymbol{f} \in [n]^d$, we define $\widehat{x}_{\boldsymbol{f}} = \sum_{\boldsymbol{t} \in [n]^d} x_{\boldsymbol{t}} e^{-2\pi i \frac{\boldsymbol{f}^T \boldsymbol{t}}{n}}$.

Note that in the case of $n = 2$, the Fourier transform reduces to the Hadamard transform of size $N = 2^d$.

Claim 1 (Parseval's theorem)**.** *For any positive integers n and d, any signal $x \in \mathbb{C}^{n^d}$ satisfies $\|\widehat{x}\|_2^2 = n^d \cdot \|x\|_2^2$.*

Definition 14 (Unit impulse)**.** For any positive integers n and d, the *unit impulse function* $\delta \in \mathbb{C}^{n^d}$ is defined as the function given by $\delta(\boldsymbol{t}) = 1$ for $\boldsymbol{t} = 0$ and $\delta(\boldsymbol{t}) = 0$ for $\boldsymbol{t} \neq 0$.

Claim 2. *For any positive integers d, n, and any $\boldsymbol{a} \in [n]^d$, the inverse Fourier transform of $\widehat{x} : [n]^d \to \mathbb{C}$ given by $\widehat{x}_{\boldsymbol{f}} = e^{2\pi i \frac{\boldsymbol{a}^T \boldsymbol{f}}{n}}$ is $x_{\boldsymbol{t}} = \delta(\boldsymbol{t} + \boldsymbol{a})$.*

Claim 3 (Convolution theorem)**.** *Suppose d and n are positive integers. Then, for any signals $x, y \in \mathbb{C}^{n^d}$, $\widehat{(x * y)} = \widehat{x} \cdot \widehat{y}$, where $x * y$ is the convolution of x and y which itself is a signal in $\mathbb{C}^{n^d}$ defined as, $(x * y)_{\boldsymbol{t}} = \sum_{\boldsymbol{\tau} \in [n]^d} x_{\boldsymbol{\tau}} y_{\boldsymbol{t}-\boldsymbol{\tau}}$ for all $\boldsymbol{t} \in [n]^d$.*

We will require the notion of a *tensor product* of signals. Given d signals $G_1, G_2, \ldots, G_d : [n] \to \mathbb{C}$, the tensor product constructs a signal in $\mathbb{C}^{n^d}$ that is defined as follows.

Definition 15 (Tensor multiplication)**.** Suppose d and n are positive integers. Given functions $G_1, G_2, \ldots, G_d : [n] \to \mathbb{C}$, we define the *tensor product* $(G_1 \times G_2 \times \cdots \times G_d) : [n]^d \to \mathbb{C}$ as $(G_1 \times G_2 \times \cdots \times G_d)(\boldsymbol{j}) = G_1(j_1) \cdot G_2(j_2) \cdots G_d(j_d)$ for all $\boldsymbol{j} = (j_1, j_2, \ldots, j_d) \in [n]^d$.

Note that the tensor product is essentially a generalization of the usual outer product on two vectors to d vectors.

Claim 4 (Fourier transform of a tensor product)**.** *For any integers n and d and $G_1, G_2, \ldots, G_d \in \mathbb{C}^n$, let $G : [n]^d \to \mathbb{C}$ denote the tensor product $G = G_1 \times G_2 \times \cdots \times G_d$. Then, the d-dimensional Fourier transform $\widehat{G}$ of G is the tensor product of $\widehat{G}_1, \widehat{G}_2, \cdots, \widehat{G}_d$, i.e., $\widehat{G} = \widehat{G}_1 \times \widehat{G}_2 \times \cdots \times \widehat{G}_d$.*

Definition 16. For any positive d, n, and k, a signal $x : [n]^d \to \mathbb{C}$ is called *Fourier k-sparse* if $\|\widehat{x}\|_0 = k$.

Definition 17 (The Restricted isometry property)**.** We say that a matrix $A \in \mathbb{C}^{q \times n}$ satisfies the *restricted isometry property (RIP)* of *order* k if for every k-sparse vector $x \in \mathbb{C}^n$, i.e., $\|x\|_0 \leq k$, it holds that $\frac{1}{2}\|x\|_2^2 \leq \|Ax\|_2^2 \leq \frac{3}{2}\|x\|_2^2$.

We will use the following theorem from [HR17].

Theorem 7. (The Restricted Isometry Property [HR17, Theorem 3.7]) *For sufficiently large N and k, and a unitary matrix $M \in \mathbb{C}^{N \times N}$ satisfying $\|M\|_\infty = O\left(\frac{1}{\sqrt{N}}\right)$, the following holds. For some $q = O\left(k \log^2 k \log N\right)$ let $A \in \mathbb{C}^{q \times N}$ be a matrix whose q rows are chosen uniformly and independently from the rows of M, multiplied by $\sqrt{\frac{N}{q}}$. Then, with probability $1 - \frac{1}{N^{10}}$, the matrix A satisfies the restricted isometry property of order k, as per Definition 17.*

4 Adaptive aliasing filters

In this section, we introduce a new class of filters that forms the basis of our algorithm for estimation of worst case Fourier sparse signals. For simplicity, we begin by introducing the filters in the one-dimensional setting and then show how they naturally extend to the multidimensional setting (using tensoring). Throughout the section, we assume that the input is a signal $x \in \mathbb{C}^n$ with supp $\widehat{x} = S$ for some $S \subseteq [n]$.

4.1 One-dimensional Fourier transform

We restate the following definition for T_n^{full} and corresponding labels of vertices:

Definition 2. Suppose n is a power of two. Let T_n^{full} be a full binary tree of height $\log_2 n$, where for every $j \in \{0, 1, \ldots, \log_2 n\}$, the nodes at level j (i.e., at distance j from the root) are labeled with integers in $[2^j]$. For a node $v \in T_n^{\text{full}}$, we let f_v be its label. The label of the root is $f_{root} = 0$. The labelling of T_n^{full} satisfies the condition that for every $j \in [\log_2 n]$ and every v at level j, the right and left children of v have labels f_v and $f_v + 2^j$, respectively. Note that the root of T_n^{full} is at level 0, while the leaves are at level $\log_2 n$.

Next, we recall the definition of the *splitting tree* of a set.

Definition 3 (Splitting tree)**.** Let n be a power of two. For every $S \subseteq [n]$, the *splitting tree* $T = \text{Tree}(S, n)$ of a set S is a binary tree that is the subtree of T_n^{full} that contains, for every $j \in [\log_2 n]$, all nodes $v \in T_n^{\text{full}}$ at level j such that $\{f \in S : f \equiv f_v \pmod{2^j}\} \neq \emptyset$.

The splitting tree $T = \text{Tree}(S, n)$ can be constructed easily in $O(|S| \log n)$ time, given S. We provide simple pseudocode in Algorithm 9.

For every node $v \in T$, the *level* of v, denoted by $l_T(v)$, is the distance from v to the root. The following basic claim will be useful and follows immediately from the definition of $T = \text{Tree}(S, n)$:

Claim 5. *For every integer power of two n, if T is a subtree of T^{full}, then for every node $v \in T$, the labels of nodes that belong to the subtree T_v of T rooted at v are congruent to f_v modulo $2^{l_T(v)}$. Furthermore, every node $u \in T$ at level $l_T(v)$ or higher which satisfies $f_u \equiv f_v \pmod{2^{l_T(v)}}$ belongs to T_v.*

Definition 6 (Weight of a leaf)**.** Suppose n is a power of two. Let T be a subtree of T_n^{full}. Then for any leaf $v \in T$, we define its *weight* $w_T(v)$ *with respect to* T to be the number of ancestors of v in tree T with two children.

Definition 18 ((f,S)-isolating filter)**.** For every power of two n, set $S \subseteq [n]$, and $f \in S$, a filter $G \in \mathbb{C}^n$ is called (f,S)-*isolating* if $\widehat{G}_f = 1$, and $\widehat{G}_{f'} = 0$ for all $f' \in S \setminus \{f\}$.

In particular, if G is (f,S)-isolating, then for every signal $x \in \mathbb{C}^n$ with supp $\widehat{x} \subseteq S$, we have

$$\begin{aligned}\sum_{j\in[n]} x_j G_{t-j} &= (x * G)_t \\ &= \frac{1}{n}\sum_{f\in[n]} \widehat{x}_f \cdot \widehat{G}_f \cdot e^{2\pi i \frac{ft}{n}} \\ &= \frac{1}{n}\widehat{x}_f e^{2\pi i \frac{ft}{n}}\end{aligned}$$

for all $t \in [n]$, by convolution theorem, see Claim 3.

While the definitions above suffice to state our estimation primitive, our Sparse FFT algorithm requires a filter G that satisfies a more refined property due to the fact that throughout the execution of the algorithm, the identity of supp $\widehat{x}$ is only partially known. We encode this knowledge as a subtree T of T_n^{full} whose leaves are not necessarily at level $\log_2 n$. Hence, every leaf $v \in T$ corresponds to a set of frequencies in the support of $\widehat{x}$ whose full identities have not been discovered yet. This is captured by the following definition:

Definition 4 (Frequency cone of a leaf of T)**.** For every power of two n, subtree T of T_n^{full}, and vertex $v \in T$ which is at level $l_T(v)$ from the root, define the *frequency cone of v with respect to T* as

$$\text{FrequencyCone}_T(v) := \left\{ f \in [n] : f \equiv f_v \pmod{2^{l_T(v)}} \right\}.$$

Note that under this definition, the frequency cone of a vertex v of T corresponds to the subtree rooted at v when T is embedded inside T_n^{full} (see Figure 2).

Definition 5 ((v,T)-isolating filter)**.** For every integer n, subtree T of T_n^{full}, and leaf v of T, a filter $G \in \mathbb{C}^n$ is called (v,T)-*isolating* if the following conditions hold:

- For all $f \in \text{FrequencyCone}_T(v)$, we have $\widehat{G}_f = 1$.
- For every $f' \in \bigcup_{\substack{u \neq v \\ u\text{: leaf of } T}} \text{FrequencyCone}_T(u)$, we have $\widehat{G}_{f'} = 0$.

Note that in particular, for all signals $x \in \mathbb{C}^n$ with supp $\widehat{x} \subseteq \bigcup_{u\text{: leaf of } T} \text{FrequencyCone}_T(u)$ and $t \in [n]$,

$$\sum_{j\in[n]} x_j G_{t-j} = \frac{1}{n} \sum_{f \in \text{FrequencyCone}_T(v)} \widehat{x}_f e^{2\pi i \frac{ft}{n}}.$$

Lemma 3 (Filter properties)**.** *For every power of two n, subtree T of T_n^{full}, and leaf $v \in T$, the procedure* FilterPreProcess(T, v, n) *outputs a static data structure $\mathbf{g} \in \mathbb{C}^{\log_2 n}$ in time $O(\log_2 n)$ such that, given $\mathbf{g}$, the following conditions hold:*

1. *The primitive* FilterTime$(\mathbf{g}, n)$ *outputs a filter G such that $|\text{supp } G| = 2^{w_T(v)}$ and G is a (v,T)-isolating filter. Moreover, the procedure runs in time $O(2^{w_T(v)} + \log_2 n)$.*
2. *For every $\xi \in [n]$, the primitive* FilterFrequency$(\mathbf{g}, n, \xi)$ *computes the Fourier transform of G at frequency ξ, namely, $\widehat{G}(\xi)$, in time $O(\log_2 n)$.*

Algorithm 1 Filter construction in time and Fourier domain

procedure FILTERPREPROCESS(T, v, n)
  $r \leftarrow$ root of T, $l \leftarrow l_T(v)$, $f \leftarrow f_v$
  $v_0, v_1, \ldots, v_l \leftarrow$ path from r to v in T, where $v_0 = r$ and $v_l = v$
  $\mathbf{g} \leftarrow \{0\}^{\log_2 n}$
  for $j = 1$ `to` l **do**
    if v_{j-1} has two children in T **then**
      $g_j \leftarrow e^{-2\pi i \frac{f}{2^j}}$
  return g
procedure FILTERTIME($\mathbf{g}, n$)
  $G(t) \leftarrow \delta(t)$ for all $t \in [n]$
  for $l = 1$ `to` $\log_2 n$ **do**
    if $g_l \neq 0$ **then**
      $G(t) \leftarrow \frac{G(t)}{2} + g_l \cdot \frac{G(t+n/2^l)}{2}$ for all $t \in [n]$
  return G
procedure FILTERFREQUENCY($\mathbf{g}, n, \xi$)
  $\widehat{G}_\xi \leftarrow 1$
  for $l = 1$ `to` $\log_2 n$ **do**
    if $g_l \neq 0$ **then**
      $\widehat{G}_\xi \leftarrow \widehat{G}_\xi \cdot (1 + g_l \cdot e^{2\pi i \frac{\xi}{2^l}})/2$
  return $\widehat{G}_\xi$

Before we prove Lemma 3, we establish the following corollary, assuming the statement of Lemma 3 holds.

Corollary 1. *Suppose n is a power of two, $S \subseteq [n]$, and $f \in S$. Then, let $T = \text{Tree}(S, n)$ be the splitting tree of S. If v is the leaf of T with label $f_v = f$, while $\mathbf{g}$ is the output of* FILTERPREPROCESS(T, v, n)*, and G is the filter computed by* FILTERTIME$(\mathbf{g}, n)$*, then the following conditions hold:*

(1) *G is an (f, S)-isolating filter.*

(2) *$|\text{supp } G| = 2^{w_T(v)}$.*

Proof. Indeed, given a subset S and $f \in S$, if $T = \text{Tree}(S, n)$, then all the leaves of T are at level $\log_2 n$ and the set of labels of the leaves is exactly S. Hence, for every leaf v of T, one has $\text{FrequencyCone}_T(v) = \{f_v\}$. By Lemma 3, G is a (v, T)-isolating filter. Therefore, by Definition 5,

$$\emptyset = \text{supp } \widehat{G} \cap \left(\bigcup_{\substack{u \neq v \\ u\text{: leaf of } T}} \text{FrequencyCone}_T(u) \right) = \text{supp } \widehat{G} \cap \left(\bigcup_{\substack{u \neq v \\ u\text{: leaf of } T}} \{f_u\} \right) = \text{supp } \widehat{G} \cap (S \setminus f_v),$$

and $\widehat{G}(f) = 1$ for all $f \in \text{FrequencyCone}_T(v) = \{f_v\}$. This implies **(1)**, see definition of (f, S)-isolating filters in 18. Property **(2)** follows directly from Lemma 3. □

Now, we prove Lemma 3.

Proof of Lemma 3: Let v be a leaf of T, $l = l_T(v)$ denote the level of v (i.e., distance from the root), r denote the root of T, and $v_0, v_1, \ldots, v_l$ denote the path from root to v in T, where $v_0 = r$ and $v_l = v$.

We first show how to efficiently construct a (v, T)-isolating filter in the *Fourier* domain, i.e., how to efficiently construct $\widehat{G}$. Then we derive the time domain representation of G. We iteratively define a sequence of functions $G_0, G_1, \ldots, G_l$ (with Fourier transforms $\widehat{G}_0, \widehat{G}_1, \ldots, \widehat{G}_l$, respectively) by traversing the path from the root to v in T, after which we let G be the final filter constructed on this path, i.e., $G := G_l$ (and $\widehat{G} := \widehat{G}_l$). We start with $\widehat{G}_0(\xi) = 1$ for all $\xi \in [n]$. Then, we iteratively define $\widehat{G}_q$ in terms of $\widehat{G}_{q-1}$ according to the following update rule for all $q = 1, 2, \ldots, l$:

$$\widehat{G}_q(\xi) = \begin{cases} \widehat{G}_{q-1}(\xi) \cdot \frac{1+e^{2\pi i \frac{\xi - f_v}{2^q}}}{2} & \text{if } v_{q-1} \text{ has two children in } T \\ \widehat{G}_{q-1}(\xi) & \text{otherwise} \end{cases}. \tag{4}$$

for every $\xi \in [n]$.

We now show that $G = G_l$ is a (v, T)-isolating filter. It is enough to show that G satisfies

$$(\text{supp } \widehat{G}) \cap \left(\bigcup_{\substack{u \neq v \\ u\text{: leaf of } T}} \text{FrequencyCone}_T(u) \right) = \emptyset \tag{5}$$

and

$$\widehat{G}(f) = 1 \text{ for all } f \in \text{FrequencyCone}_T(v). \tag{6}$$

We now prove (5). Consider a leaf u of T distinct from v. Recall that $v_0, v_1, \ldots, v_l$ denotes the root to v path in T. Let j be the largest integer such that v_j is a common ancestor of v and u.

By definition of tree T (Definition 2) and because v_j is at level j, one has that the label of the right child a of v_j is f_{v_j}, and the label of the left child b is $f_{v_j} + 2^j$. Furthermore, using this together with Claim 5, we get that the labels of nodes in subtree T_a of T subtended at the right child a of v are congruent to $f_a = f_{v_j}$ modulo 2^{j+1}, and labels in the subtree T_b rooted at the left child b of v_j are all congruent to $f_b = f_{v_j} + 2^j$ modulo 2^{j+1}.

Suppose that v belongs to the right subtree of v_j, and u belongs to the left subtree (the other case is symmetric). We thus get that $f_v \equiv f_{v_j} \pmod{2^{j+1}}$, and $f_u \equiv f_{v_j} + 2^j \pmod{2^{j+1}}$. It now suffices to note that by construction of $\widehat{G}$ (see (4)), we have that for all $\xi \in [n]$,

$$\widehat{G}_{j+1}(\xi) = \widehat{G}_j(\xi) \cdot \frac{1 + e^{2\pi i \frac{\xi - f_v}{2^{j+1}}}}{2}.$$

By Claim 5, for all $f \in \text{FrequencyCone}_T(u)$ one has that $f \equiv f_u \pmod{2^{l_T(u)}}$ and hence, $f \equiv f_u \pmod{2^{j+1}}$ because $j + 1 \leq l_T(u)$. Therefore, by substituting $\xi = f$ in the above, we get

$$\widehat{G}_{j+1}(f) = \widehat{G}_j(f) \cdot \frac{1 + e^{2\pi i \frac{f - f_v}{2^{j+1}}}}{2} = \widehat{G}_j(f) \cdot \frac{1 + e^{2\pi i \frac{f_u - f_v}{2^{j+1}}}}{2} = 0,$$

implying that $\widehat{G}_{j+1}(f) = 0$ and, hence, $\widehat{G}_l(f) = 0$, as required.

It remains to prove (6). Consider any $f' \in \text{FrequencyCone}_T(v)$, and note that by Claim 5, $f' \equiv f_v \pmod{2^l}$. Using this in (4), we get

$$\begin{aligned}\widehat{G}(f') &= \prod_{\substack{q\in\{1,2,\ldots,l\}\\ v_{q-1} \text{ has two children in } T}} \frac{1+e^{2\pi i \frac{f'-f_v}{2^q}}}{2} \\ &= 1,\end{aligned}$$

since $f' - f_v \equiv 0 \pmod{2^q}$ for every $q = 0, \ldots, l$.

Next, note that the primitive FilterPreProcess(T, v, n) preprocesses the tree T by traversing the path from root to leaf v in time $O(\log_2 n)$. Given $\mathbf{g}$, the primitive FilterFrequency $(\mathbf{g}, n, \xi)$ implements (4) for successive values of q, and the runtime of this algorithm is $O(\log_2 n)$ because of the *for* loop passing through vector $\mathbf{g}$.

Finally, it remains to show that the filter G in *time domain* can be computed efficiently and has a small support. First note that by Claim 2, the inverse Fourier transform of $\frac{1+e^{2\pi i \frac{\xi-f_v}{2^q}}}{2}$ is $\frac{\delta(t)+e^{-2\pi i f_v/2^q}\delta\left(t+\frac{n}{2^q}\right)}{2}$.

By the duality of convolution in the time domain and multiplication in Fourier domain (see Claim 3), we can equivalently define G (see (4)) by letting $G_0(t) = \delta(t)$ and setting

$$G_q(\xi) = \begin{cases} G_{q-1}(t) * \frac{\delta(t)+e^{-2\pi i f_v/2^q}\delta\left(t+\frac{n}{2^q}\right)}{2} & \text{if } v_{q-1} \text{ has two children in } T \\ G_{q-1}(t) & \text{otherwise} \end{cases} \tag{7}$$

for every $q = 1, \ldots, l$. Thus, $G = G_l$ is the time domain representation of the filter $\widehat{G}$ defined in (4). We now note that convolving any function with a function supported on two points, e.g., $\frac{1}{2}\left(\delta(t) + e^{-2\pi i f_v/2^q}\delta(t+\frac{n}{2^q})\right)$, at most doubles the support. Since the number of times the convolution is performed in obtaining G_l from G_0 (as per (7)) is $w_T(v)$, the support size of G is at most $2^{w_T(v)}$. Given $\mathbf{g}$, the primitive FilterTime $(\mathbf{g}, n)$ implements the above algorithm for construction of G and, therefore, runs in time $O(2^{w_T(v)} + \log_2 n)$. □

4.2 d-dimensional Fourier transform

In this section, we show that our construction of adaptive aliasing filters from the previous section naturally extends to higher dimensions without any loss by tensoring.

Definition 19 (Flattening of $[n]^d$ to $[n^d]$. Unflattening of $[n^d]$ to $[n]^d$)**.** For every power of two n, positive integer d, and $\boldsymbol{f} = (f_1, \ldots, f_d) \in [n]^d$ we define the *flattening* of $\boldsymbol{f}$ as

$$\widetilde{\boldsymbol{f}} = \sum_{r=1}^{d} f_r \cdot n^{r-1}.$$

Similarly, for a subset $S \subseteq [n]^d$ we let $\widetilde{S} := \{\widetilde{\boldsymbol{f}} : \boldsymbol{f} \in S\}$ denote the flattening of S.

For $\widetilde{\boldsymbol{\xi}} \in [n^d]$, we define the *unflattening* of $\widetilde{\boldsymbol{\xi}}$ as $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_d) \in [n]^d$, where

$$\xi_q = \frac{\widetilde{\boldsymbol{\xi}} - \widetilde{\boldsymbol{\xi}} \pmod{n^{q-1}}}{n^{q-1}} \pmod{n}.$$

for every $q = 1, \ldots, d$. Similarly, for a subset $\widetilde{R} \subseteq [n^d]$, we let $R := \{\boldsymbol{\xi} \in [n]^d : \widetilde{\boldsymbol{\xi}} \in \widetilde{R}\}$ denote the unflattening of $\widetilde{R}$.

Definition 20 (Multidimensional splitting tree)**.** Suppose d is a positive integer and n is a power of two. For every $S \subseteq [n]^d$, the *flattened splitting tree* of S is defined as $\widetilde{T} = \mathrm{Tree}(\widetilde{S}, n^d)$ where $\widetilde{S}$ is flattening of S.

The unflattened splitting tree of S is denoted by T and is obtained from the flattened splitting tree $\widetilde{T}$ by unflattening the labels $\widetilde{\boldsymbol{f}}_v$ of all nodes $v \in \widetilde{T}$.

Definition 21 (Multidimensional $(\boldsymbol{f}, S)$-isolating filter)**.** Suppose n is a power of two integer and $S \subseteq [n]^d$ for a positive integer d. Then, for any frequency $\boldsymbol{f} \in S$, a filter $G : [n]^d \to \mathbb{C}$ is called $(\boldsymbol{f}, S)$-*isolating* if $\widehat{G}_{\boldsymbol{f}} = 1$ and $\widehat{G}_{\boldsymbol{f}'} = 0$ for every $\boldsymbol{f}' \in S \setminus \{\boldsymbol{f}\}$.

Definition 22 (Frequency cone of a leaf of T in high dimensions)**.** Suppose d is a positive integer, n is a power of two, and $N = n^d$. For every unflattened subtree T of T_N^{full} and $v \in T$, we define the *frequency cone of* v as

$$\mathrm{FrequencyCone}_T(v) := \left\{ \boldsymbol{f} \in [n]^d : \widetilde{\boldsymbol{f}} \equiv \widetilde{\boldsymbol{f}}_v \pmod{2^{l_T(v)}} \right\},$$

where $l_T(v)$ denotes the level of v in T (i.e., the distance from the root).

Claim 6. *For every positive integer d, power of two n, and every subtree T of $T_{n^d}^{\mathrm{full}}$ and every leaf $v \in T$ of height $l_T(v) < d \log_2 n$, let $T' = T \cup \{$left child u of $v\} \cup \{$right child w of $v\}$. Then the following holds,*

$$\mathrm{FrequencyCone}_T(v) = \mathrm{FrequencyCone}_{T'}(u) \cup \mathrm{FrequencyCone}_{T'}(w)$$

Definition 23 (Multidimensional (v, T)-isolating filter)**.** Suppose d is a positive integer, n is a power of two, and $N = n^d$. For every subtree T of T_N^{full} and vertex $v \in T$, a filter $G \in \mathbb{C}^{n^d}$ is called (v, T)-*isolating* if $\widehat{G}_{\boldsymbol{f}} = 1$ for all $\boldsymbol{f} \in \mathrm{FrequencyCone}_T(v)$ and for every $\boldsymbol{f}' \in \bigcup_{u:\text{ leaf of } T} \mathrm{FrequencyCone}_T(u)$ one has $\widehat{G}_{\boldsymbol{f}'} = 0$.

In particular, for every signal $x \in \mathbb{C}^{n^d}$ with $\operatorname{supp} \widehat{x} \subseteq \bigcup_{u:\text{ leaf of } T} \mathrm{FrequencyCone}_T(u)$ and for all $\boldsymbol{t} \in [n]^d$,

$$\sum_{\boldsymbol{j} \in [n]^d} x_{\boldsymbol{j}} G_{\boldsymbol{t}-\boldsymbol{j}} = \frac{1}{N} \sum_{\boldsymbol{f} \in \mathrm{FrequencyCone}_T(v)} \widehat{x}_{\boldsymbol{f}} e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{t}}{n}}.$$

Lemma 4 (Construction of a multidimensional isolating filter)**.** *Suppose n is a power of two integer and d is a positive integer. Let $N = n^d$. For every subtree T of T_N^{full} and every leaf $v \in T$, there exists a (v, T)-isolating filter G such that $|\operatorname{supp} G| = 2^{w_T(v)}$. Such a filter G can be constructed in time $O(2^{w_T(v)} + \log_2 N)$. Moreover, for any frequency $\boldsymbol{\xi} \in [n]^d$, the Fourier transform of G at frequency $\boldsymbol{\xi}$, i.e., $\widehat{G}(\boldsymbol{\xi})$, can be computed in time $O(\log_2 N)$.*

The proof of Lemma 4 appears in Appendix A. The key idea is to choose q^* to be the smallest positive integer such that $l_T(v) \le q^* \cdot \log_2 n$. One then defines successive filters $G^{(0)}, G^{(1)}, \ldots, G^{(q^*)}$ by letting $\widehat{G}^{(0)} = 1$ and

$$\widehat{G}^{(q)}(\boldsymbol{f}) = \widehat{G}^{(q-1)}(\boldsymbol{f}) \cdot \widehat{G}_q(f_q)$$

for $q = 1, 2, \ldots, q^*$, where $\widehat{G}_q$ is an isolating filter corresponding to the projection of the leaves of tree T into coordinate q. The final filter $G = G^{(q^*)}$ turns out to be (v, T)-isolating.

4.3 Putting it together

Claim 7. *For any binary tree T let L be the set of leaves of T. There exists a leaf $v \in L$ such that $w_T(v) \leq \log_2 |L|$.*

Proof. Let T' be the tree obtained by "collapsing" T, i.e., removing all nodes (and incident edges) of T that have exactly one child. Then, observe that the leaves of T are still preserved in T', except that they are at possibly varying levels. In particular, a leaf v in T' will be at level $w_T(v)$. Thus, by applying Kraft's inequality to T' (which is an equality because every node in T' is either a leaf or has two children), we see that

$$\sum_{v \in L} 2^{-w_T(v)} = 1.$$

Therefore, there exists a $v \in L$ such that $2^{-w_T(v)} \geq \frac{1}{|L|}$ and, therefore, $w_T(v) \leq \log_2 |L|$, as desired. □

This gives us the main result of this section, and the main technical lemma of the paper:

Corollary 2. *For every integer $n \geq 1$ a power of two and every positive integer d, every $S \subseteq [n]^d$, there exists an $\boldsymbol{f} \in S$ and an $(\boldsymbol{f}, S)$-isolating filter G (as defined in Definition 21) such that $|\text{supp } G| \leq |S|$.*

Proof. Follows by combining Lemma 4 with Claim 7. □

5 Estimation of sparse high-dimensional signals in quadratic time

In this section, we use the filters that we have constructed in Section 4 in order to show the first result of the paper, a deterministic algorithm for estimation of Fourier-sparse signals in time which is quadratic in the sparsity.

Theorem 8 (Estimation guarantee)**.** *Suppose n is a power of two integer and d is a positive integer and $S \subseteq [n]^d$. Then, for any signal $x \in \mathbb{C}^{n^d}$ with $\text{supp } \widehat{x} \subseteq S$, the procedure* ESTIMATE(x, S, n, d) *(see Algorithm 2) returns $\widehat{x}$. Moreover, the sample complexity of this procedure is $O(|S|^2)$ and its runtime is $O(|S|^2 \cdot d \log_2 n)$.*

Proof. The proof is by induction on the iteration number $t = 0, 1, 2, ...$ of the *while* loop in Algorithm 2. One can see that since at each iteration the tree T looses one of its leaves, the algorithm terminates after $|S|$ iterations, since initially the number of leaves of T is $|S|$. Let $\widehat{\chi}^{(t)}$ denote the signal $\widehat{\chi}$ after iteration t, and let $T^{(t)}$ denote the tree T after iteration t and let $S^{(t)}$ denote the set of frequencies corresponding to leaves of $T^{(t)}$, i.e., $S^{(t)} = \{\boldsymbol{f}_u : u \text{ is a leaf of } T^{(t)}\}$. In particular, $\widehat{\chi}^{(0)} = 0$ and $T^{(0)}$ is the unflattened spltting tree of S and $S^{(0)} = S$.

We claim that for each $t = 0, 1, \ldots, |S|$, we have

$$\text{supp } (\widehat{x} - \widehat{\chi}^{(t)}) \subseteq S^{(t)} \text{ and } |S^{(t)}| = |S| - t \tag{8}$$

Base case of induction: We have $S^{(0)} = S$ and $\widehat{\chi}^{(0)} \equiv 0$, which immediately implies (8) for $t = 0$.

Algorithm 2 d-dimensional Estimation for Sparse FFT with sample and time complexity k^2

1: **procedure** ESTIMATE(x, S, n, d)
2: $\widetilde{T} \leftarrow \text{TREE}(\widetilde{S}, n^d)$ ▷ $\widetilde{S}$: flattening of S ▷ $\widetilde{T}$: flattened splitting tree of S
3: Let T be the unflattening of $\widetilde{T}$
4: **while** $T \neq \emptyset$ **do**
5: $v \leftarrow \text{argmin}_{u:\text{ leaf of } T} w_T(u)$, $\boldsymbol{f} \leftarrow \boldsymbol{f}_v$ ▷ $\boldsymbol{f}$ is label of node v
6: $v_0, v_1, \ldots, v_{d\cdot\log_2 n} \leftarrow$ path from r to v in T, where $v_0 = r$ and $v_{d\cdot\log_2 n} = v$
7: **for** $q = 1$ `to` d **do**
8: $T_q^v \leftarrow$ subtree of T rooted at $v_{(q-1)\cdot\log_2 n}$
9: Remove all nodes of T_q^v which are at distance more than $\log_2 n$ from $v_{(q-1)\cdot\log_2 n}$
10: Label every node $u \in T_q^v$ as $f_u = (\boldsymbol{f}_u)_q$
11: $\mathbf{g} \leftarrow \text{FILTERPREPROCESS}(T_q^v, v_{q\cdot\log_2 n}, n)$
12: $G_q \leftarrow \text{FILTERTIME}(\mathbf{g}_q, n)$
13: $\widehat{G}_q(\xi_q) = \text{FILTERFREQUENCY}(\mathbf{g}_q, n, \xi_q)$
14: $G \leftarrow G_1 \times G_2 \times ... \times G_d$
15: $h_{\boldsymbol{f}} \leftarrow \sum_{\boldsymbol{\xi}\in[n]^d} \left(\widehat{\chi}_{\boldsymbol{\xi}} \cdot \prod_{q=1}^d \widehat{G}_q(\xi_q)\right)$
16: $\widehat{\chi}_{\boldsymbol{f}} \leftarrow \widehat{\chi}_{\boldsymbol{f}} + \left(n^d \cdot \sum_{\boldsymbol{j}\in[n]^d} x_{\boldsymbol{j}} \cdot G_{-\boldsymbol{j}}\right) - h_{\boldsymbol{f}}$
17: $T \leftarrow \text{Tree.REMOVE}(T, v)$
18: **return** $\widehat{\chi}$

Inductive step: For the inductive hypothesis, let $r \geq 1$ and assume that (8) holds for $t = r-1$. The main loop of the algorithm finds $v = \text{argmin}_{u:\text{ leaf of } T^{(r-1)}} w_{T^{(r-1)}}(u)$. By Claim 7 along with inductive hypothesis, $w_{T^{(r-1)}}(v) \leq \log_2 |S^{(r-1)}| \leq \log_2 |S|$. Note that the main loop of the algorithm constructs a $(\boldsymbol{f}_v, S^{(r-1)})$-isolating filter G, along with $\widehat{G}$. In order to do so, the algorithm constructs trees T_q^v for all $q \in \{1, ..., d\}$ which in total takes time $O(|S| d \log_2 n)$. Given T_q^v's, the algorithm constructs filter G and $\widehat{G}$ in time $O\left(2^{w_{T^{(r-1)}}(v)} + d\log_2 n\right) = O\left(|S| + d\log_2 n\right)$, by Lemma 4. Moreover, the filter G has support size $2^{w_{T^{(r-1)}}(v)} \leq |S|$ by Lemma 4.

By Lemma 4 computing the quantity $h_{\boldsymbol{f}} = \sum_{\boldsymbol{\xi}\in[n]^d} \widehat{\chi}_{\boldsymbol{\xi}}^{(r-1)} \cdot \widehat{G}(\boldsymbol{\xi})$ in line 15 of Algorithm 2 can be done in time $O(\|\widehat{\chi}^{(r-1)}\|_0 \cdot d\log_2 n) = O(|S| \cdot d\log_2 n)$. By convolution theorem 3, the quantity $h_{\boldsymbol{f}}$ satisfies $h_{\boldsymbol{f}} = n^d \cdot (\chi^{(r-1)} * G)_0$, and thus

$$\left(n^d \cdot \sum_{\boldsymbol{j}\in[n]^d} x_{\boldsymbol{j}} \cdot G_{-\boldsymbol{j}}\right) - h_{\boldsymbol{f}} = n^d \cdot \left(\left(x - \chi^{(r-1)}\right) * G\right)_0$$
$$= \widehat{x}_{\boldsymbol{f}_v} - \widehat{\chi}_{\boldsymbol{f}_v}^{(r-1)},$$

where the last transition is due to the fact that G is $\left(\boldsymbol{f}_v, S^{(r-1)}\right)$-isolating along with the inductive hypothesis of $\text{supp}\left(\widehat{x} - \widehat{\chi}^{(r-1)}\right) \subseteq S^{(r-1)}$.

We thus get that $\widehat{\chi}^{(r)}(\cdot) \leftarrow \widehat{\chi}^{(r-1)}(\cdot) + \left(\widehat{x} - \widehat{\chi}^{(r-1)}\right)_{\boldsymbol{f}_v} \cdot \delta_{\boldsymbol{f}_v}(\cdot)$. Moreover, it updates the tree $T^{(r)} \leftarrow \text{Tree.REMOVE}(T^{(r-1)}, v)$. Also note that the set $S^{(r)}$ gets updated to $S^{(r-1)} \setminus \{\boldsymbol{f}_v\}$ accordingly. This establishes (8) for $t = r$, thereby completing the inductive step.

The number of steps is exactly $|S|$, as follows from the inductive claim. Thus, the total runtime is $O(|S|^2 \cdot d\log_2 n)$. □

6 A lower bound of $k^{1-o(1)}$ rounds of tree pruning

One apparent disadvantage of our algorithm presented in the previous section is the fact that it only estimates elements of the Fourier spectrum one at a time, thereby taking k rounds to estimate all elements in the spectrum. Since the isolation of one element takes up to k time due to the support size of G, the resulting bound on the runtime is quadratic in k. A natural conjecture is that our analysis is not tight, and one can achieve better runtime by removing several nodes of weight at most $\log_2 k + O(1)$ at once. If one could argue that the filters G that isolate the nodes removed in one round have nontrivial overlap, runtime improvements could be achieved. In this section we present a class of signals on which $k^{1-o(1)}$ rounds of pruning the tree are required, showing that our analysis is essentially optimal.

Tree pruning process Suppose n is a power of two integer and τ is a positive integer. Let T be a subtree of T_n^{full}. The *tree pruning process,* $\mathcal{P}(T, \tau, n)$, is an iterative algorithm that performs the following operations on T successively until T is empty:

1. Find $\tilde{S}_\tau = \{\text{leaves } v \text{ of } T : w_T(v) \leq \tau\}$, i.e., set of vertices of weight no more than τ.
2. For each $v \in \tilde{S}_\tau$ (in an arbitrary order) remove v from T together with the path from v to its closest ancestor that has two children (i.e., run $T.remove(v)$; see Algorithm 9).

We show that for every k and sufficiently large integer n there exists a tree T with k leaves such that $\mathcal{P}(T, \tau, n)$ with $\tau = \log_2 k + O(1)$ requires $k^{1-o(1)}$ rounds to terminate. This in particular shows that our k^2 runtime analysis from section 5 cannot be improved by reusing work done in a single iteration, and hence our analysis is essentially optimal. Our construction is one-dimensional, although higher dimensional extensions can be readily obtained.

Theorem 9. *For any integer constant $c \geq 1$, sufficiently large power of two integer n there exists $k = \Theta(\log^c n)$ such that if $\tau = \log_2 k + O(1)$, the following condition holds. There exists a subtree T of T_n^{full} with k leaves such that the tree pruning process $\mathcal{P}(T, \tau, n)$ requires $k^{1-o(1)}$ iterations to terminate.*

The following simple lemma is crucial to our analysis

Lemma 5 (Monotonicity of tree pruning process)**.** *Suppose n is a power of two integer T' a subtree of T_n^{full} and T a subtree of T'. Then for every integer τ the number of rounds that it takes $\mathcal{P}(T, \tau, n)$ to collapse T is at most the number of rounds that it takes $\mathcal{P}(T', \tau, n)$ to collapse T'.*

Proof. For $j = 0, 1, 2, \ldots$, let $T^{(j)}$ (respectively $T'^{(j)}$) denote the tree obtained by performing j rounds of the tree pruning process (with threshold τ) to T (respectively T'). Note that $T^{(0)} = T$ and $T'^{(0)} = T'$.

We claim that $T^{(j)}$ is a subtree of $T'^{(j)}$ for all $j = 0, 1, \ldots$, which will obviously imply the desired conclusion. We use induction on j. Note that the **base** of induction is trivial for $j = 0$. Now, we prove the **inductive step**. Suppose $j > 0$. By the inductive hypothesis, we have that $T^{(j-1)}$ is a subtree of $T'^{(j-1)}$. Thus, for any leaf v that appears in both $T^{(j-1)}$ and $T'^{(j-1)}$, we have $w_T(v) \leq w_{T'}(v)$ (this is because any node in $T'^{(j-1)}$ along the path from the root to v that has exactly one child will also have exactly one child in $T^{(j-1)}$). Hence, if v is removed from $T'^{(j-1)}$ in the j-th iteration of the process, then it is also removed from $T^{(j-1)}$ during the j-th iteration. Hence, $T^{(j)}$ is a subtree of $T'^{(j)}$, which completes the inductive step and, therefore, proves the claim. □

We recall a few definitions.

Definition 8 (Hamming ball)**.** For any power of two integer n any integer $0 \leq c \leq \log_2 n$, we define H_c^n to be the *closed Hamming ball* of radius c centered at 0:

$$H_c^n = \{f \in [n] : w(f) \leq c\},$$

where $w(f)$ is the Hamming weight of the binary representation of f, i.e., $w(f)$ is the number of ones in the binary representation of f.

Note that $|H_c^n| = \sum_{j=0}^{c} \binom{\log_2 n}{j}$.

Definition 9 (Class of signals with low Hamming support)**.** For any power of two integer n and any integer c, Let $\mathcal{X}_c^n$ denote the class of signals in $\mathbb{C}^n$ with support H_c^n as in Definition 8,

$$\mathcal{X}_c^n = \{x \in \mathbb{C}^n : \text{supp } x \subseteq H_c^n\}$$

Note that for any $x \in \mathcal{X}_c^n$ we have that $\|x\|_0 = \sum_{i=0}^{c} \binom{\log_2 n}{i}$, so for any $c \leq (\frac{1}{2} - \epsilon) \log_2 n$, the signals that are contained in class $\mathcal{X}_c^n$ are $\Theta\Big(\binom{\log_2 n}{c}\Big)$-sparse.

Definition 10 (Low Hamming weight binary trees)**.** Suppose n is a power of two integer. Then, we define a *low Hamming weight binary tree* T_c^n inductively for $c = 0, 1, \ldots, \log_2 n$:

1. T_0^n is defined to be the unique tree of depth $\log_2 n$ that has a single leaf node and satisfies the property that each non-leaf node has a single right child only. Thus, T_0^n has $\log_2 n + 1$ nodes.

2. For any $c > 0$, T_c^n is constructed as follows: Take T_0^n and label the nodes in order from the root to the leaf as $0, 1, \ldots, \log_2 n$. Then, for each node $0 \leq j < \log_2 n$, take a copy of $T_{c-1}^{n/2^{j+1}}$ and let its root be the left child of node j. The resulting tree defines T_c^n.

Note that all the leaves of T_c^n are at level $\log_2 n$.

It is not hard to see that T_c^n is in fact the splitting tree for the set H_c^n and, hence, the number of its leaves is $\sum_{i=0}^{c} \binom{\log_2 n}{i}$.

Now, we are ready to prove Theorem 9.

Proof of Theorem 9: Let us choose the tree T to be T_c^n for some positive integer c. We will set parameter c at the end of the proof. Let $D(n, c, \tau)$ denote the number of iterations required to collapse T_c^n with threshold τ. We prove that

$$D(n, c, \tau) \geq \frac{\log_2^c n}{c! \cdot \tau^c} \tag{9}$$

for any power of two integer n, any integer $0 \leq c \leq \log_2 n$, and any positive integer τ. We use induction on c.

Base: Note that for $c = 0$, the tree T_c^n has one leaf, which gets removed in the first iteration of the tree pruning process. Thus, $D(n, 0, \tau) = 1$ for any power of two n and $\tau \geq 1$, and so, (9) holds for $c = 0$.

Inductive step: Suppose $c > 0$. For any T_c^n, we label the nodes along the path from the root to the rightmost leaf (i.e., the path formed by starting at the root and repeatedly following the right child) in order as $0, 1, \ldots, \log_2 n$.

Note that if $n \leq 2^\tau$, then

$$\frac{\log_2^c n}{c! \cdot \tau^c} \leq \frac{\tau^c}{c! \cdot \tau^c} \leq 1.$$

Thus, (9) does indeed hold for $n \leq 2^\tau$.

Now, suppose $n > 2^\tau$. Recall that a copy of $T_{c-1}^{n/2^{j+1}}$ is rooted at the left child of node j of T_c^n for all $j = 0, 1, \ldots, \tau - 1$. We divide the pruning process on T_c^n into two phases. The first phase consists of the process up until the point at which the left subtree of node j in T_c^n completely collapses for some $j \in \{0, 1, \ldots, \tau - 1\}$, while the second phases consists of the process thereafter. Thus, the number of rounds in the first phase is just the number of rounds till the top τ left subtrees collapses.

Note that during the first phase, the behavior of the collapsing process on the left subtree of node j corresponds to running a collapsing process with threshold $\tau - j - 1$ on $T_{c-1}^{n/2^{j+1}}$. Thus, the number of rounds in the first phase is,

$$R = \min_{0 \leq j < \tau} \{D(n/2^{j+1}, c-1, \tau - j - 1)\}.$$

By the inductive hypothesis (on c), we have that for $j = 0, 1, \ldots, \tau - 1$

$$D(n/2^{j+1}, c-1, \tau - j - 1) \geq \frac{1}{(c-1)!} \cdot \left(\frac{\log_2 n - j - 1}{\tau - j - 1} \right)^{c-1},$$

which implies that $R \geq \frac{1}{(c-1)!} \cdot \left(\frac{\log_2 n - 1}{\tau - 1} \right)^{c-1}$ since we assumed $\tau \leq \log_2 n$.

Now, let T' be the tree obtained after performing R rounds of the collapsing process on T_c^n. Moreover, let T'' be the tree obtained by further removing any left subtrees of nodes $0, 1, \ldots, \tau - 1$. By Lemma 5, we have that the number of rounds needed to collapse T' is at least the number of rounds needed to collapse T''. Moreover, observe that the number of rounds needed to collapse T'' is precisely $D(n/2^\tau, c, \tau)$, thus, the number of rounds in the second phase is at least $D(n/2^\tau, c, \tau)$, and so,

$$\begin{aligned} D(n, c, \tau) &\geq R + D(n/2^\tau, c, \tau) \\ &\geq \frac{1}{(c-1)!} \cdot \left(\frac{\log_2 n - 1}{\tau - 1} \right)^{c-1} + D(n/2^\tau, c, \tau). \end{aligned}$$

Note that a similar argument gives us

$$D(n/2^{a\tau}, c, \tau) \geq \frac{1}{(c-1)!} \cdot \left(\frac{\log_2 n - a\tau - 1}{\tau - 1} \right)^{c-1} + D(n/2^{(a+1)\tau}, c, \tau)$$

for all $a = 0, 1, \ldots, \lfloor (\log_2 n - 1)/\tau \rfloor - 1$ (this condition ensures that $\tau \leq \log_2(n/2^{a\tau})$, as required

by our argument above). Hence, it follows that

$$\begin{aligned}
D(n,c,\tau) &\geq \sum_{a=0}^{\lfloor(\log_2 n-1)/\tau\rfloor-1} \frac{1}{(c-1)!}\cdot\left(\frac{\log_2 n - a\tau - 1}{\tau-1}\right)^{c-1} + D(n/2^{\tau\cdot\lfloor(\log_2 n-1)/\tau\rfloor}, c, \tau)\\
&\geq \frac{1}{(c-1)!}\sum_{a=0}^{\lfloor(\log_2 n-1)/\tau\rfloor-1}\left(\frac{\log_2 n}{\tau} - a\right)^{c-1} + 1\\
&\geq \frac{1}{(c-1)!}\cdot\int_1^{\frac{\log_2 n}{\tau}} u^{c-1}\,du + 1\\
&= \frac{1}{(c-1)!}\cdot\frac{1}{c}\left(\left(\frac{\log_2 n}{\tau}\right)^c - 1\right) + 1\\
&\geq \frac{\log_2^c n}{c!\cdot\tau^c},
\end{aligned}$$

which establishes (9) for $n > 2^\tau$. This completes the inductive step.

Recall that $k = \Theta\left(\binom{\log_2 n}{c}\right)$, so for any constant c one has $k = \Theta(\binom{\log_2 n}{c}) \leq (e\log_2 n/c)^c$. Setting $\tau = \log_2 k + O(1)$, we get

$$D(n,c,\tau) \geq \frac{\log_2^c n}{c!\cdot\tau^c} = \Theta(k/(\log_2 k)^c) = k^{1-o(1)},$$

as required.

□

7 Sparse FFT for worst-case sparse signals and worst case signals with random phase

In this section we prove the main result of the paper, namely

Theorem 3 (Sparse FFT for worst-case signals)**.** *For any power of two integer n and any positive integer d and any signal $x \in \mathbb{C}^{n^d}$ with $\|\widehat{x}\|_0 = k$, the procedure* SparseFFT(x, n, d, k) *in Algorithm 4 recovers $\widehat{x}$. Moreover, the sample complexity of this procedure is $O(k^3\log^2 k\log^2 N)$ and its runtime is $O(k^3\log^2 k\log^2 N)$.*

We also study Fourier sparse signals x whose nonzero frequencies are distributed arbitrarily (worst-case) and whose values at the nonzero frequencies are independently chosen to have a uniformly random phase. Recall Definition 7:

Definition 7 (Worst-case signal with random phase)**.** For any positive integer d and power of two n, we define x to be a *worst-case signal with random phase* having values $\{\beta_{\boldsymbol{f}}\}_{\boldsymbol{f}\in[n]^d}$ if

$$\widehat{x}_{\boldsymbol{f}} = \beta_{\boldsymbol{f}} e^{2\pi i\theta} \quad \text{for uniformly random } \theta \in [0, 2\pi),$$

independently for every $\boldsymbol{f} \in [n]^d$. Furthermore, if k of the values $\{\beta_{\boldsymbol{f}}\}_{\boldsymbol{f}\in[n]^d}$ are nonzero, then x is said to be a *worst-case k-sparse signal with random phase* and is guaranteed to have sparsity k.

For this model we prove the stronger result:

Theorem 4 (Sparse FFT for worst-case signals with random support)**.** *For any power of two integer n, positive integer d, and worst-case k-sparse signal with random phase $x \in \mathbb{C}^{n^d}$, the procedure* SparseFFT-RandomPhase(x, n, d, k) *in Algorithm 5 recovers $\widehat{x}$ with probability $1 - \frac{1}{N^2}$. Moreover, the sample complexity of this procedure is $O(k^2 \log^4 N)$ and its runtime is $O(k^2 \log^4 N)$.*

The main property that allows us to obtain the stronger result is the fact that a small number of time domain samples from such a signal suffice to approximate its energy with high confidence (whereas $\Omega(k)$ samples are required in general for a worst-case k-sparse signal). This is reflected by the following

Lemma 6. *For any positive integer d, power of two n, and worst-case signal with random phase x, we have*

$$\Pr\left[\frac{1}{2} \cdot \frac{\|\beta\|_2^2}{n^{2d}} \leq \frac{1}{s} \sum_{j=1}^{s} |x_{\boldsymbol{t}_j}|^2 \leq \frac{3}{2} \cdot \frac{\|\beta\|_2^2}{n^{2d}}\right] \geq 1 - \frac{1}{n^{4d}},$$

where $s = Cd^3 \log_2^3 n$ for some absolute constant $C > 0$ and $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_s \sim \mathrm{Unif}([n]^d)$ are i.i.d. random variables. The probability is over the randomness in choosing the various $\boldsymbol{t}_j$ as well the randomness in the choice of phase for each frequency of $\widehat{x}$.

For completeness we present a proof for this lemma in Appendix A.

7.1 Proofs of Theorems 3 and 4

Given the construction of our adaptive aliasing filter from the previous section, our sparse recovery algorithms follow by a reduction to the estimation problem. We find the vertex $v^* = \mathrm{argmin}_{v \in T} w_T(v)$, which, by Kraft's inequality, satisfies $w_T(v^*) \leq \log_2 k$. We then define an auxiliary tree T' by appending a left a and a right child b to v. Then for each of the children a, b, we, in turn, construct a filter G that isolates them from the rest of T (i.e., from the frequency cones of other nodes in T) and check whether the corresponding restricted signals are nonzero. The latter is unfortunately a nontrivial task, since the sparsity of these signals can be as high as k, and detecting whether a k-sparse signal is nonzero requires $\Omega(k)$ samples. However, a fixed set of $k \log^3 N$ locations that satisfies the restricted isometry property (RIP) can be selected, and accessing the signal on those values suffices to test whether it is nonzero. If the signal is further assumed to be a worst case random phase signal, then a polylogarithmic number of samples suffices. The following lemma (Lemma 7) makes the latter claim formal. The algorithm is presented as Algorithm 4.

Lemma 7 (ZeroTest guarantee)**.** *Suppose d is a positive integer and n is a power of two. Assume T is a subtree of $T^{\mathrm{full}}_{n^d}$. Suppose that signals $x, \widehat{\chi} \in \mathbb{C}^{n^d}$ satisfy* supp $(\widehat{x} - \widehat{\chi}) \subseteq \bigcup_{u:\ \text{leaf of } T} \mathrm{FrequencyCone}_T(u)$*. Suppose that $\boldsymbol{\Delta}$ is a multiset of sample from $[n]^d$ which satisfies the following for every leaf v of T:*

$$\frac{1}{2} \cdot \frac{\|\widehat{y}\|_2^2}{n^{2d}} \leq \frac{1}{|\boldsymbol{\Delta}|} \cdot \sum_{\Delta \in \boldsymbol{\Delta}} |y_\Delta|^2 \leq \frac{3}{2} \cdot \frac{\|\widehat{y}\|_2^2}{n^{2d}}$$

where $y = (\widehat{x} - \widehat{\chi})_{\mathrm{FrequencyCone}_T(v)}$ is the signal obtained by restricting $\widehat{x} - \widehat{\chi}$ to frequencies $\boldsymbol{\xi} \in \mathrm{FrequencyCone}_T(v)$ and zeroing it out on all other frequencies.

Then the following conditions hold:

- ZeroTest$(x, \widehat{\chi}, T, v, n, d, \boldsymbol{\Delta})$ *outputs* **true** *if* supp $(\widehat{x} - \widehat{\chi}) \cap \text{FrequencyCone}_T(v) \neq \emptyset$*; otherwise, it outputs* **false***.*
- *The sample complexity of this procedure is* $O(2^{w_T(v)} \cdot |\boldsymbol{\Delta}|)$*, where* $w_T(v)$ *is the weight of leaf* v *in* T *(see Definition 6).*
- *The runtime of the* ZeroTest *procedure is*

$$O\left(\|\widehat{\chi}\|_0 \cdot |\boldsymbol{\Delta}| + |T| \cdot d \log_2 n + 2^{w_T(v)} \cdot |\boldsymbol{\Delta}|\right),$$

where $|T|$ *denotes the number of leaves of* T*.*

Proof. Consider lines 14-15 in Algorithm 3. By Claim 3, we have that

$$\begin{aligned} h_{\boldsymbol{f}}^{\Delta} &= \frac{1}{n^d} \sum_{\boldsymbol{\xi} \in [n]^d} e^{2\pi i \frac{\boldsymbol{\xi}^T \Delta}{n}} \cdot \widehat{\chi}_{\boldsymbol{\xi}} \widehat{G}_{\boldsymbol{\xi}} \\ &= \sum_{\boldsymbol{j} \in [n]^d} G_{\Delta - \boldsymbol{j}} \cdot \chi_{\boldsymbol{j}}. \end{aligned}$$

Thus,

$$\begin{aligned} H_{\boldsymbol{f}}^{\Delta} &= \left(\sum_{\boldsymbol{j} \in [n]^d} G_{\Delta - \boldsymbol{j}} \cdot x_{\boldsymbol{j}} \right) - h_{\boldsymbol{f}}^{\Delta} \\ &= \sum_{\boldsymbol{j} \in [n]^d} G_{\Delta - \boldsymbol{j}} \cdot (x - \chi)_{\boldsymbol{j}}. \end{aligned}$$

Note that, by Lemma 4, the filter G used in Algorithm 3 is a (v, T)-isolating filter. Therefore, by the assumption supp $(\widehat{x} - \widehat{\chi}) \subseteq \bigcup_{u:\text{ leaf of } T} \text{FrequencyCone}_T(u)$ and the definition of a (v, T)-isolating filter (see Definition 23), we have

$$\begin{aligned} H_{\boldsymbol{f}}^{\Delta} &= \sum_{\boldsymbol{j} \in [n]^d} G_{\Delta - \boldsymbol{j}} \cdot (x - \chi)_{\boldsymbol{j}} \\ &= \frac{1}{n^d} \sum_{\boldsymbol{\xi} \in \text{FrequencyCone}_T(v)} (\widehat{x} - \widehat{\chi})_{\boldsymbol{\xi}} \cdot e^{2\pi i \frac{\boldsymbol{\xi}^T \Delta}{n}}. \end{aligned}$$

Note that $H_{\boldsymbol{f}}^{\Delta}$ is essentially the inverse Fourier transform of $(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_T(v)}$, where $(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_T(v)}$ denotes the signal obtained by restricting $\widehat{x} - \widehat{\chi}$ to frequencies $\boldsymbol{\xi} \in \text{FrequencyCone}_T(v)$ and zeroing out the signal on all other frequencies. By the assumption of the lemma we have the following:

$$\frac{1}{2} \cdot \frac{\|(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_T(v)}\|_2^2}{n^{2d}} \leq \frac{1}{|\boldsymbol{\Delta}|} \cdot \sum_{\Delta \in \boldsymbol{\Delta}} |H_{\boldsymbol{f}}^{\Delta}|^2 \leq \frac{3}{2} \cdot \frac{\|(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_T(v)}\|_2^2}{n^{2d}}.$$

Therefore the first claim of the lemma holds.

Note that in order to construct a (v, T)-isolating filter G, along with $\widehat{G}$, the algorithm constructs trees T_q^v for all $q \in \{1, ..., d\}$, which has total time complexity $O(|T| d \log_2 n)$. Given T_q^v's,

the algorithm constructs filter G and $\widehat{G}$ in time $O\left(2^{w_T(v)} + d\log_2 n\right)$, by Lemma 4. Moreover, the filter G has support size $2^{w_T(v)}$, by Lemma 4.

By Lemma 4, computing the quantities $h_{\boldsymbol{f}}^{\Delta} = \frac{1}{n^d}\sum_{\boldsymbol{\xi}\in[n]^d} e^{2\pi i \frac{\boldsymbol{\xi}^T\Delta}{n}} \cdot \widehat{\chi}_{\boldsymbol{\xi}}\widehat{G}_{\boldsymbol{\xi}}$ for all Δ in line 14 of Algorithm 3 can be done in time $O\left(\|\widehat{\chi}\|_0 \cdot (|\boldsymbol{\Delta}| + d\log_2 n)\right) = O\left(\|\widehat{\chi}\|_0 \cdot |\boldsymbol{\Delta}|\right)$. Given the values of $h_{\boldsymbol{f}}^{\Delta}$ for various Δ, computing all $\left\{|H_{f^*}^{\Delta}|^2\right\}_{\Delta\in\boldsymbol{\Delta}}$ in line 15 takes time $O\left(2^{w_T(v)} \cdot |\boldsymbol{\Delta}|\right)$. Therefore the total runtime of this procedure is

$$O\left(|T| d\log_2 n + 2^{w_T(v)} \cdot |\boldsymbol{\Delta}| + \|\widehat{\chi}\|_0 \cdot |\boldsymbol{\Delta}|\right),$$

as desired.

Because support size of G is $2^{w_T(v)}$, computing all $\left\{|H_{f^*}^{\Delta}|^2\right\}_{\Delta\in\boldsymbol{\Delta}}$ in line 15 of the algorithm requires $O(2^{w_T(v)} \cdot |\boldsymbol{\Delta}|)$ samples from x which proves the second claim of the lemma. □

Algorithm 3 Procedure for testing zero hypothesis

1: **procedure** ZEROTEST($x, \widehat{\chi}, T, v, n, d, \boldsymbol{\Delta}$) ▷ $\boldsymbol{\Delta}$: multiset of elements from $[n]^d$
2: $\boldsymbol{f} \leftarrow \boldsymbol{f}_v$, $l \leftarrow l_T(v)$, $q^* \leftarrow \left\lceil \frac{l}{\log_2 n} \right\rceil$
3: $v_0, v_1, \ldots, v_l \leftarrow$ path from r to v in T, where $v_0 = r$ and $v_l = v$
4: $(u_1, u_2, \cdots, u_{q^*-1}, u_{q^*}) \leftarrow (v_{\log_2 n}, v_{2\log_2 n}, \cdots, v_{(q^*-1)\cdot\log_2 n}, v_l)$
5: **for** $q = 1$ `to` q^* **do**
6: $T_q^v \leftarrow$ subtree of T rooted at $v_{(q-1)\cdot\log_2 n}$
7: Remove all nodes of T_q^v which are at distance more than $\log_2 n$ from $v_{(q-1)\cdot\log_2 n}$
8: Label every node $w \in T_q^v$ as $f_w = (\boldsymbol{f}_w)_q$
9: $\mathbf{g}_q \leftarrow$ FILTERPREPROCESS(T_q^v, u_q, n)
10: $G_q \leftarrow$ FILTERTIME($\mathbf{g}_q, n$)
11: $\widehat{G}_q(\xi_q) =$ FILTERFREQUENCY($\mathbf{g}_q, n, \xi_q$)
12: $G \leftarrow G_1 \times G_2 \times ... \times G_d$
13: $h_{\boldsymbol{f}}^{\Delta} \leftarrow \frac{1}{n^d}\sum_{\boldsymbol{\xi}\in[n]^d}\left(e^{2\pi i\frac{\boldsymbol{\xi}^T\Delta}{n}} \cdot \widehat{\chi}_{\boldsymbol{\xi}} \cdot \prod_{q=1}^{d}\widehat{G}_q(\xi_q)\right)$ for all $\Delta \in \boldsymbol{\Delta}$
14: $H_{\boldsymbol{f}}^{\Delta} \leftarrow \left(\sum_{\boldsymbol{j}\in[n]^d} G(\Delta - \boldsymbol{j}) \cdot x_{\boldsymbol{j}}\right) - h_{\boldsymbol{f}}^{\Delta}$ for all $\Delta \in \boldsymbol{\Delta}$
15: **if** $\frac{1}{|\boldsymbol{\Delta}|}\sum_{\Delta\in\boldsymbol{\Delta}} |H_{\boldsymbol{f}}^{\Delta}|^2 = 0$ **then**
16: **return false**
17: **else**
18: **return true**

We now prove our main result:

Proof of Theorems 3 and 4: Note that Algorithms 4 and 5 are identical except in line 2. We first analyze the common code of the algorithms (after line 2) under the assumption that the set $\boldsymbol{\Delta}$ in all calls to ZEROTEST are replaced with a more powerful set which satisfies the precondition of Lemma 7 hence ZEROTEST correctly tests the zero hypothesis on its input signal with probability 1. We then establish a coupling between this idealized execution and the actual execution for both Algorithms 4 and 5, leading to our result.

Let m denote the size of the set $m = |\boldsymbol{\Delta}|$. We prove that the following properties are maintained throughout the execution of SPARSEFFT (Algorithm 4) and SPARSEFFT-RANDOMPHASE (Algorithm 5):

(1) supp $(\widehat{x} - \widehat{\chi}) \subseteq \bigcup_{u:\text{ leaf of } T} \text{FrequencyCone}_T(u)$;

Algorithm 4 Sparse FFT for worst-case sparse signals

procedure SPARSEFFT(x, n, d, k)
  $\boldsymbol{\Delta} \leftarrow$ Multiset of $[n]^d$ corresponding to Fourier measurements satisfying RIP of order k ▷ $|\boldsymbol{\Delta}| = O(k \log_2^2 k \cdot d \log_2 n)$, see Theorem 7
  $T \leftarrow \{r\}, \boldsymbol{f}_r \leftarrow 0$
  while $T \neq \emptyset$ **do**
    $v \leftarrow \text{argmin}_{u:\text{ leaf of } T} w_T(u)$, $\boldsymbol{f} \leftarrow \boldsymbol{f}_v$ and $l \leftarrow l_T(v)$
    if $l = d \log_2 n$ **then** ▷ All bits of v have been discovered
      $v_0, v_1, \ldots, v_{d\cdot\log_2 n} \leftarrow$ path from r to v in T, where $v_0 = r$ and $v_{d\cdot\log_2 n} = v$
      for $q = 1$ `to` d **do**
        $T_q^v \leftarrow$ subtree of T rooted at $v_{(q-1)\cdot\log_2 n}$
        Remove all nodes of T_q^v which are at distance more than $\log_2 n$ from $v_{(q-1)\cdot\log_2 n}$
        Label every node $u \in T_q^v$ as $f_u = (\boldsymbol{f}_u)_q$
        $\mathbf{g}_q \leftarrow$ FILTERPREPROCESS$(T_q^v, v_{q\cdot\log_2 n}, n)$
        $G_q \leftarrow$ FILTERTIME$(\mathbf{g}_q, n)$
        $\widehat{G}_q(\xi_q) =$ FILTERFREQUENCY$(\mathbf{g}_q, n, \xi_q)$
      $G \leftarrow G_1 \times G_2 \times ... \times G_d$
      $h_{\boldsymbol{f}} \leftarrow \sum_{\boldsymbol{\xi} \in [n]^d} \left(\widehat{\chi}_{\boldsymbol{\xi}} \cdot \prod_{q=1}^d \widehat{G}_q(\xi_q)\right)$
      $\widehat{\chi}_{\boldsymbol{f}} \leftarrow \widehat{\chi}_{\boldsymbol{f}} + \left(n^d \cdot \sum_{\boldsymbol{j} \in [n]^d} x_{\boldsymbol{j}} \cdot G_{-\boldsymbol{j}}\right) - h_{\boldsymbol{f}}$
      $T \leftarrow$ Tree.REMOVE(T, v)
    else
      $T' \leftarrow T \cup \{\text{left child } u \text{ of } v\} \cup \{\text{right child } w \text{ of } v\}$
      if ZEROTEST$(x, \widehat{\chi}, T', w, n, d, \boldsymbol{\Delta})$ **then**
        Add w as the right child of node v to tree T
        $\boldsymbol{f}_w \leftarrow \boldsymbol{f}$ ▷ Frequency corresponding to node w
      if ZEROTEST$(x, \widehat{\chi}, T', u, n, d, \boldsymbol{\Delta})$ **then**
        Add u as the left child of node v to tree T
        $\boldsymbol{f}_u \leftarrow \boldsymbol{f} + 2^l$; ▷ Frequency corresponding to node u
  return $\widehat{\chi}$;

Algorithm 5 Sparse FFT for worst-case sparse signals with random phase

procedure SPARSEFFT-RANDOMPHASE(x, n, d, k)
  $\boldsymbol{\Delta} \leftarrow \left\{\Delta_i : \Delta_i \sim \mathrm{Unif}([n]^d), \forall i \in [Cd^3 \log_2^3 n]\right\}$ ▷ C: constant ▷ $\boldsymbol{\Delta}$: Multiset
  $T \leftarrow \{r\}, \boldsymbol{f}_r \leftarrow 0$
  while $T \neq \emptyset$ **do**
    $v \leftarrow \mathrm{argmin}_{u:\text{ leaf of } T} w_T(u)$, $\boldsymbol{f} \leftarrow \boldsymbol{f}_v$ and $l \leftarrow l_T(v)$
    if $l = d \log_2 n$ **then** ▷ All bits of v have been discovered
      $v_0, v_1, \ldots, v_{d\cdot\log_2 n} \leftarrow$ path from r to v in T, where $v_0 = r$ and $v_{d\cdot\log_2 n} = v$
      for $q = 1$ `to` d **do**
        $T_q^v \leftarrow$ subtree of T rooted at $v_{(q-1)\cdot\log_2 n}$
        Remove all nodes of T_q^v which are at distance more than $log_2 n$ from $v_{(q-1)\cdot\log_2 n}$
        Label every node $u \in T_q^v$ as $f_u = (\boldsymbol{f}_u)_q$
        $\mathbf{g}_q \leftarrow$ FILTERPREPROCESS$(T_q^v, v_{q\cdot\log_2 n}, n)$
        $G_q \leftarrow$ FILTERTIME$(\mathbf{g}_q, n)$
        $\widehat{G}_q(\xi_q) =$ FILTERFREQUENCY$(\mathbf{g}_q, n, \xi_q)$
      $G \leftarrow G_1 \times G_2 \times ... \times G_d$
      $h_{\boldsymbol{f}} \leftarrow \sum_{\boldsymbol{\xi}\in[n]^d} \left(\widehat{\chi}_{\boldsymbol{\xi}} \cdot \prod_{q=1}^d \widehat{G}_q(\xi_q)\right)$
      $\widehat{\chi}_{\boldsymbol{f}} \leftarrow \widehat{\chi}_{\boldsymbol{f}} + \left(n^d \cdot \sum_{\boldsymbol{j}\in[n]^d} x_{\boldsymbol{j}} \cdot G_{-\boldsymbol{j}}\right) - h_{\boldsymbol{f}}$
      $T \leftarrow$ Tree.REMOVE(T, v)
    else
      $T' \leftarrow T \cup \{\text{left child } u \text{ of } v\} \cup \{\text{right child } w \text{ of } v\}$
      if ZEROTEST$(x, \widehat{\chi}, T', w, n, d, \boldsymbol{\Delta})$ **then**
        Add w as the right child of node v to tree T
        $\boldsymbol{f}_w \leftarrow \boldsymbol{f}$ ▷ Frequency corresponding to node w
      if ZEROTEST$(x, \widehat{\chi}, T', u, n, d, \boldsymbol{\Delta})$ **then**
        Add u as the left child of node v to tree T
        $\boldsymbol{f}_u \leftarrow \boldsymbol{f} + 2^l$; ▷ Frequency corresponding to node u
  return $\widehat{\chi}$;

(2) For every leaf u of tree T one has supp $(\widehat{x}-\widehat{\chi}) \cap \text{FrequencyCone}_T(u) \neq \emptyset$;

(3) If $\widehat{x}$ is a worst-case signal with random phase, then $\widehat{x}-\widehat{\chi}$ is a worst-case signal with random phase;

(4) The quantity $\phi = (d\log_2 n+1)\|\widehat{x}-\widehat{\chi}\|_0 - \sum_{u:\text{ leaf of } T} l_T(u)$ always decreases by at least 1 on every iteration of Algorithm 4 or 5;

(5) Always $\|\widehat{x}-\widehat{\chi}\|_0 \leq k$;

The **base** of the induction is provided by the first iteration, at which point T is a single vertex $T=\{r\}$ where r is the root with $\boldsymbol{f}_r = 0$ and $\widehat{\chi}=0$. The conditions **(1)** and **(2)** and **(3)** and **(5)** are satisfied since $\text{FrequencyCone}_T(r) = [n]^d$ and supp $(\widehat{x}-\widehat{\chi}) = \text{supp}\,\widehat{x} \neq \emptyset$ and $\widehat{x}-\widehat{\chi} = \widehat{x}$ is a worst-case signal with random phase if $\widehat{x}$ is a worst-case signal with random phase.

We now prove the **inductive step**. We assume that conditions **(1)** and **(2)** and **(3)** and **(5)** of the inductive hypothesis are satisfied at the beginning of a certain iteration and argue that conditions **(1)** and **(2)** and **(3)** and **(5)** are maintained at the end of the iteration. We also show that the value of the quantity ϕ defined in **(4)**, at the end of the loop is smaller than its value at the start of the loop by at least one. Let $v \in T$ be the smallest weight leaf chosen by the algorithm in line 4. We now consider two cases.

Case 1: $l_T(v) = d\log_2 n$**.** Since G is a (v,T)-isolating filter, we have by Definition 5 that for every signal $z \in \mathbb{C}^{n^d}$ with Fourier support supp $\widehat{z} \subseteq \bigcup_{u:\text{ leaf in } T} \text{FrequencyCone}_T(u)$ and for all $t \in [n]^d$,

$$\sum_{\boldsymbol{j}\in[n]^d} z_{\boldsymbol{j}} G_{\boldsymbol{t}-\boldsymbol{j}} = \frac{1}{n^d} \sum_{\boldsymbol{f}'\in \text{FrequencyCone}_T(v)} \widehat{z}_{\boldsymbol{f}'} e^{2\pi i \frac{\boldsymbol{f}'^T \boldsymbol{t}}{n}}. \tag{10}$$

By condition **(1)** of the inductive hypothesis one has supp $(\widehat{x}-\widehat{\chi}) \subseteq \bigcup_{u:\text{ leaf of } T} \text{FrequencyCone}_T(u)$, and thus we can apply (10) with $z = x-\chi$ and $\boldsymbol{t}=0$, obtaining

$$\sum_{\boldsymbol{j}\in[n]^d} (x-\chi)_{\boldsymbol{j}} G_{-\boldsymbol{j}} = \frac{1}{n^d} \sum_{\boldsymbol{f}'\in \text{FrequencyCone}_T(v)} \widehat{(x-\chi)}_{\boldsymbol{f}'}. \tag{11}$$

Note that by Claim 3,

$$n^d \cdot \sum_{\boldsymbol{j}\in[n]^d} \chi_{\boldsymbol{j}} G_{-\boldsymbol{j}} = \sum_{f\in[n]} \widehat{\chi}_f \widehat{G}_f = h_{\boldsymbol{f}},$$

where $h_{\boldsymbol{f}}$ is the quantity computed in line 15. We thus get that

$$\begin{aligned} n^d \sum_{\boldsymbol{j}\in[n]^d} x_{\boldsymbol{j}} \cdot G_{-\boldsymbol{j}} - h_{\boldsymbol{f}} &= \sum_{\boldsymbol{f}'\in \text{FrequencyCone}_T(v)} \widehat{(x-\chi)}_{\boldsymbol{f}'} \\ &= \widehat{(x-\chi)}_{\boldsymbol{f}_v}, \end{aligned}$$

because $\text{FrequencyCone}_T(v) = \{\boldsymbol{f}_v\}$ due to the assumption that $l_T(v) = d\log_2 n$. Thus we get that $\widehat{\chi}(\cdot) \leftarrow \widehat{\chi}(\cdot) + \widehat{(x-\chi)}_{\boldsymbol{f}_v}\delta_{\boldsymbol{f}_v}(\cdot)$ therefore at the end of the loop we have $\widehat{(x-\chi)}_{\boldsymbol{f}_v} = 0$ which means that $\boldsymbol{f}_v$ will no longer be in supp $\widehat{(x-\chi)}$. And also v gets removed from tree T implying

that $\{\boldsymbol{f}_v\} = \text{FrequencyCone}_T(v)$ will be excluded from $\bigcup_{u\text{: leaf of } T} \text{FrequencyCone}_T(u)$. Note that this also implies that $\widehat{(x-\chi)}$ will remain a worst-case signal with random phase. Therefore, condition **(1)** and **(2)** and **(3)** hold.

Now, note that $\|\widehat{(x-\chi)}\|_0$ will decrease by 1 exactly because $\boldsymbol{f}_v$ is no longer in supp $\widehat{(x-\chi)}$ and the rest of the support is unchanged. This shows that condition **(5)** holds. Also, $\sum_{u\text{: leaf of } T} l_T(u)$ decreases by exactly $d\log_2 n$ because the level of v was $l_T(v) = d\log_2 n$ and v gets removed from T. So ϕ will decrease by exactly one as required in condition **(4)**.

Case 2 Suppose that $l_T(v) < d\log_2 n$. We first check that the invocation of ZEROTEST satisfies preconditions of Lemma 7. We need to ensure that for the residual signal $\widehat{x} - \widehat{\chi}$ one has

$$\text{supp } (\widehat{x} - \widehat{\chi}) \subseteq \bigcup_{u\text{: leaf of } T'} \text{FrequencyCone}_{T'}(u),$$

where T' is the tree obtained from T by adding two children of v (line 19). This follows, since by the inductive hypothesis we have

$$\text{supp } (\widehat{x} - \widehat{\chi}) \subseteq \bigcup_{u\text{: leaf of } T} \text{FrequencyCone}_{T}(u),$$

and because by claim 6 we have,

$$\text{FrequencyCone}_T(v) = \text{FrequencyCone}_{T'}(u) \cup \text{FrequencyCone}_{T'}(v).$$

We thus get that the preconditions of Lemma 7 are satisfied, and the output of $\text{ZEROTEST}(x, \widehat{\chi}, T', w, n, d, \boldsymbol{\Delta})$ is **true** if $(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_{T'}(w)} \neq 0$ and **false** otherwise. A similar analysis shows that the algorithm correctly tests the zero hypothesis on $(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_{T'}(u)}$. We thus get, letting T_{new} denote the tree T at the end of the while loop, that

$$\text{supp } (\widehat{x} - \widehat{\chi}) \subseteq \bigcup_{u\text{: leaf of } T_{new}} \text{FrequencyCone}_{T_{new}}(u),$$

and for every $v \in T_{new}$ one has supp $(\widehat{x} - \widehat{\chi}) \cap \text{FrequencyCone}_{T_{new}}(v) \neq \emptyset$. Hence, because $\widehat{(x-\chi)}$ remains unchanged, conditions **(1)** and **(2)** and **(3)** hold at the end of the loop.

Now, we show ϕ is decreased at least by one. By inductive hypothesis supp $(\widehat{x} - \widehat{\chi}) \cap \text{FrequencyCone}_T(v) \neq \emptyset$ and at least one of w or u will be added to T because $\text{FrequencyCone}_T(v) = \text{FrequencyCone}_{T'}(u) \cup \text{FrequencyCone}_{T'}(v)$. Note that $l_{T_{new}}(w) = l_{T_{new}}(u) = l_T(v) + 1$ hence $\sum_{u'\text{: leaf of } T} l_{T_{new}}(u') \geq \sum_{u'\text{: leaf of } T} l_T(u') + 1$. Because $\|\widehat{x} - \widehat{\chi}\|_0$ remains unchanged, the value of ϕ will decrease by at least one hence conditions **(4)** and **(d)** hold.

Because $l_T(u) \leq d\log_2 n$ for every leaf $u \in T$, it follows from condition (2) that the quantity $\phi = (d\log_2 n + 1)\|\widehat{x} - \widehat{\chi}\|_0 - \sum_{u\text{: leaf of } T} l_T(u)$ is non-negative. At the first iteration, $\widehat{\chi} = 0$ and $T = \{r\}$ where r is the root with $l_T(f) = 0$. Hence, $\phi = \|\widehat{x}\|_0(1 + d\log_2 n)$ at first iteration. Because ϕ is decreasing by at least 1 at each iteration, the algorithm terminates after $O(\|\widehat{x}\|_0 \cdot d\log_2 n)$ iterations. By Lemma 7 along with Claim 7, the runtime of each iteration of algorithm is $O(km)$ and also sample complexity of each iteration is $O(km)$ therefore the total runtime and sample complexity both will be $O(k^2 m \cdot d\log_2 n)$.

Finally, observe that throughout this analysis we have assumed that the set $\boldsymbol{\Delta}$ satisfies the precondition of Lemma 7 for all the invocations of ZEROTEST by our algorithm.

In reality, there are two cases. The first case is for worst-case signals (Algorithm 4, Theorem 3). In this case, the algorithm chooses $\boldsymbol{\Delta}$ to be a multiset which corresponds to the Fourier

measurements with RIP of order k. Let F_N^{-1} be the d dimensional inverse Fourier transform's matrix with $N = n^d$ points. The matrix $M = \sqrt{N}F_N^{-1}$ is a unitary matrix. If you let $M_{\mathbf{\Delta}}$ denote the submatrix of M whose rows are sampled from M according to set $\mathbf{\Delta}$ defined in line 5 of Algorithm 4 then by Theorem 7 there exists a multiset $\mathbf{\Delta}$ of size $m = O(k \log_2^2 k \cdot d \log_2 n)$ such that $\frac{\sqrt{N}}{|\mathbf{\Delta}|} M_{\mathbf{\Delta}}$ satisfies the restricted isometry property of order k. Therefore, for every signal $y \in \mathbb{C}^{n^d}$:

$$\frac{1}{2} \cdot \frac{\|\widehat{y}\|_2^2}{n^{2d}} \leq \frac{1}{|\mathbf{\Delta}|} \cdot \sum_{\Delta \in \mathbf{\Delta}} |y_\Delta|^2 \leq \frac{3}{2} \cdot \frac{\|\widehat{y}\|_2^2}{n^{2d}}$$

As we have shown in condition **(5)** of the induction, $\|\widehat{x} - \widehat{\chi}\|_0 \leq k$. Hence for every leaf v of the tree $\|(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_T(v)}\|_0 \leq k$ therefore the precondition of lemma 7 is satisfied.

The second case is for worst-case signals with random phase (Algorithm 5, Theorem 4). We have shown in condition **(3)** of the induction that $x - \chi$ is a worst-case signal with random phase in every iteration of the algorithm. Therefore for every leaf v of the tree it is true that $(\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_T(v)}$ is a worst-case signal with random phase. In this case, the multiset $\mathbf{\Delta}$ is defined in line 5 of Algorithm 5 therefore by Lemma 6 for a fixed leaf v of tree T with probability at least $1 - 1/n^{4d}$ the following holds:

$$\frac{1}{2} \cdot \frac{\|\widehat{y}\|_2^2}{n^{2d}} \leq \frac{1}{|\mathbf{\Delta}|} \cdot \sum_{\Delta \in \mathbf{\Delta}} |y_\Delta|^2 \leq \frac{3}{2} \cdot \frac{\|\widehat{y}\|_2^2}{n^{2d}}$$

where $y = (\widehat{x} - \widehat{\chi})_{\text{FrequencyCone}_T(v)}$.

This shows that in the second case which corresponds to theorem 4, the failure probability of procedure ZeroTest is at most $\frac{1}{n^{4d}}$. Moreover, the above analysis shows that SFFT makes at most $O(kd \log_2 n)$ calls to ZeroTest. Therefore, by a union bound, the overall failure probability of the calls to ZeroTest is $O\left((kd \log_2 n)\frac{1}{n^{4d}}\right) \leq O(n^{-2d})$. Hence, we obtain the desired result.

□

8 Signals with random support in high dimension

In this section, we consider Fourier sparse signals whose support in the frequency domain is chosen randomly, while the values at the nonzero frequencies are chosen arbitrarily (worst-case). In other words, we assume a Bernoulli model for supp $\widehat{x}$, while the values at the frequencies that are chosen to be in the support are arbitrary. We will present an algorithm that runs in time $O(k \log^{O(1)} N)$. The model for random support signals can be found in Definition 11 (Section 2), which we restate here for convenience of the reader:

Definition 11 (Random support signal)**.** For any positive integer d, power of two n, and arbitrary $\beta : [n]^d \to \mathbb{C}$, we define $x : [n]^d \to \mathbb{C}$ to be a *random support signal* of Fourier sparsity k (with values given by β) if $\widehat{x}$ is the signal defined by

$$\widehat{x}_{\boldsymbol{f}} = \begin{cases} \beta_{\boldsymbol{f}} & \text{with probability } k/n^d \\ 0 & \text{with probability } 1 - k/n^d \end{cases},$$

where the $x_{\boldsymbol{f}}$ are independently chosen for the various $\boldsymbol{f} \in [n]^d$.

In other words, we assume a Bernoulli model for supp $\widehat{x}$, while the values at the frequencies that are chosen to be in the support are arbitrary.

8.1 Outline of our approach

The algorithm is motivated by the idea of speeding up our algorithm for worst-case signals (Algorithm 4, also see Theorem 3) by reducing the number of iterations of the process from $\Theta(k)$ down to $O(\log N)$. Such a reduction (which we show to be impossible for worst-case signals in Section 6) requires the ability to peel off many elements of the residual in a single phase of the algorithm, which turns out to be possible if the support of $\widehat{x}$ is chosen uniformly at random as in Definition 11. Indeed, if one considers the splitting tree T of a signal with uniformly random support, one sees that

(a) a large constant fraction of nodes $v \in T$ satisfy $w_T(v) \leq \log_2 k + O(1)$;

(b) the adaptive aliasing filters G constructed for such nodes will have significantly overlapping support in time domain.

We provide the intuition for this for the one-dimensional setting ($d = 1$) to simplify notation (changes required in higher dimensions are minor). In this setting, property **(b)** above is simply a manifestation of the fact that since the support is uniformly random, any given congruence classes modulo $B' = Ck$ for a large enough constant $C > 1$ is likely to contain only a single element of the support of $\widehat{x}$. Our adaptive aliasing filters provide a way to only partition frequency space along a carefully selected subset of bits in $[\log_2 N]$, but due to the randomness assumption, one can isolate most of the elements by simply partitioning using the bottom $\log_2 k + O(1)$ bits. This essentially corresponds to hashing $\widehat{x}$ into $B = Ck$ buckets at computational cost $O(B' \log B') = O(k \log k)$. While this scheme is efficient, it unfortunately only recovers a constant fraction of coefficients. One solution would be to hash into $B = Ck^2$ buckets (i.e., consider congruence classes modulo Ck^2), which would result in perfect hashing with good constant probability, allowing us to recover the entire signal in a single round. However, this hashing scheme would result in a runtime of $\Omega(k^2 \log k)$ and is, hence, not satisfactory. On the other hand, hashing into Ck^2 buckets is clearly wasteful, as most buckets would be empty. Our main algorithmic contribution is a way of "implicitly" hashing into Ck^2 buckets, i.e., getting access to the nonempty buckets, at an improved cost of $\widetilde{O}(k)$.

Our algorithm uses an iterative approach, and the main underlying observation is very simple. Suppose that we are given the ability to "implicitly" hash into B buckets for some B, namely, get access to the nonempty buckets. If B is at least $\min(Ck^2, N)$, we know that there are no collisions with high probability and we are done. If not, then we show that, given access to nonempty buckets in the B-hashing (i.e. a hashing into B buckets), we can get access to the nonempty buckets of a (ΓB)-hashing for some appropriately chosen constant $\Gamma > 1$ at a polylogarithmic cost in the size of each nonempty bucket of the B-bucketing by essentially computing the Fourier transform of the signal restricted to nonempty buckets in the B-bucketing. We then proceed iteratively in this manner, starting with $B = Ck$, for which we can perform the hashing explicitly. Since the number of nonzero frequencies remaining in the residual after t iterations of this process decays geometrically in t, we can also afford to use a smaller number of buckets B' in the hashing that we construct explicitly, ensuring that the runtime is dominated by the first iteration.

Ultimately, the algorithm takes the following form. At every iteration, we explicitly compute a hashing into $B^{\text{base}} \leq Ck$ buckets explicitly. Then, using a list of nonempty buckets in a B^{prev}-hashing from the previous iteration, we extend this list to a list of nonempty buckets in a B^{next}-bucketing at polylogarithmic cost per bucket (by solving a well-conditioned linear system,

see Algorithm 6), where $B^{\text{next}} = \Gamma \cdot B^{\text{prev}}$ for some large enough constant $\Gamma > 1$. Meanwhile, we reduce B^{base} by a factor of Γ, thus maintaining the invariant $B^{\text{base}} \cdot B^{\text{next}} \approx k^2$ at all times (note that this is satisfied at the start, when $B^{\text{base}} = B^{\text{prev}} \approx k$, and $B^{\text{base}} \cdot B^{\text{next}}$ remains invariant at each iteration). Therefore, after a logarithmic number of iterations, we have effectively emulated hashing into $\approx k^2$ but at a total cost of roughly one hashing computation into $\approx k$ buckets (see Figure 6 for an illustration).

Bucketing in high dimensions (MakeBucket function). We note that our vectorial notation for buckets in high dimensions (see section 8.2) allows us to continue talking about bucketings with $\boldsymbol{B}^{base}, \boldsymbol{B}^{prev}, \boldsymbol{B}^{next}$ buckets, even though now the number of buckets is in fact a vector of length d. In fact in dimension d the only property of the bucketing that matters for our analysis is the number of buckets $|\boldsymbol{B}^{base}|, |\boldsymbol{B}^{prev}|, |\boldsymbol{B}^{next}|$ and the shape of each bucket is not important (this is due to the fact that the support is sampled from a permutation invariant distribution). In order to avoid unnecessary notation overload, in Algorithm 8 we introduce procedure MakeBucket that constructs a bucketing $\mathbf{B}$ of size $|\mathbf{B}| = b$ of the following simple form. The vector $\boldsymbol{B}$ is defined by

$$\boldsymbol{B}_p = \begin{cases} n & \text{if } p \leq \lfloor \log_n b \rfloor \\ \frac{b}{n^p} & \text{if } p = \lfloor \log_n b \rfloor + 1 \\ 1 & \text{o.w.} \end{cases}$$

The pseudocode for MakeBucket, which implements the formula above, is given in Algorithm 8.

8.2 Notation

We will need notations for vectorial operations, e.g., entrywise multiplication and/or division of vectors, which is defined in the following definition.

Definition 24 (Entrywise vectorial arithmetic)**.** Suppose that $\boldsymbol{B} = (B_1, B_2, \cdots, B_d)$, $\boldsymbol{j} = (j_1, j_2, \cdots, j_d)$ and $\boldsymbol{t} = (t_1, t_2, \cdots, t_d)$ are d-dimensional vectors and a is a scalar value. Then we define the following operations,

$\boldsymbol{j} \cdot \boldsymbol{t}$	d-dimensional vector $(j_1 \cdot t_1, j_2 \cdot t_2, \cdots, j_d \cdot t_d)$.
$\boldsymbol{j}/\boldsymbol{t}$	d-dimensional vector $(j_1/t_1, j_2/t_2, \cdots, j_d/t_d)$.
$a/\boldsymbol{t}$	d-dimensional vector $(a/t_1, a/t_2, \cdots, a/t_d)$.
$\boldsymbol{j} \equiv \boldsymbol{t} \pmod{\boldsymbol{B}}$	entrywise congruence mod $\boldsymbol{B}$, i.e., for all $q = 1, \ldots, d$, one has $j_q \equiv t_q \pmod{B_q}$.
$\boldsymbol{j} \pmod{\boldsymbol{B}}$	d-dimensional vector $(j_1 \pmod{B_1}, j_2 \pmod{B_2}, \cdots, j_d \pmod{B_d})$.
$\boldsymbol{j} \pmod{a}$	d-dimensional vector $(j_1 \pmod{a}, j_2 \pmod{a}, \cdots, j_d \pmod{a})$.
$\lvert\boldsymbol{j}\rvert$	the product of all the entries of vector $\boldsymbol{j}$, i.e., $\lvert\boldsymbol{j}\rvert = j_1 j_2 \cdots j_d$.
$[\boldsymbol{B}]$	Cartesian product $[B_1] \times [B_2] \times \cdots \times [B_d]$.

8.3 Outline of our approach

The algorithm is motivated by the idea of speeding up our algorithm for worst-case signals (Algorithm 4, also see Theorem 3) by reducing the number of iterations of the process from

$\Theta(k)$ down to $O(\log k)$. Such a reduction (which is shown to be impossible for worst-case signals in Section 6) requires the ability to peel off many elements of the residual in a single phase of the algorithm, which turns out to be possible if the support of $\widehat{x}$ is chosen uniformly at random as in Definition 11. Indeed, if one considers the splitting tree T of a signal with uniformly random support (see Figure 5 for an illustration), one sees that

(a) a large constant fraction of nodes $v \in T$ satisfy $w_T(v) \leq \log_2 k + O(1)$;

(b) the adaptive aliasing filters G constructed for such nodes will have significantly overlapping support in time domain.

We provide the intuition for this for the one-dimensional setting ($d = 1$) to simplify notation (changes required in higher dimensions are minor). In this setting, property **(b)** above is simply a manifestation of the fact that since the support is uniformly random, any given non-empty congruence class modulo $B' = Ck$ for a large enough constant $C > 1$ is likely to contain only a single element of the support of $\widehat{x}$. Our adaptive aliasing filters provide a way to only partition frequency space along a carefully selected subset of bits in $[\log_2 N]$, but due to the randomness assumption, one can isolate most of the elements by simply partitioning using the bottom $\log_2 k + O(1)$ bits. This essentially corresponds to hashing $\widehat{x}$ into $B = Ck$ buckets at computational cost $O(B' \log B') = O(k \log k)$. While this scheme is efficient, it unfortunately only recovers a constant fraction of coefficients. One solution would be to hash into $B = Ck^2$ buckets (i.e., consider congruence classes modulo Ck^2), which would result in perfect hashing with good constant probability, allowing us to recover the entire signal in a single round. However, this hashing scheme would result in a runtime of $\Omega(k^2 \log k)$ and is, hence, not satisfactory. On the other hand, hashing into Ck^2 buckets is clearly wasteful, as most buckets would be empty. Our main algorithmic contribution is a way of "implicitly" hashing into Ck^2 buckets, i.e., getting access to the nonempty buckets, at an improved cost of $\widetilde{O}(k)$.

Our algorithm uses an iterative approach, and the main underlying observation is very simple. Suppose that we are given the ability to "implicitly" hash into B buckets for some B, namely, get access to the nonempty buckets. If B is at least $\min(Ck^2, N)$, we know that there are no collisions with high probability and we are done. If not, then we show that, given access to nonempty buckets in the B-hashing (i.e. a hashing into B buckets), we can get access to the nonempty buckets of a (ΓB)-hashing for some appropriately chosen constant $\Gamma > 1$ at a polylogarithmic cost in the size of each nonempty bucket of the B-bucketing by essentially computing the Fourier transform of the signal restricted to nonempty buckets in the B-bucketing. We then proceed iteratively in this manner, starting with $B = Ck$, for which we can perform the hashing explicitly. Since the number of nonzero frequencies remaining in the residual after t iterations of this process decays geometrically in t, we can also afford to use a smaller number of buckets B' in the hashing that we construct explicitly, ensuring that the runtime is dominated by the first iteration.

Ultimately, the algorithm takes the following form. At every iteration, we explicitly compute a hashing into $B^{\text{base}} \leq Ck$ buckets explicitly. Then, using a list of nonempty buckets in a B^{prev}-hashing from the previous iteration, we extend this list to a list of nonempty buckets in a B^{next}-bucketing at polylogarithmic cost per bucket (by solving a well-conditioned linear system, see Algorithm 6), where $B^{\text{next}} = \Gamma \cdot B^{\text{prev}}$ for some large enough constant $\Gamma > 1$. Meanwhile, we reduce B^{base} by a factor of Γ, thus maintaining the invariant $B^{\text{base}} \cdot B^{\text{next}} \approx k^2$ at all times (note that this is satisfied at the start, when $B^{\text{base}} = B^{\text{prev}} \approx k$, and $B^{\text{base}} \cdot B^{\text{next}}$ remains invariant at each iteration). Therefore, after a logarithmic number of iterations, we have effectively emulated

hashing into $\approx k^2$ but at a total cost of roughly one hashing computation into $\approx k$ buckets (see Figure 6 for an illustration).

Bucketing in high dimensions (MakeBucket function). We note that our vectorial notation for buckets in high dimensions (see section 8.2) allows us to continue talking about bucketings with $\boldsymbol{B}^{base}, \boldsymbol{B}^{prev}, \boldsymbol{B}^{next}$ buckets, even though now the number of buckets is in fact a vector of length d. In fact in dimension d the only property of the bucketing that matters for our analysis is the number of buckets $|\boldsymbol{B}^{base}|, |\boldsymbol{B}^{prev}|, |\boldsymbol{B}^{next}|$ and the shape of each bucket is not important (this is due to the fact that the support is sampled from a permutation invariant distribution). In order to avoid unnecessary notation overload, in Algorithm 8 we introduce procedure MakeBucket that constructs a bucketing $\mathbf{B}$ of size $|\mathbf{B}| = b$ of the following simple form. The vector $\boldsymbol{B}$ is defined by

$$\boldsymbol{B}_p = \begin{cases} n & \text{if } p \leq \lfloor \log_n b \rfloor \\ \frac{b}{n^p} & \text{if } p = \lfloor \log_n b \rfloor + 1 \\ 1 & \text{o.w.} \end{cases}$$

The pseudocode for MakeBucket, which implements the formula above, is given in Algorithm 8.

8.4 Filtering, hashing, and bucketing in high dimensions

We introduce the main definitions here. Our techniques in this section use a version of our adaptive aliasing filters that is taylored to the assumption that the support of $\widehat{x}$ is chosen uniformly at random. Since the signal is assumed to be sampled from a distribution, we are able to design a fast algorithm by adapting to a distribution as opposed to a given realization of the support of $\widehat{x}$. The next definition is essentially a simplified version of the definition of a frequency cone from Section 2 (see Definition 4):

Definition 25 (Congruence classes of support)**.** Suppose d and n are positive integers such that n is a power of two. Let $\boldsymbol{B} = (B_1, B_2, \ldots, B_d)$ be a vector of powers of two such that $B_j \mid n$ for $j = 1, 2, \ldots, d$. For every $\boldsymbol{b} \in [\boldsymbol{B}]$, and signal $x \in \mathbb{C}^{n^d}$, we define the $(\boldsymbol{B}, \boldsymbol{b})$-*congruence class* of supp $\widehat{x}$ to be the set $S_x(\boldsymbol{B}, \boldsymbol{b})$, given by

$$S_x(\boldsymbol{B}, \boldsymbol{b}) = \{\boldsymbol{f} \in \text{supp } \widehat{x} : \boldsymbol{f} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}}\}.$$

We access the signal using a bucketing operation, defined below.

Definition 26 (Bucketing in high dimensions)**.** Suppose d and n are positive integers such that n is a power of two. Let $\boldsymbol{B} = (B_1, B_2, \ldots, B_d)$ be a vector of powers of two such that $B_j \mid n$ for all $j = 1, 2, \ldots, d$. For every $\boldsymbol{a} \in [n]^d$, $\boldsymbol{b} \in [\boldsymbol{B}]$, and signal $x \in \mathbb{C}^{n^d}$, we define the $(\boldsymbol{B}, \boldsymbol{b})$-*bucketing* of x with *shift* $\boldsymbol{a}$ to be $U_x^{\boldsymbol{a}}(\boldsymbol{B}, \boldsymbol{b})$, given by

$$\begin{aligned} U_x^{\boldsymbol{a}}(\boldsymbol{B}, \boldsymbol{b}) &= \sum_{\boldsymbol{f} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}}} \widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{a}}{n}} \\ &= \sum_{\boldsymbol{f} \in S_x(\boldsymbol{B}, \boldsymbol{b})} \widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{a}}{n}}. \end{aligned}$$

The following definition of Bernoulli set provides a compact way of referring to the distribution of supp $\widehat{x}$:

Definition 27 (Bernoulli set)**.** For every power of two n and positive integer d, let $N = n^d$ and set $S \subseteq [n]^d$ be a random set such that each $\boldsymbol{j} \in [n]^d$ is independently chosen to be in S with probability k/N,

$$\Pr[\boldsymbol{j} \in S] = \frac{k}{N}.$$

Moreover, for any $\boldsymbol{B} = (B_1, B_2, \ldots, B_d)$ such that $B_1, B_2, \ldots, B_d \mid n$, we define

$$S^{(\boldsymbol{B})} = \{\boldsymbol{f} \in [\boldsymbol{B}] : \exists\, \boldsymbol{g}, \boldsymbol{h} \in S \text{ such that } \boldsymbol{g} \neq \boldsymbol{h}, \boldsymbol{f} \equiv \boldsymbol{g} \equiv \boldsymbol{h} \pmod{\boldsymbol{B}}\}.$$

The next lemma is crucial for our analysis. The lemma considers two bucketings $\boldsymbol{B}$ and $\boldsymbol{B}'$, where $\boldsymbol{B}$ is a refinement of $\boldsymbol{B}$. The object of interest is the number of buckets in the bucketing $\boldsymbol{B}$ that contain at least two elements of a Bernoulli set S, i.e. **non-singleton buckets**. The lemma shows that as long as the product of the number of buckets in $\boldsymbol{B}$ and $\boldsymbol{B}'$ is at least k^2, the elements (i.e. frequencies) of a Bernoulli set S that belong to non-singleton buckets in $\boldsymbol{B}$ must be rather uniformly spread over the coarser bucketing $\boldsymbol{B}'$. Specifically, no bucket in $\boldsymbol{B}'$ contains more than $O(\log N)$ such frequencies with high probability. We will use this lemma with $\boldsymbol{B}' = \boldsymbol{B}^{base}$ and $\boldsymbol{B} = \boldsymbol{B}^{next}$ and $\boldsymbol{B} = \boldsymbol{B}^{prev}$ (see proof of Theorem 6 below).

Lemma 8 (Refinement lemma)**.** *For any power of two integers n and k, suppose $\boldsymbol{B} = (B_1, B_2, \ldots B_d)$ and $\boldsymbol{B}' = (B'_1, B'_2, \ldots, B'_d)$ satisfy $B'_j \mid B_j \mid n$ for all $j = 1, 2, \ldots, d$ as well as $|\boldsymbol{B}| \cdot |\boldsymbol{B}'| \geq k^2$. Then, with probability at least $1 - \frac{1}{N^3}$, we have that for all $\boldsymbol{b} \in [\boldsymbol{B}']$,*

$$\left| S^{(\boldsymbol{B})} \cap \{\boldsymbol{f} \in [\boldsymbol{B}] : \boldsymbol{f} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}'}\} \right| = O(\log N).$$

The proof is a simple probabilistic argument and is given in Appendix B.

8.5 Hashing in high dimensions by using downsampling

The main primitive we need for developing an algorithm with $\tilde{O}(k)$ sample complexity is a hashing function based on downsampling, presented below as Algorithm 6. The algorithm takes as input the list R of buckets in the hashing into $\boldsymbol{B}^{\text{prev}}$ (that will later be guaranteed to be superset of the nonempty buckets in a $\boldsymbol{B}^{\text{prev}}$-hashing of the residual signal $x - \chi$) and outputs a list of potentially nonempty buckets in the hashing into $\boldsymbol{B}^{\text{next}}$, together with evaluations of the corresponding hashed signals at a point $\boldsymbol{\alpha}$ that is given as a parameter.

Lemma 9. (HASHING in high dimensions) *Suppose d and n are positive integers such that n is a power of two. Suppose $\boldsymbol{B}^{\text{base}} = (B_1^{\text{base}}, B_2^{\text{base}}, \ldots, B_d^{\text{base}})$, $\boldsymbol{B}^{\text{prev}} = (B_1^{\text{prev}}, B_2^{\text{prev}}, \ldots, B_d^{\text{prev}})$ and $\boldsymbol{B}^{\text{next}} = (B_1^{\text{next}}, B_2^{\text{next}}, \ldots, B_d^{\text{next}})$ are vectors of powers of two such that $B_j^{\text{prev}} \mid B_j^{\text{next}}$ and $B_j^{\text{base}} \mid B_j^{\text{prev}}$ for all j. Moreover, let $\boldsymbol{\alpha} \in [n]^d$ be a shift vector. For any signals $x, \widehat{\chi} \in \mathbb{C}^{n^d}$ suppose that*

$$R \supseteq \{\boldsymbol{b} \in [\boldsymbol{B}^{\text{prev}}] : S_{x-\chi}(\boldsymbol{B}^{\text{prev}}, \boldsymbol{b}) \neq \emptyset\}, \tag{12}$$

where $S_{x-\chi}(\boldsymbol{B}^{\text{prev}}, \boldsymbol{b})$ is $(\boldsymbol{B}^{\text{prev}}, \boldsymbol{b})$-congruence class of supp $\widehat{(x - \chi)}$ *as per Definition 25. Then the procedure* HASHING $(x, \widehat{\chi}, n, d, \boldsymbol{B}^{\text{base}}, \boldsymbol{B}^{\text{prev}}, \boldsymbol{B}^{\text{next}}, \boldsymbol{\alpha}, R)$ *outputs a set W that with probability $1 - \frac{1}{n^{10d}}$ satisfies*

$$W = \{(\boldsymbol{b}', U^{\boldsymbol{\alpha}}_{x-\chi}(\boldsymbol{B}^{\text{next}}, \boldsymbol{b}')) : \boldsymbol{b}' \in [\boldsymbol{B}^{\text{next}}] \text{ s.t. } \boldsymbol{b}' \equiv \boldsymbol{b} \pmod{\boldsymbol{B}^{\text{prev}}} \text{ for some } \boldsymbol{b} \in R\}. \tag{13}$$

Algorithm 6 Procedure for hashing a signal's Fourier transform using downsampling

procedure HASHING($x, \widehat{\chi}, n, d, \boldsymbol{B}^{\text{base}}, \boldsymbol{B}^{\text{prev}}, \boldsymbol{B}^{\text{next}}, \boldsymbol{\alpha}, R$) ▷ $\boldsymbol{\alpha}$ is a point in time domain
▷ R is the set of nonempty buckets in $\boldsymbol{B}^{\text{prev}}$
  $m \leftarrow \frac{|\boldsymbol{B}^{\text{next}}|}{|\boldsymbol{B}^{\text{prev}}|} \max_{\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]} \left|\left\{\boldsymbol{r} \in R : \boldsymbol{r} \equiv \boldsymbol{b} \pmod{B^{\text{base}}}\right\}\right|$
  $S \leftarrow \left\{\boldsymbol{\beta}_i : \boldsymbol{\beta}_i \sim \text{Unif}([\boldsymbol{B}^{\text{next}}]), \forall i \in [Cm \log_2^2 m \cdot d \log_2 n]\right\}$ ▷ S: multiset ▷ C: constant
  for $\boldsymbol{\beta} \in S$ **do**
    $\boldsymbol{a} \leftarrow \boldsymbol{\alpha} + \boldsymbol{\beta} \cdot \frac{n}{\boldsymbol{B}^{\text{next}}}$
    $Z_{\boldsymbol{b}}^{(\boldsymbol{\alpha},\boldsymbol{\beta})} \leftarrow \frac{n^d}{|\boldsymbol{B}^{\text{base}}|} x_{\boldsymbol{b} \cdot \frac{n}{\boldsymbol{B}^{\text{base}}} + \boldsymbol{a}}$ for all $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$
    $\widehat{Z}^{(\boldsymbol{\alpha},\boldsymbol{\beta})} \leftarrow FFT\left(Z^{(\boldsymbol{\alpha},\boldsymbol{\beta})}\right)$
    $\widehat{Z}_{\boldsymbol{b}}^{(\boldsymbol{\alpha},\boldsymbol{\beta})} \leftarrow \widehat{Z}_{\boldsymbol{b}}^{(\boldsymbol{\alpha},\boldsymbol{\beta})} - \sum_{\boldsymbol{r} \in \left[\frac{n}{\boldsymbol{B}^{\text{base}}}\right]} \widehat{\chi}_{\boldsymbol{b}+\boldsymbol{r}\cdot\boldsymbol{B}^{\text{base}}} e^{2\pi i \frac{\left(\boldsymbol{b}+\boldsymbol{r}\cdot\boldsymbol{B}^{\text{base}}\right)^T \boldsymbol{a}}{n}}$ for all $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$
  $W \leftarrow \emptyset$;
  for $\boldsymbol{b} \in [B^{\text{base}}]$ **do**
    $w_{\boldsymbol{\beta}} \leftarrow \widehat{Z}_{\boldsymbol{b}}^{(\boldsymbol{\alpha},\boldsymbol{\beta})}$ for all $\boldsymbol{\beta} \in S$
    $R_{\boldsymbol{b}} \leftarrow \{(\boldsymbol{r}, \boldsymbol{s}) : \boldsymbol{b} + \boldsymbol{r} \cdot \boldsymbol{B}^{\text{base}} \in R \text{ and } \boldsymbol{s} \in [\boldsymbol{B}^{\text{next}}/\boldsymbol{B}^{\text{prev}}]\}$
    $(\mathbf{A}_{\boldsymbol{b}})_{\boldsymbol{\beta},(\boldsymbol{r},\boldsymbol{s})} \leftarrow e^{2\pi i (\boldsymbol{b}+\boldsymbol{r}\boldsymbol{B}^{\text{base}}+\boldsymbol{s}\boldsymbol{B}^{\text{prev}})^T \left(\frac{\boldsymbol{\beta}}{\boldsymbol{B}^{\text{next}}}\right)}$ for all $\boldsymbol{\beta} \in S$ and $(\boldsymbol{r}, \boldsymbol{s}) \in R_{\boldsymbol{b}}$
    $\boldsymbol{v} \leftarrow$ LEASTSQUARESSOLVER($\mathbf{A}_{\boldsymbol{b}}, \boldsymbol{w}$)
    for $(\boldsymbol{r}, \boldsymbol{s}) \in R_{\boldsymbol{b}}$ **do**
      $W \leftarrow W \cup \left\{\left(\boldsymbol{b} + \boldsymbol{r}\boldsymbol{B}^{\text{base}} + \boldsymbol{s}\boldsymbol{B}^{\text{prev}}, \boldsymbol{v}_{(\boldsymbol{r},\boldsymbol{s})}^{(\boldsymbol{b})}\right)\right\}$
  return W
procedure LEASTSQUARESSOLVER(A, b)
  return $(A^T A)^{-1} A^T b$

Moreover, the sample complexity of this procedure is

$$O\left(m\log^2 m\cdot d\log n\cdot\left|\boldsymbol{B}^{\text{base}}\right|\right),$$

while the time complexity of this procedure is

$$O\left((m\log^2 m\cdot d\log n)^3\cdot\left|\boldsymbol{B}^{\text{base}}\right|+(m\log^2 m\cdot d\log n)\cdot\left(\left|\boldsymbol{B}^{\text{base}}\right|\log_2\left(|\boldsymbol{B}^{\text{base}}|\right)+\|\widehat{\chi}\|_0\right)\right),$$

where $m=\frac{|\boldsymbol{B}^{\text{next}}|}{|\boldsymbol{B}^{\text{prev}}|}\max_{\boldsymbol{b}\in[\boldsymbol{B}^{\text{base}}]}\left|\left\{\boldsymbol{r}\in R:\boldsymbol{r}\equiv\boldsymbol{b}\pmod{B^{\text{base}}}\right\}\right|$.

Proof. Recall that the algorithm uses a coarse $\boldsymbol{B}^{\text{base}}$-bucketing to refine a $\boldsymbol{B}^{\text{prev}}$-bucketing, for which only the non-empty buckets are computed, to a $\boldsymbol{B}^{\text{next}}$-bucketing. Let $x'=x-\chi$ and $\boldsymbol{a}=\boldsymbol{\alpha}+\boldsymbol{\beta}\cdot\frac{n}{\boldsymbol{B}^{\text{next}}}$, where $\boldsymbol{\alpha}=\boldsymbol{a}\bmod\frac{n}{\boldsymbol{B}^{\text{next}}}$. Suppose (12) holds.

Note that for any $\boldsymbol{b}\in[\boldsymbol{B}^{\text{base}}]$,

$$\begin{aligned}
U^{\boldsymbol{a}}_{x'}(\boldsymbol{B}^{\text{base}},\boldsymbol{b}) &= \sum_{\boldsymbol{f}\equiv\boldsymbol{b}\pmod{\boldsymbol{B}^{\text{base}}}}\widehat{x}'_{\boldsymbol{f}}e^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{a}}{n}}\\
&= \sum_{\boldsymbol{r}\in\left[\frac{\boldsymbol{B}^{\text{prev}}}{\boldsymbol{B}^{\text{base}}}\right]}\sum_{\boldsymbol{s}\in\left[\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{prev}}}\right]}\sum_{\boldsymbol{f}\equiv\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})\pmod{\boldsymbol{B}^{\text{next}}}}\widehat{x}'_{\boldsymbol{f}}e^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{a}}{n}}\\
&= \sum_{\boldsymbol{r}\in\left[\frac{\boldsymbol{B}^{\text{prev}}}{\boldsymbol{B}^{\text{base}}}\right]}\sum_{\boldsymbol{s}\in\left[\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{prev}}}\right]}e^{2\pi i\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})^T\left(\frac{\boldsymbol{\beta}}{\boldsymbol{B}^{\text{next}}}\right)}\sum_{\boldsymbol{q}\in\left[\frac{n}{\boldsymbol{B}^{\text{next}}}\right]}\widehat{x}'_{\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})+\boldsymbol{q}\cdot\boldsymbol{B}^{\text{next}}}e^{2\pi i\frac{(\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})+\boldsymbol{q}\cdot\boldsymbol{B}^{\text{next}})^T\boldsymbol{\alpha}}{n}}\\
&= \sum_{\boldsymbol{r}\in\left[\frac{\boldsymbol{B}^{\text{prev}}}{\boldsymbol{B}^{\text{base}}}\right]}\sum_{\boldsymbol{s}\in\left[\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{prev}}}\right]}e^{2\pi i\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})^T\frac{\boldsymbol{\beta}}{\boldsymbol{B}^{\text{next}}}}\cdot U^{\boldsymbol{\alpha}}_{x'}(\boldsymbol{B}^{\text{next}},\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})),
\end{aligned}\tag{14}$$

where $\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})$ denotes $\boldsymbol{b}+\boldsymbol{r}\boldsymbol{B}^{\text{base}}+\boldsymbol{s}\boldsymbol{B}^{\text{prev}}\in[\boldsymbol{B}^{\text{next}}]$ for ease of notation. Hence, $\boldsymbol{B}^{\text{base}}$-bucketing and $\boldsymbol{B}^{\text{next}}$-bucketing are related via a linear system which is defined in (14) for any collection of values of $\boldsymbol{\beta}$. We also show how to choose a relatively small number of values of $\boldsymbol{\beta}$ such that the above linear system will be well-conditioned. Note that $\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{prev}}}$ is a vector of length d, as per our vectorial notations in section 8.2, which denotes by how much we want to further refine each bucket of $\boldsymbol{B}^{\text{prev}}$-bucketing.

Choosing $\boldsymbol{\beta}$'s that make the linear system in (14) **well-conditioned:** For every $\boldsymbol{b}\in[\boldsymbol{B}^{\text{base}}]$, let $\mathbf{A}_{\boldsymbol{b}}$ denote the $\left|\boldsymbol{B}^{\text{next}}\right|\times\left|\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{base}}}\right|$ matrix whose rows are indexed by $\boldsymbol{\beta}\in[\boldsymbol{B}^{\text{next}}]$ and columns are indexed by $(\boldsymbol{r},\boldsymbol{s})\in[\frac{\boldsymbol{B}^{\text{prev}}}{\boldsymbol{B}^{\text{base}}}]\times[\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{prev}}}]$, with entries defined by

$$(\mathbf{A}_{\boldsymbol{b}})_{\boldsymbol{\beta},(\boldsymbol{r},\boldsymbol{s})}=e^{2\pi i\boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})^T\left(\frac{\boldsymbol{\beta}}{\boldsymbol{B}^{\text{next}}}\right)}.$$

Moreover, let $\boldsymbol{v}_b$ denote the column vector of length $\left|\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{base}}}\right|$ with entries indexed by $(\boldsymbol{r},\boldsymbol{s})\in[\frac{\boldsymbol{B}^{\text{prev}}}{\boldsymbol{B}^{\text{base}}}]\times[\frac{\boldsymbol{B}^{\text{next}}}{\boldsymbol{B}^{\text{prev}}}]$ and given by

$$(\boldsymbol{v}_{\boldsymbol{b}})_{(\boldsymbol{r},\boldsymbol{s})}=U^{\boldsymbol{\alpha}}_{x'}(\boldsymbol{B}^{\text{next}},\phi(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})),\tag{15}$$

while we let $\boldsymbol{w_b}$ denote the column vector of length $\left|\boldsymbol{B}^{\text{next}}\right|$ with entries indexed by $\boldsymbol{\beta} \in [\boldsymbol{B}^{\text{next}}]$ and given by

$$\boldsymbol{w_\beta} = U_{x'}^{\boldsymbol{\alpha}+\boldsymbol{\beta}\cdot\frac{n}{\boldsymbol{B}^{\text{next}}}}(\boldsymbol{B}^{\text{base}}, \boldsymbol{b}). \tag{16}$$

Then, (14) implies the following linear system of equations,

$$\mathbf{A_b} \cdot \boldsymbol{v}^{(\boldsymbol{b})} = \boldsymbol{w}^{(\boldsymbol{b})}. \tag{17}$$

Next, for any $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$, let

$$R_{\boldsymbol{b}} = \{(\boldsymbol{r}, \boldsymbol{s}) : \boldsymbol{b} + \boldsymbol{r}\boldsymbol{B}^{\text{base}} \in R \text{ and } \boldsymbol{s} \in [\boldsymbol{B}^{\text{next}}/\boldsymbol{B}^{\text{prev}}]\}.$$

By the assumption of the lemma on the set R, one has that the vectors $\boldsymbol{v_b}$ in (15) satisfy $\operatorname{supp} \boldsymbol{v_b} \subseteq R_{\boldsymbol{b}}$ for every $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$. This shows that the linear system in (17) can be solved very efficiently by randomly sampling its rows. More formally, suppose that S is a multiset such that $S = \left\{\boldsymbol{\beta}_i : \boldsymbol{\beta}_i \sim \text{Unif}([\boldsymbol{B}^{\text{next}}]), \forall i \in [Cm \log_2^2 m \cdot d \log_2 n]\right\}$ where $m = \max_{\boldsymbol{b'} \in [\boldsymbol{B}^{\text{base}}]} |R_{\boldsymbol{b'}}|$ and let $\mathbf{A}_{\boldsymbol{b}}^S$ denote the submatrix of $\mathbf{A_b}$ whose rows are selected with respect to set S. Then by Theorem 7, the matrix $\frac{1}{|S|}\mathbf{A}_{\boldsymbol{b}}^S$ satisfies RIP of order m. We will use this property to solve the system (17) efficiently.

Let $\widetilde{\mathbf{A}}_{\boldsymbol{b}}$ be a submatrix of $\mathbf{A_b}$ with size $|S| \times |R_{\boldsymbol{b}}|$. Suppose that its rows are selected with respect to set S and its columns are selected with respect to $R_{\boldsymbol{b}}$. More specifically, $\widetilde{\mathbf{A}}_{\boldsymbol{b}}$ is a $|S| \times |R_{\boldsymbol{b}}|$ matrix whose rows are indexed by $\boldsymbol{\beta} \in S$ and columns are indexed by $(\boldsymbol{r}, \boldsymbol{s}) \in R_{\boldsymbol{b}}$, with entries defined by

$$(\widetilde{\mathbf{A}}_{\boldsymbol{b}})_{\boldsymbol{\beta},(\boldsymbol{r},\boldsymbol{s})} = e^{2\pi i \boldsymbol{\phi}(\boldsymbol{b},\boldsymbol{r},\boldsymbol{s})^T\left(\frac{\boldsymbol{\beta}}{\boldsymbol{B}^{\text{next}}}\right)}.$$

Moreover, let $\boldsymbol{v}^{(\boldsymbol{b})}$ denote the column vector of length $|R_{\boldsymbol{b}}|$ with entries indexed by elements of $R_{\boldsymbol{b}}$ and given by

$$\boldsymbol{v}^{(\boldsymbol{b})}_{(\boldsymbol{r},\boldsymbol{s})} = U_{x-\chi}^{\boldsymbol{\alpha}}(\boldsymbol{B}^{\text{next}}, \phi(\boldsymbol{b}, \boldsymbol{r}, \boldsymbol{s})), \tag{18}$$

while we let $\boldsymbol{w}^{(\boldsymbol{b})}$ denote the column vector of length $|S|$ with entries indexed by elements of S and given by

$$\boldsymbol{w}^{(\boldsymbol{b})}_{\boldsymbol{\beta}} = U_{x-\chi}^{\boldsymbol{\alpha}+\boldsymbol{\beta}\cdot\frac{n}{\boldsymbol{B}^{\text{next}}}}(\boldsymbol{B}^{\text{base}}, \boldsymbol{b}).$$

Then, (17) implies that

$$\widetilde{\mathbf{A}}_{\boldsymbol{b}} \cdot \boldsymbol{v}^{(\boldsymbol{b})} = \boldsymbol{w}^{(\boldsymbol{b})}. \tag{19}$$

Now, because the matrix

$$\frac{1}{|S|}\mathbf{A}_{\boldsymbol{b}}^S$$

satisfies RIP of order $|R_{\boldsymbol{b}}|$ for all $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$, one has that the condition number of $\widetilde{\mathbf{A}_{\boldsymbol{b}}}^T \widetilde{\mathbf{A}_{\boldsymbol{b}}}$ is at most $\sqrt{3}$. Therefore, a linear least squares solver can compute $\boldsymbol{v}^{(\boldsymbol{b})}$ efficiently and in a numerically stable way using the reduced linear system in (19). Note that lines 11-14 of Algorithm 6 carry out this procedure and compute $\boldsymbol{v}^{(\boldsymbol{b})}$ for each $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$.

Computing $U^{\boldsymbol{a}}_{x-\chi}(\boldsymbol{B}^{\text{base}}, \boldsymbol{b})$: Now we show how to compute $U^{\boldsymbol{a}}_{x-\chi}(\boldsymbol{B}^{\text{base}}, \boldsymbol{b})$ for any $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$ and $\boldsymbol{a} = \boldsymbol{\alpha} + \boldsymbol{\beta} \cdot \frac{n}{\boldsymbol{B}^{\text{next}}}$. By standard downsampling properties, we have that if $Z^{(\boldsymbol{\alpha},\boldsymbol{\beta})} : [\boldsymbol{B}^{\text{base}}] \to \mathbb{C}$ is the signal defined by

$$Z^{(\boldsymbol{\alpha},\boldsymbol{\beta})}_{\boldsymbol{t}} = \frac{n^d}{|\boldsymbol{B}^{\text{base}}|} x_{\boldsymbol{t} \cdot \frac{n}{\boldsymbol{B}^{\text{base}}} + \boldsymbol{a}}$$

then its Fourier transform is given by

$$\widehat{Z}^{(\boldsymbol{\alpha},\boldsymbol{\beta})}_{\boldsymbol{b}} = \sum_{\boldsymbol{f} \equiv \boldsymbol{b} \pmod{B^{\text{base}}}} \widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{a}}{n}} = U^{\boldsymbol{a}}_{x}(B^{\text{base}}, \boldsymbol{b}).$$

Hence,

$$U^{\boldsymbol{a}}_{x-\chi}(B^{\text{base}}, \boldsymbol{b}) = \widehat{Z}^{(\boldsymbol{\alpha},\boldsymbol{\beta})}_{\boldsymbol{b}} - \sum_{\boldsymbol{f} \equiv \boldsymbol{b} \pmod{B^{\text{base}}}} \widehat{\chi}_{\boldsymbol{f}} e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{a}}{n}}, \tag{20}$$

which demonstrates how to compute $U^{\boldsymbol{a}}_{x-\chi}(\boldsymbol{B}^{\text{base}}, \boldsymbol{b})$. Note that lines 6-8 of Algorithm 6 simply compute $U^{\boldsymbol{a}}_{x-\chi}(B^{\text{base}}, \boldsymbol{b})$, for some $\boldsymbol{b} \in [B^{\text{base}}]$ with $\boldsymbol{a} = \boldsymbol{\alpha} + \boldsymbol{\beta} \cdot \frac{n}{\boldsymbol{B}^{\text{next}}}$ for some $\boldsymbol{\beta}$.

Sample complexity and Runtime: Lines 6-8 of Algorithm 6 compute $U^{\boldsymbol{a}}_{x-\chi}(B^{\text{base}}, \boldsymbol{b})$, for some $\boldsymbol{b} \in [B^{\text{base}}]$ with $\boldsymbol{a} = \boldsymbol{\alpha} + \boldsymbol{\beta} \cdot \frac{n}{\boldsymbol{B}^{\text{next}}}$ for some $\boldsymbol{\beta}$, in time $O(\left|\boldsymbol{B}^{\text{base}}\right| \log_2 \left(|\boldsymbol{B}^{\text{base}}|\right) + \|\widehat{\chi}\|_0)$ and with sample complexity $O\left(\left|\boldsymbol{B}^{\text{base}}\right|\right)$, according to the rule (20). This shows that the vector $\boldsymbol{w_b}$ in (16) can be constructed efficiently.

Note that lines 11-14 of Algorithm 6 carry out a least squares linear system procedure and compute $\boldsymbol{v}^{(\boldsymbol{b})}$ for each $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$ in time $O(|S|^3)$, as the time complexity of LeastSquares-Solver procedure is $O(|S|^3)$. Moreover, by (18), it follows that for a fixed $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$, line 16 simply adds all pairs $(\boldsymbol{b}', U^{\boldsymbol{\alpha}}_{x-\chi}(\boldsymbol{B}^{\text{next}}, \boldsymbol{b}'))$ with $\boldsymbol{b}' \in [\boldsymbol{B}^{\text{next}}]$ satisfying $(\boldsymbol{b}' \bmod B^{\text{prev}}) \in R$. Also, any $\boldsymbol{b}' \in [\boldsymbol{B}^{\text{next}}]$ for which there exists $\boldsymbol{f} \in \operatorname{supp} \widehat{x - \chi}$ with $\boldsymbol{f} \equiv \boldsymbol{b}' \pmod{\boldsymbol{B}^{\text{next}}}$ must satisfy $\boldsymbol{f} \equiv \boldsymbol{b}' \pmod{\boldsymbol{B}^{\text{prev}}}$ and, hence, $S_{x-\chi}(\boldsymbol{B}^{\text{prev}}, \boldsymbol{b}' \pmod{\boldsymbol{B}^{\text{prev}}}) \neq \emptyset$. This shows that $(\boldsymbol{b}' \bmod \boldsymbol{B}^{\text{prev}}) \in R$ (by (12)). Thus, it follows that after looping over all $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$, the final W satisfies (13), as desired.

Note that the sample complexity of Algorithm 6 is determined by the total number of samples required to construct the various $Z^{(\boldsymbol{\alpha},\boldsymbol{\beta})}$. For any fixed $\boldsymbol{\beta} \in S$, constructing $Z^{(\boldsymbol{\alpha},\boldsymbol{\beta})}_{\boldsymbol{j}}$ requires $|\boldsymbol{B}^{\text{base}}|$ samples from X. Since there are $|S|$ values of $\boldsymbol{\beta}$ that are relevant, it follows that the total sample complexity is

$$O\left(|S| \cdot \left|\boldsymbol{B}^{\text{base}}\right|\right) = O\left(m \log^2 m \cdot d \log n \cdot \left|\boldsymbol{B}^{\text{base}}\right|\right).$$

The time complexity of this procedure is due to two computations. First, constructing each $\widehat{Z}^{(\boldsymbol{\alpha},\boldsymbol{\beta})}$ for a fixed $\boldsymbol{\beta}$ takes time $O(\left|\boldsymbol{B}^{\text{base}}\right| \log_2 \left(|\boldsymbol{B}^{\text{base}}|\right) + \|\widehat{\chi}\|_0)$. Second, computing the $\boldsymbol{v}^{(\boldsymbol{b})}$ vector for each fixed $\boldsymbol{b} \in [\boldsymbol{B}^{\text{base}}]$ requires time $O(|S|^3)$. Therefore, the total time complexity is

$$O\left(|S|^3 \cdot \left|\boldsymbol{B}^{\text{base}}\right| + |S| \cdot \left(\left|\boldsymbol{B}^{\text{base}}\right| \log_2 \left(|\boldsymbol{B}^{\text{base}}|\right) + \|\widehat{\chi}\|_0\right)\right).$$

□

8.6 Resolving buckets in the hashed signal

The other major building block we need for developing a sparse FFT algorithm is a function for testing bucketings of signals with various shifts for emptyness and one-sparsity. Such a primitive takes in a list of buckets of a hashed signal and determines whether each bucket is empty or not. If a bucket is not empty, then we determine whether the bucket consists of exactly one frequency using a one-sparse test. If so, we can determine this frequency and the value of the signal at this frequency from the bucketed signals. If not, then we retain the bucket for the next iteration, in which we will hash to more buckets.

Algorithm 7 Procedure for testing a hashed signal

procedure TESTBUCKETS($\{W_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{A}\cup\{e_1,\ldots,e_d\}}, n, d, \boldsymbol{B}$) ▷ 1-sparse test and zero test
▷ $e_1,\ldots,e_d$ are standard basis vectors in $[n]^d$
  $\widehat{\chi} \leftarrow \{0\}^{n^d}$;
  $R \leftarrow \emptyset$
  for $\boldsymbol{b} \in \mathrm{Dom}\,(W_{\boldsymbol{\alpha}})$ **do** ▷ $\mathrm{Dom}\,(W_{\boldsymbol{\alpha}})$: set of all first coordinates in $W_{\boldsymbol{\alpha}}$
    if $\sum_{\boldsymbol{\alpha}\in\mathcal{A}} |W_{\boldsymbol{\alpha}}(\boldsymbol{b})|^2 > 0$ **then** ▷ Zero test on $U_x(\boldsymbol{B},\boldsymbol{b})$
      $f_q \leftarrow \frac{n}{2\pi}\cdot\phi\left(\frac{W_{e_q}(\boldsymbol{b})}{W_0(\boldsymbol{b})}\right)$ for every $q\in\{1,...,d\}$ ▷ $e_1,...,e_d$ are standard bases
      $v \leftarrow W_0(\boldsymbol{b})$
      if $\sum_{\boldsymbol{\alpha}\in\mathcal{A}} |ve^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{\alpha}}{n}} - W_{\boldsymbol{\alpha}}(\boldsymbol{b})|^2 = 0$ **then** ▷ One sparse test on $U_x(\boldsymbol{B},\boldsymbol{b})$
        $\widehat{\chi}_{\boldsymbol{f}} \leftarrow v$;
      else
        $R \leftarrow R\cup\{\boldsymbol{b}\}$;
  return $(\widehat{\chi}, R)$

Lemma 10. (TESTBUCKETS in high dimensions) *Suppose d and n are positive integers such that n is a power of two. Suppose $\boldsymbol{B} = (B_1, B_2, \ldots, B_d)$ is a vector of powers of two such that $B_j \mid n$ for all j. Suppose $x\in\mathbb{C}^{n^d}$ is a signal such that $W_{\boldsymbol{\alpha}}(\boldsymbol{b}) = U_x^{\boldsymbol{\alpha}}(\boldsymbol{B},\boldsymbol{b})$ is a $(\boldsymbol{B},\boldsymbol{b})$-bucketing of x with shift $\boldsymbol{\alpha}$ for all $\boldsymbol{\alpha}\in\mathcal{A}$ and $\boldsymbol{b}\in\mathrm{Dom}(W_{\boldsymbol{\alpha}})$ where $\mathcal{A}$ is a multiset of q i.i.d samples from* $\mathrm{Unif}([n]^d)$ *for some*

$$q = \Omega\left(\max_{\boldsymbol{b}\in[\boldsymbol{B}]} |S_x(\boldsymbol{B},\boldsymbol{b})|\cdot\log^2(\max_{\boldsymbol{b}\in[\boldsymbol{B}]} |S_x(\boldsymbol{B},\boldsymbol{b})|)\cdot(d\log n)\right),$$

and $S_x(\boldsymbol{B},\boldsymbol{b})$ for all $\boldsymbol{b}\in[\boldsymbol{B}]$ are Congruence classes of $\mathrm{supp}\ \widehat{x}$. *Also suppose that Algorithm 7 takes in the quantities $W_{\boldsymbol{\alpha}}(\boldsymbol{b})$ for all $\boldsymbol{\alpha}\in\{e_1,\ldots,e_d\}$, standard basis vectors in $[n]^d$. Then,* TESTBUCKETS$(\{W_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{A}\cup\{e_1,\ldots,e_d\}}, n, d, \boldsymbol{B})$ *returns $\widehat{\chi}$ and R that with probability $1-\frac{1}{n^{10d}}$ satisfy the following:*

- *For any $\boldsymbol{b}\in\mathrm{Dom}(W_{\boldsymbol{\alpha}})$ such that $S_x(\boldsymbol{B},\boldsymbol{b})$ is a singleton set, $S_x(\boldsymbol{B},\boldsymbol{b}) = \{\boldsymbol{f}\}$, we have $\widehat{\chi}_{\boldsymbol{f}} = \widehat{x}_{\boldsymbol{f}}$.*
- *We have $R = \{\boldsymbol{b}\in\mathrm{Dom}(W_{\boldsymbol{\alpha}}) : |S_x(\boldsymbol{B},\boldsymbol{b})|\geq 2\}$.*

Moreover, the runtime of this procedure is $O\left(|\mathcal{A}|\cdot|\mathrm{Dom}(W_{\boldsymbol{\alpha}})|\right)$.

Proof. Let F_N^{-1} be the d dimensional inverse Fourier transform's matrix with $N = n^d$ points. The matrix $M = \sqrt{N}F_N^{-1}$ is a unitary matrix and all of its elements have absolute value $\frac{1}{\sqrt{N}}$. If

you let $M_{\mathcal{A}}$ denote the submatrix of M whose rows are sampled from M according to set $\mathcal{A}$ then by Theorem 7, $\frac{\sqrt{N}}{|\mathcal{A}|}M_{\mathcal{A}}$ satisfies the restricted isometry property of order $\max_{\boldsymbol{b}\in[\boldsymbol{B}]}|S_x(\boldsymbol{B},\boldsymbol{b})|+1$ with probability $1-\frac{1}{N^{10}}$. In the rest we condition on the event corresponding to matrix $\frac{\sqrt{N}}{|\mathcal{A}|}M_{\mathcal{A}}$ satisfying RIP of order $\max_{\boldsymbol{b}\in[\boldsymbol{B}]}|S_x(\boldsymbol{B},\boldsymbol{b})|+1$.

Now, note that by definition of $U_x(\boldsymbol{B},\boldsymbol{b})$ and $S_x(\boldsymbol{B},\boldsymbol{b})$ (definitions 26 and 25 respectively) we have,

$$\begin{aligned}W_{\boldsymbol{\alpha}}(\boldsymbol{b}) &= U_x^{\boldsymbol{\alpha}}(\boldsymbol{B},\boldsymbol{b})\\ &= \sum_{\boldsymbol{f}\in S_x(\boldsymbol{B},\boldsymbol{b})}\widehat{x}_{\boldsymbol{f}}\cdot e^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{a}}{n}}\\ &= \left((NF_N^{-1})\cdot\widehat{x}_{S_x(\boldsymbol{B},\boldsymbol{b})}\right)_{\boldsymbol{\alpha}}\end{aligned}$$

therefore, for every $\boldsymbol{b}\in\mathrm{Dom}(W_{\boldsymbol{\alpha}})$ the following holds true,

$$\frac{|\mathcal{A}|}{2}\|\widehat{x}_{S_x(\boldsymbol{B},\boldsymbol{b})}\|_2^2\le\sum_{\boldsymbol{\alpha}\in\mathcal{A}}|W_{\boldsymbol{\alpha}}(\boldsymbol{b})|^2\le\frac{3|\mathcal{A}|}{2}\|\widehat{x}_{S_x(\boldsymbol{B},\boldsymbol{b})}\|_2^2$$

thus the zero test in line 5 of Algorithm 7 works correctly for all buckets.

Now note that if $S_x(\boldsymbol{B},\boldsymbol{b})$ is a singleton set $\{\boldsymbol{f}\}$ then,

$$\begin{aligned}W_{\boldsymbol{\alpha}}(\boldsymbol{b}) &= U_x^{\boldsymbol{\alpha}}(\boldsymbol{B},\boldsymbol{b})\\ &= \sum_{\boldsymbol{j}\in S_x(\boldsymbol{B},\boldsymbol{b})}\widehat{x}_{\boldsymbol{j}}\cdot e^{2\pi i\frac{\boldsymbol{j}^T\boldsymbol{a}}{n}}\\ &= \sum_{\boldsymbol{j}\in\{\boldsymbol{f}\}}\widehat{x}_{\boldsymbol{j}}\cdot e^{2\pi i\frac{\boldsymbol{j}^T\boldsymbol{a}}{n}}\\ &= x_{\boldsymbol{f}}\cdot e^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{a}}{n}}.\end{aligned}$$

Therefore, for every $q=1,2,\ldots,d$, the following holds,

$$\frac{W_{e_q}(\boldsymbol{b})}{W_0(\boldsymbol{b})}=e^{2\pi i\frac{\boldsymbol{f}^Te_q}{n}}=e^{2\pi i\frac{f_q}{n}}$$

thus, $\frac{n}{2\pi}\cdot\phi\left(\frac{W_{e_q}(\boldsymbol{b})}{W_0(\boldsymbol{b})}\right)=f_q$. Also note that, $W_0(\boldsymbol{b})=\widehat{x}_{\boldsymbol{f}}$. This is precisely implemented in line 6-7 of Algorithm 7. On the other hand if the hypothesis that $S_x(\boldsymbol{B},\boldsymbol{b})$ is a singleton set is incorrect our algorithm will find it. Using the notation $v=W_0(\boldsymbol{b})$ as in line 7 of Algorithm 7,

$$\begin{aligned}W_{\boldsymbol{\alpha}}(\boldsymbol{b})-ve^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{a}}{n}} &= \sum_{\boldsymbol{j}\in S_x(\boldsymbol{B},\boldsymbol{b})}\widehat{x}_{\boldsymbol{j}}\cdot e^{2\pi i\frac{\boldsymbol{j}^T\boldsymbol{a}}{n}}-ve^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{a}}{n}}\\ &= \sum_{\boldsymbol{j}\in S_x(\boldsymbol{B},\boldsymbol{b})\cup\{\boldsymbol{f}\}}\widehat{x}'_{\boldsymbol{j}}\cdot e^{2\pi i\frac{\boldsymbol{j}^T\boldsymbol{a}}{n}}\\ &= \left((NF_N^{-1})\cdot\widehat{x}'_{S_x(\boldsymbol{B},\boldsymbol{b})\cup\{\boldsymbol{f}\}}\right)_{\boldsymbol{\alpha}}\end{aligned}$$

where, $\widehat{x}'=\widehat{x}(\cdot)-v\delta_{\boldsymbol{f}}(\cdot)$. Because, $\widehat{x}'_{S_x(\boldsymbol{B},\boldsymbol{b})\cup\{\boldsymbol{f}\}}$ is at most $\max_{\boldsymbol{b}\in[\boldsymbol{B}]}|S_x(\boldsymbol{B},\boldsymbol{b})|+1$ sparse and matrix $\frac{\sqrt{N}}{|\mathcal{A}|}M_{\mathcal{A}}$ satisfies RIP of order $\max_{\boldsymbol{b}\in[\boldsymbol{B}]}|S_x(\boldsymbol{B},\boldsymbol{b})|+1$ we have that,

$$\frac{|\mathcal{A}|}{2}\|\widehat{x}'_{S_x(\boldsymbol{B},\boldsymbol{b})\cup\{\boldsymbol{f}\}}\|_2^2 \le \sum_{\boldsymbol{\alpha}\in\mathcal{A}} |W_{\boldsymbol{\alpha}}(\boldsymbol{b})ve^{2\pi i\frac{\boldsymbol{f}^T\boldsymbol{a}}{n}}|^2 \le \frac{3|\mathcal{A}|}{2}\|\widehat{x}_{S_x(\boldsymbol{B},\boldsymbol{b})\cup\{\boldsymbol{f}\}}\|_2^2$$

thus, the one sparse test in line 8 of Algorithm 7 works correctly for all buckets.

It is straightforward to see that the runtime of this procedure is $O\left(|\mathcal{A}|\cdot|\mathrm{Dom}(W_{\boldsymbol{\alpha}})|\right)$. □

8.7 Sparse FFT for signals with random support in nearly linear time

Algorithm 8 Procedure for Sparse FFT on random support signals with nearly linear sample complexity and runtime

1: **procedure** SPARSEFFT(x, n, d, k)
2: $\Gamma \leftarrow \Theta(1)$
3: $\boldsymbol{B}^{\text{base}} \leftarrow$ MAKEBUCKET($\Gamma k, n, d$)
4: $\boldsymbol{B}^{\text{prev}} \leftarrow \boldsymbol{B}^{\text{base}}$
5: $\boldsymbol{B}^{\text{next}} \leftarrow$ MAKEBUCKET($\Gamma^2 k, n, d$)
6: $\widehat{\chi}^0 \leftarrow \{0\}^{n^d}$
7: $R \leftarrow [\boldsymbol{B}^{\text{prev}}]$
8: $L \leftarrow \log_\Gamma k$
9: **for** $t = 0$ `to` L **do**
10: **if** $R = \emptyset$ **then**
11: **return** $\widehat{\chi}$
12: $\mathcal{A} \leftarrow \{\boldsymbol{\alpha}_j : \boldsymbol{\alpha}_j$ is an i.i.d. sample from Unif$([n]^d)$ for all $j \in [C\log^2 N\cdot(\log\log N)^2]\}$
13: $W_{\boldsymbol{\alpha}} \leftarrow$ HASHING$(x, \widehat{\chi}, n, d, \boldsymbol{B}^{\text{base}}, \boldsymbol{B}^{\text{prev}}, \boldsymbol{B}^{\text{next}}, \boldsymbol{\alpha}, R)$ for all $\boldsymbol{\alpha} \in \mathcal{A}\cup\{e_1, ..., e_d\}$
14: $(\widehat{\chi}', R) \leftarrow$ TESTBUCKETS $\left(\{W_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{A}\cup\{e_1,...,e_d\}}, n, d, \boldsymbol{B}^{\text{next}}\right)$
15: $\widehat{\chi} \leftarrow \widehat{\chi} + \widehat{\chi}'$
16: $\boldsymbol{B}^{\text{prev}} \leftarrow \boldsymbol{B}^{\text{next}}$
17: $\boldsymbol{B}^{\text{base}} \leftarrow$ MAKEBUCKET$(\Gamma^{-t}k, n, d)$
18: $\boldsymbol{B}^{\text{next}} \leftarrow$ MAKEBUCKET$(\Gamma^{t+3}k, n, d)$
19: **return** $\widehat{\chi}$
20: **procedure** MAKEBUCKET(k, n, d)
21: $p \leftarrow \lfloor\log_n k\rfloor$
22: $r \leftarrow \frac{k}{n^p}$
23: $B_1, ..., B_p \leftarrow n$
24: $B_{p+1} \leftarrow r$
25: $B_{p+2}, ..., B_d \leftarrow 1$
26: **return** $\boldsymbol{B}$

Now, we are ready to present the main theorem of this section.

Theorem 6 (Sparse FFT algorithm for random support signals)**.** *Suppose d is a positive integer and n and k are powers of two. For any signal $x \in \mathbb{C}^{n^d}$ such that x is a random support signal of Fourier sparsity k, the procedure* SPARSEFFT (x, n, d, k) *(see Algorithm 8) returns $\widehat{x}$ with probability $9/10$. Moreover, the runtime and sample complexity of this procedure are $\widetilde{O}(k)$.*

The theorem is a consequence of the following lemma.

Lemma 11. *Let $\boldsymbol{B}^{\text{base},(t)}$, $\boldsymbol{B}^{\text{prev},(t)}$, $\boldsymbol{B}^{\text{next},(t)}$, $R^{(t)}$, and $\chi^{(t)}$ denote the values of $\boldsymbol{B}^{\text{base}}$, $\boldsymbol{B}^{\text{prev}}$, $\boldsymbol{B}^{\text{next}}$, R, and χ, respectively, at the start of iteration t of the main* for *loop in Algorithm 8. Then, for all $t = 0, 1, \ldots, L$, we define the event $\mathcal{E}_t$ to be the occurrence of the following statements:*

1. $R^{(t)} = \{\boldsymbol{b} \in [\boldsymbol{B}^{\text{prev},(t)}] : S_{x-\chi^{(t)}}(\boldsymbol{B}^{\text{prev},(t)}, \boldsymbol{b}) \neq \emptyset\}$.
2. $\text{supp } (\widehat{x - \chi^{(t)}}) \subseteq \text{supp } \widehat{x}$ *and* $\text{supp } \widehat{\chi}^{(t)} \cap \text{supp } (\widehat{x - \chi^{(t)}}) = \emptyset$.
3. *If $t > 0$ then* $\left|S_{x-\chi^{(t)}}\left(\boldsymbol{B}^{\text{prev},(t)}, \boldsymbol{\xi} \pmod{\boldsymbol{B}^{\text{prev},(t)}}\right)\right| \geq 2$ *for every* $\boldsymbol{\xi} \in \text{supp } (\widehat{X} - \widehat{\chi}^{(t)})$.

Then, $\mathcal{E}_0$ holds with probability 1, while $\Pr[\mathcal{E}_t \mid \mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_{t-1}] \geq 1 - \frac{1}{n^{2d}}$ *for $t = 1, \ldots, L$.*

Proof. Note that for $t = 0$, we have $R^{(t)} = \left[\boldsymbol{B}^{\text{prev},(0)}\right]$. Thus, condition (1) trivially holds. Condition (2) also trivially holds, since $\widehat{\chi}^{(0)} = 0$. Furthermore, (3) does not apply for event $\mathcal{E}_0$. Thus, $\mathcal{E}_0$ holds with probability 1.

Now, assume that $\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_m$ hold for some $m \geq 0$. We consider the probability of $\mathcal{E}_{m+1}$ occurring, conditioned on the aforementioned events. Note that it follows from the values that Algorithm 8 assigns to $\boldsymbol{B}^{\text{base},(m)}$, $\boldsymbol{B}^{\text{prev},(m)}$ and $\boldsymbol{B}^{\text{next},(m)}$ along with condition (1) of the inductive hypothesis that Lemma 9 can be applied to invocations of HASHING $(x, \widehat{\chi}^{(m)}, n, d, \boldsymbol{B}^{\text{base},(m)}, \boldsymbol{B}^{\text{prev},(m)}, \boldsymbol{B}^{\text{next},(m)}, \boldsymbol{\alpha}, R)$ in line 13 of Algorithm 8, and hence the output of HASHING procedure satisfies the following,

$$W_{\boldsymbol{\alpha}} = \left\{(\boldsymbol{b}', U^{\boldsymbol{\alpha}}_{x-\chi^{(m)}}(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b}')) : \boldsymbol{b}' \in [\boldsymbol{B}^{\text{next},(m)}] \text{ s.t. } \boldsymbol{b}' \equiv \boldsymbol{b} \pmod{\boldsymbol{B}^{\text{prev},(m)}} \text{ for some } \boldsymbol{b} \in R^{(m)}\right\}. \tag{21}$$

Moreover, it is clear that for every $\boldsymbol{\alpha} \in [n]^d$ and every $\boldsymbol{b}' \in [\boldsymbol{B}^{\text{next},(m)}]$, one has that $W_{\boldsymbol{\alpha}}(\boldsymbol{b}')$ is $(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b}')$-bucketing of $x - \chi^{(m)}$.

Now, note that by condition (2) of the inductive hypothesis along with definition of Congruence class presented in Definition 25, it follows that $S_{x-\chi^{(m)}}(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b}) \subseteq S_x(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b})$ for all $\boldsymbol{b} \in [\boldsymbol{B}^{\text{next},(m)}]$. Hence by Lemma 12, with probability $1 - \frac{1}{n^{3d}}$,

$$S_{x-\chi^{(m)}}(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b}) = O(d \log_2 n)$$

This show that set $\mathcal{A}$ defined in line 12 of Algorithm 8 satisfies the precondition of Lemma 10. Therefore by Lemma 10, a call to TESTBUCKETS procedure in line 14 of Algorithm 8, with probability $1 - \frac{1}{n^{10d}}$, outputs $(\widehat{\chi}', R^{(m+1)})$ such that the following hold:

(a) For any $\boldsymbol{b}' \in \text{Dom}(W_{\boldsymbol{\alpha}})$ such that $S_{x-\chi^{(m)}}(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b}')$ is a singleton set, $S_{x-\chi^{(m)}}(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b}') = \{\boldsymbol{f}\}$, one has $\widehat{\chi}'_{\boldsymbol{f}} = (\widehat{x - \chi^{(m)}})_{\boldsymbol{f}}$.

(b) $R^{(m+1)} = \left\{\boldsymbol{b}' \in \text{Dom}(W_{\boldsymbol{\alpha}}) : |S_{x-\chi^{(m)}}(\boldsymbol{B}^{\text{next},(m)}, \boldsymbol{b}')| \geq 2\right\}$.

In order to complete the inductive step, it suffices to show that (a) and (b) imply conditions (1), (2), and (3) in the definition of $\mathcal{E}_t$ for $t = m + 1$.

Observe that condition (a) imply that $\text{supp } \widehat{\chi}' \subseteq \text{supp } (\widehat{x - \chi^{(m)}})$ therefore since $\chi^{(m+1)} = \chi^{(m)} + \chi'$ (by line 15), it follows that $\text{supp } (\widehat{x - \chi^{(m+1)}}) \subseteq \text{supp } (\widehat{x - \chi^{(m)}})$. Hence, by inductive hypothesis $\mathcal{E}_m$ we have $\text{supp } (\widehat{x - \chi^{(m+1)}}) \subseteq \text{supp } \widehat{x}$. Also note that for every $\boldsymbol{f} \in$

supp $(\widehat{x - \chi^{(m+1)}})$ we have that $\boldsymbol{f} \in$ supp $(\widehat{x - \chi^{(m)}})$ and hence by condition (2) of the inductive hypothesis $\mathcal{E}_m$, one has $\widehat{\chi}^{(m)}_{\boldsymbol{f}} = 0$. Condition (a) implies that $\widehat{\chi}'_{\boldsymbol{f}} = 0$ for every $\boldsymbol{f} \in$ supp $(\widehat{x - \chi^{(m+1)}})$ and hence $\chi^{(m+1)}_{\boldsymbol{f}} = \chi^{(m)}_{\boldsymbol{f}} + \chi'_{\boldsymbol{f}} = 0$ for every such every $\boldsymbol{f}$. This establishes condition (2) of $\mathcal{E}_t$ for $t = m + 1$.

Next note that $\boldsymbol{B}^{\mathrm{prev},(m+1)} = \boldsymbol{B}^{\mathrm{next},(m)}$ (by line 16). Conditions (a) along with condition (1) of the inductive hypothesis for $\mathcal{E}_m$ and (21) imply that there exists no $\boldsymbol{b'} \in [\boldsymbol{B}^{\mathrm{prev},(m+1)}]$ such that $|S_{x-\chi^{(m+1)}}(\boldsymbol{B}^{\mathrm{prev},(m+1)}, \boldsymbol{b'})| = 1$. Also note that condition (b) implies that $|S_{x-\chi^{(m+1)}}(\boldsymbol{B}^{\mathrm{prev},(m+1)}, \boldsymbol{b'})| \geq 2$ for every $\boldsymbol{b'} \in R^{(m+1)}$, therefore $R^{(m+1)}$ satisfies condition (1) of the induction $\mathcal{E}_t$ for $t = m+1$. This also establishes condition (3) of $\mathcal{E}_t$ for $t = m + 1$.

By a union bound we have that with probability $1 - \frac{1}{n^{3d}} - \frac{1}{n^{10d}} \geq 1 - \frac{1}{n^{2d}}$, event $\mathcal{E}_{m+1}$ holds true as desired. □

Now we are ready to prove Theorem 6.

Proof of Theorem 6. Note that by Lemma 11, there exist events $\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_L$ such that $\Pr[\mathcal{E}_0] = 1$ and $\Pr[\mathcal{E}_t \mid \mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_{t-1}] \geq 1 - \frac{1}{n^{2d}}$ for $t = 1, 2, \ldots, L$. Observe that

$$\begin{aligned}
\Pr[\mathcal{E}_L] &\geq \Pr[\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_L] \\
&\geq 1 - \sum_{t=0}^{L} \Pr[\mathcal{E}_0, \ldots, \mathcal{E}_{t-1}] \cdot \Pr[\overline{\mathcal{E}_t} \mid \mathcal{E}_0, \ldots, \mathcal{E}_{t-1}] \\
&\geq 1 - \sum_{t=0}^{L} \Pr[\overline{\mathcal{E}_t} \mid \mathcal{E}_0, \ldots, \mathcal{E}_{t-1}] \\
&\geq 1 - \sum_{t=1}^{L} \frac{1}{n^{2d}} \\
&\geq 1 - \frac{L}{n^{2d}}.
\end{aligned}$$

Note that condition (3) of $\mathcal{E}_L$ implies that the existence of a $\boldsymbol{\xi} \in [n]^d$ such that $\widehat{\chi}^{(L)}_{\boldsymbol{\xi}} \neq \widehat{x}_{\boldsymbol{\xi}}$ requires $S^{(\boldsymbol{B}^{\mathrm{prev},(L)})} \neq \emptyset$. Now, recall that after the main *for* loop in Algorithm 8 finishes execution,we have

$$|\boldsymbol{B}^{\mathrm{prev}}| = k \cdot \Gamma^{L+2}.$$

Thus, by Lemma 14, we have that $\mathbb{E}\left[S^{(\boldsymbol{B}^{\mathrm{prev}})}\right] \leq \frac{k^2}{k \cdot \Gamma^{L+2}} = \frac{k}{\Gamma^{L+2}} \leq \frac{1}{100}$, by our choice of Γ and L. Thus, by Markov's inequality, with probability $\geq \frac{99}{100}$ over the randomness in the choice of $S =$ supp $\widehat{x}$, we have that $S^{(\boldsymbol{B}^{\mathrm{prev}})} = \emptyset$. Hence, by a union bound, we have that $\Pr\left[\mathcal{E}_L \wedge \left(S^{(\boldsymbol{B}^{\mathrm{prev}})} = \emptyset\right)\right] \geq \frac{9}{10}$. Thus, by condition (3) in Lemma 11, we see that with probability $\geq \frac{9}{10}$, the output $\widehat{\chi}$ of Algorithm 8 satisfies $\widehat{\chi}_{\boldsymbol{f}} = \widehat{x}_{\boldsymbol{f}}$ for all $\boldsymbol{f} \in [n]^d$, which proves the correctness of Algorithm 8.

Now, let us compute the sample complexity of Algorithm 8. Note that for each iteration t of the main *for* loop in SparseFFT, we have $\frac{|\boldsymbol{B}^{\mathrm{next}}|}{|\boldsymbol{B}^{\mathrm{prev}}|} = \Gamma$. Also, By condition (1) of $\mathcal{E}_t$ in Lemma 11,

$$R^{(t)} = \{\boldsymbol{b} \in [\boldsymbol{B}^{\mathrm{prev},(t)}] : S_{x-\chi^{(t)}}(\boldsymbol{B}^{\mathrm{prev},(t)}, \boldsymbol{b}) \neq \emptyset\}$$

Therefore, since $|\boldsymbol{B}^{\text{prev}}| \cdot |\boldsymbol{B}^{\text{base}}| \geq k^2$, by Lemma 8, we have that

$$\begin{aligned}
&\max_{\boldsymbol{b} \in \boldsymbol{B}^{\text{base},(t)}} \left| \left\{ \boldsymbol{r} \in R^{(t)} : \boldsymbol{r} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}^{\text{base},(t)}} \right\} \right| \\
&\quad = \max_{\boldsymbol{b} \in \boldsymbol{B}^{\text{base},(t)}} \left| S^{(\boldsymbol{B})} \cap \{ \boldsymbol{f} \in [\boldsymbol{B}] : \boldsymbol{f} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}'} \} \right| \\
&\quad = O(\log N)
\end{aligned}$$

with probability $\geq 1 - \frac{1}{N^3}$.

Moreover, $|\boldsymbol{B}^{\text{base},(t)}| = O\left(\frac{k}{\Gamma^t}\right)$. Therefore, by Lemma 9, each call to HASHING in the t-th iteration has sample complexity

$$O\left(\Gamma \frac{k}{\Gamma^t} (\log^2 N)(\log\log N)^2\right).$$

Hence, because in each iteration HASHING is invoked $O((\log^2 N)(\log\log N)^2)$ times, the total sample complexity for the algorithm is

$$O\left(\sum_{t=0}^{L-1} \Gamma \cdot \frac{k}{\Gamma^t} \log^4 N (\log\log N)^4\right) = O\left(k(\log^4 N)(\log\log N)^4\right),$$

since $\Gamma = O(1)$.

Finally, we compute the time complexity of Algorithm 8. By Lemma 9, we have that the time complexity for each call to HASHING in the t-th iteration of the main *for* loop is

$$O\left(\left(\Gamma(\log^2 N)(\log\log N)^2\right)^3 \cdot \frac{k}{\Gamma^t} + \left(\Gamma(\log^2 N)(\log\log N)^2\right) \cdot \left(\frac{k}{\Gamma^t} \cdot \log k + k\right)\right).$$

Thus, the total time complexity due to calls to HASHING is

$$O\left(\log^2 N (\log\log N)^2 \sum_{t=0}^{L-1} \left(\left(\Gamma(\log^2 N)(\log\log N)^2\right)^3 \cdot \frac{k}{\Gamma^t} + \left(\Gamma(\log^2 N)(\log\log N)^2\right) \cdot \left(\frac{k}{\Gamma^t} \log k + k\right)\right)\right),$$

which can be simplified as

$$O\left(k(\log^8 N)(\log\log N)^8\right), \tag{22}$$

since $\Gamma = O(1)$. Moreover, the call to TESTBUCKET in the t-th iteration of the main *for* loop has time complexity

$$O\left(|\mathcal{A}| \cdot \max_{\boldsymbol{\alpha} \in \mathcal{A}} |\text{Dom}(W_{\boldsymbol{\alpha}})|\right) = O\left(\frac{k}{\Gamma^t} \log^2 N \cdot (\log\log N)^2\right).$$

Hence, the total time complexity due to calls to TESTBUCKET is

$$O\left(\sum_{t=0}^{L-1} \frac{k}{\Gamma^t} \log^2 N \cdot (\log\log N)^2\right) = O\left(k \log^2 N \cdot (\log\log N)^2\right). \tag{23}$$

Therefore, by (22) and (23), the total time complexity of Algorithm 8 is

$$O\left(k(\log^8 N)(\log\log N)^8\right),$$

as desired. □

Acknowledgements

Michael Kapralov is supported in part by ERC Starting Grant 759471.

References

[AGS03] A. Akavia, S. Goldwasser, and S. Safra. Proving hard-core predicates using list decoding. *FOCS*, 44:146–159, 2003.

[Aka10] A. Akavia. Deterministic sparse Fourier approximation via fooling arithmetic progressions. *COLT*, pages 381–393, 2010.

[BCG+12] P. Boufounos, V. Cevher, A. C. Gilbert, Y. Li, and M. J. Strauss. What's the frequency, kenneth?: Sublinear fourier sampling off the grid. *RANDOM/APPROX*, 2012.

[BG] R. Beatson and Greengard. A short course on fast multipole methods. *https://web.njit.edu/~jiang/math614/beatson-greengard.pdf*.

[BG97] Rick Beatson and Leslie Greengard. A short course on fast multipole methods. *Wavelets, multilevel methods and elliptic PDEs*, 1:1–37, 1997.

[Bou14] J. Bourgain. An improved estimate in the restricted isometry problem. *GAFA*, 2014.

[CGV12] Mahdi Cheraghchi, Venkatesan Guruswami, and Ameya Velingker. Restricted isometry of fourier matrices and list decodability of random linear codes. *SODA*, 2012.

[CI17] Mahdi Cheraghchi and Piotr Indyk. Nearly optimal deterministic algorithm for sparse walsh-hadamard transform. *ACM Trans. Algorithms*, 13(3):34:1–34:36, 2017.

[Cip00] B. A. Cipra. The Best of the 20th Century: Editors Name Top 10 Algorithms. *SIAM News*, 33, 2000.

[CKPS16] Xue Chen, Daniel M. Kane, Eric Price, and Zhao Song. Fourier-sparse interpolation without a frequency gap. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 741–750, 2016.

[CKSZ17] Volkan Cevher, Michael Kapralov, Jonathan Scarlett, and Amir Zandieh. An adaptive sublinear-time block sparse Fourier transform, `https://arxiv.org/abs/1702.01286`. In *STOC*, 2017.

[CT06] E. Candes and T. Tao. Near optimal signal recovery from random projections: Universal encoding strategies. *IEEE Trans. on Info.Theory*, 2006.

[GGI+02] A. Gilbert, S. Guha, P. Indyk, M. Muthukrishnan, and M. Strauss. Near-optimal sparse Fourier representations via sampling. *STOC*, 2002.

[GHI+13] Badih Ghazi, Haitham Hassanieh, Piotr Indyk, Dina Katabi, Eric Price, and Lixin Shi. Sample-optimal average-case sparse fourier transform in two dimensions. *arXiv preprint arXiv:1303.1209*, 2013.

[GL89] O. Goldreich and L. Levin. A hard-corepredicate for allone-way functions. *STOC*, pages 25–32, 1989.

[GMS05] A. Gilbert, M. Muthukrishnan, and M. Strauss. Improved time bounds for near-optimal space Fourier representations. *SPIE Conference, Wavelets*, 2005.

[GR87a] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, December 1987.

[GR87b] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[HAKI12] H. Hassanieh, F. Adib, D. Katabi, and P. Indyk. Faster gps via the sparse fourier transform. *MOBICOM*, 2012.

[HIKP12a] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Near-optimal algorithm for sparse Fourier transform. *STOC*, 2012.

[HIKP12b] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Simple and practical algorithm for sparse Fourier transform. *SODA*, 2012.

[HKPV13] Sabine Heider, Stefan Kunis, Daniel Potts, and Michael Veit. A sparse prony fft. *SAMPTA*, 2013.

[HR17] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled fourier matrices. In *Geometric Aspects of Functional Analysis*, pages 163–179. Springer, 2017.

[IK14] Piotr Indyk and Michael Kapralov. Sample-optimal fourier sampling in any fixed dimension. *FOCS*, 2014.

[IKP14] Piotr Indyk, Michael Kapralov, and Eric Price. (Nearly) sample-optimal sparse fourier transform. *SODA*, 2014.

[Iwe10a] M. A. Iwen. Combinatorial sublinear-time Fourier algorithms. *Foundations of Computational Mathematics*, 10:303–338, 2010.

[Iwe10b] Mark A. Iwen. Combinatorial sublinear-time fourier algorithms. *Foundations of Computational Mathematics*, 10(3):303–338, 2010.

[Iwe12] M.A. Iwen. Improved approximation guarantees for sublinear-time Fourier algorithms. *Applied And Computational Harmonic Analysis*, 2012.

[Kap16] Michael Kapralov. Sparse fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time (available as an arxiv report at `http://arxiv.org/abs/1604.00845`). *STOC*, 2016.

[Kap17] Michael Kapralov. Sample efficient estimation and recovery in sparse FFT via isolation on average. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 651–662, 2017.

[KM91] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *STOC*, 1991.

[LWC12] D. Lawlor, Y. Wang, and A. Christlieb. Adaptive sub-linear time fourier algorithms. *arXiv:1207.6368*, 2012.

[Man92] Y. Mansour. Randomized interpolation and approximation of sparse polynomials. *ICALP*, 1992.

[MZIC17] Sami Merhi, Ruochuan Zhang, Mark A Iwen, and Andrew Christlieb. A new class of fully discrete sparse fourier transforms: Faster stable implementations with guarantees. *Journal of Fourier Analysis and Applications*, pages 1–34, 2017.

[PR13] Sameer Pawar and Kannan Ramchandran. Computing a k-sparse n-length discrete fourier transform using at most 4k samples and o (k log k) complexity. *ISIT*, 2013.

[PS15] Eric Price and Zhao Song. A robust sparse fourier transform in the continuous setting. *FOCS*, 2015.

[RV08] M. Rudelson and R. Vershynin. On sparse reconstruction from Fourier and Gaussian measurements. *CPAM*, 61(8):1025–1171, 2008.

A Proofs and pseudocode omitted from section 4

Proof of Lemma 4: Let v be a leaf of T, let $l = l_T(v)$ denote the level of v, let r denote the root of T, and let $v_0, v_1, \ldots, v_l$ denote the path from root to v in T, where $v_0 = r$ and $v_l = v$. Let q^* denote the smallest positive integer such that $l \leq q^* \cdot \log_2 n$. Note that $q^* \leq d$.

For $q \in \{0, 1, \ldots, d\}$ let $T^{(q)}$ be a subtree of T_N^{full} which denotes the result of truncating T to contain only the nodes that are at distance at most $q \log_2 n$ from the root.

We construct the (v, T)-isolating filter $\widehat{G}$ iteratively by starting with $\widehat{G}^{(0)} = 1$ and refining $\widehat{G}^{(q-1)}$ to $\widehat{G}^{(q)}$ over q^* steps. The filters $\widehat{G}^{(q)}$ will be $(v_{q \cdot \log_2 n}, T^{(q)})$-isolating for $q = 0, 1, \ldots, q^* - 1$ and $\widehat{G}^{(q^*)}$ will be $(v_l, T^{(q^*)})$-isolating. Since $T^{(q^*)} = T$ and $v_l = v$, the filter $\widehat{G}^{(q^*)}$ will be (v, T)-isolating, as required.

For every $q \in \{1, ..., q^*\}$ let T_q^v be the subtree of T which is rooted at $v_{(q-1) \cdot \log_2 n}$ and is restricted to contain only the nodes that are at distance at most $\log_2 n$ from $v_{(q-1) \cdot \log_2 n}$. For every node $u \in T_q^v$ the label of u is defined to be $f_u = (\boldsymbol{f}_u)_q$, i.e., the qth coordinate of $\boldsymbol{f}_u$, where $\boldsymbol{f}_u$ is the label of node u in tree T.

We now define $\widehat{G}^{(q)}$ for $q = 1, \ldots, q^*$. We start by letting $\widehat{G}^{(0)} = 1$ and letting for every $\boldsymbol{f} = (f_1, \ldots, f_q, \ldots, f_d) \in [n]^d$

$$\widehat{G}^{(q)}(\boldsymbol{f}) = \widehat{G}^{(q-1)}(\boldsymbol{f}) \cdot \widehat{G}_q(f_q). \tag{24}$$

where $\widehat{G}_q$ is a $(v_{q \cdot \log_2 n}, T_q^v)$-isolating filter for all $q = 1, ..., q^* - 1$ and $\widehat{G}_{q^*}$ is a $(v_l, T_{q^*}^v)$-isolating filter. By lemma 3, for every $q = 1, \ldots, q^*$ there exists such G_q with $|\text{supp } G_q| = 2^{w_{T_q^v}(v_{q \cdot \log_2 n})}$ and can be constructed in time $O(2^{w_{T_q^v}(v_{q \cdot \log_2 n})} + \log_2 n)$. Such a filter can be computed in Fourier domain at any desired frequency in time $O(\log_2 n)$. Note that $\widehat{G}^{(q)}$ is a tensor product of q filters in dimension one. We now show by induction on q that $\widehat{G}^{(q)}$ is a $(v_{q \cdot \log_2 n}, T^{(q)})$-isolating filter.

Algorithm 9 Splitting tree construction in time $O(|S| \log n)$

procedure TREE(S, n)
  $\mathcal{C}_0 \leftarrow \{(r, S)\}$
  Let T be a tree with one node, labeled $f_r = 0$
  for $j = 1$ `to` $\log_2 n$ **do**
    $\mathcal{C}_j \leftarrow \emptyset$
    for all $(v, S_v) \in \mathcal{C}_{j-1}$ **do**▷ $\mathcal{C}_{j-1}$ contains nodes at level $j-1$ and their corresponding set of frequencies
      $R \leftarrow \{g \in S_v : g = f_v \pmod{2^j}\}$
      $L \leftarrow \{g \in S_v : g = f_v + 2^{j-1} \pmod{2^j}\}$
      if $R \neq \emptyset$ **then**
        Add u as right child of v in T
        $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{(f_v, R)\}$
      if $L \neq \emptyset$ **then**
        Add w as left child of u in T
        $\mathcal{C}_j \leftarrow \mathcal{C}_j \cup \{(f_v + 2^{j-1}, L)\}$
  return T
procedure TREE.REMOVE(T, v)
  $r \leftarrow$ root of T, $l \leftarrow l_T(v)$
  $v_0, v_1, \ldots, v_l \leftarrow$ path from r to v in T, where $v_0 = r$ and $v_l = v$
  $q \leftarrow$ largest integer $j \leq l$ such that v_j has two children
  Remove $v_{q+1}, ..., v_l$ and their connecting edges from T
  return T

The **base** of the induction is provided by $q = 0$: since v_0 is the root of $T^{(0)}$, we have that $\text{FrequencyCone}_{T^{(0)}}(v_0) = [n]^d$ and $\widehat{G}^{(0)} \equiv 1$ as required.

We now prove the **inductive step**: $q-1 \to q$. We first show that $\widehat{G}^{(q)}_{\boldsymbol{f}'} = 0$ for every $\boldsymbol{f}' \in \bigcup_{\substack{u \neq v_{q\cdot\log_2 n} \\ u\text{: leaf of } T^{(q)}}} \text{FrequencyCone}_{T^{(q)}}(u)$. Let u be a leaf of $T^{(q)}$ distinct from $v_{q\cdot\log_2 n}$. Let u' denote the leaf of $T^{(q-1)}$ which is the ancestor of u. We consider two cases.

Case 1: $\boldsymbol{f}' \notin \text{FrequencyCone}_{T^{(q-1)}}(v_{(q-1)\log_2 n})$ Suppose that $u' \neq v_{(q-1)\cdot\log_2 n}$. Note that $l_T(u') \leq (q-1)\log_2 n$, and also note that

$$\text{FrequencyCone}_{T^{(q)}}(u) \subseteq \text{FrequencyCone}_{T^{(q-1)}}(u'),$$

Thus for every $\boldsymbol{f}' \in \text{FrequencyCone}_{T^{(q)}}(u)$ it is true that $\boldsymbol{f}' \in \text{FrequencyCone}_{T^{q-1}}(u')$. By the inductive hypothesis we have that $\widehat{G}^{(q-1)}$ is $(v_{(q-1)\log_2 n}, T^{(q-1)})$-isolating, and hence by the assumption of $u' \neq v_{(q-1)\cdot\log_2 n}$, one has $\widehat{G}^{(q-1)}(\boldsymbol{f}') = 0$ for every such $\boldsymbol{f}'$, and thus $\widehat{G}^{(q)}(\boldsymbol{f}') = \widehat{G}^{(q-1)}(\boldsymbol{f}') \cdot \widehat{G}_q(f'_q) = 0$ as required.

Case 2: $\boldsymbol{f}' \in \text{FrequencyCone}_{T^{(q-1)}}(v_{(q-1)\log_2 n})$ Suppose that $v_{(q-1)\cdot\log_2 n}$ is ancestor of u. Therefore, by definition of T^v_q, one can see that u is a leaf in T^v_q. Hence, by definition of T^v_q, for every $\boldsymbol{f}' \in \text{FrequencyCone}_{T^{(q)}}(u)$, it is true that $f'_q \in \text{FrequencyCone}_{T^v_q}(u)$. Recall that $\widehat{G}_q$ is a $(v_{q\cdot\log_2 n}, T^v_q)$-isolating filter and therefore, $\widehat{G}_q(f'_q) = 0$, and thus $\widehat{G}^{(q)}(\boldsymbol{f}') = \widehat{G}^{(q-1)}(\boldsymbol{f}') \cdot \widehat{G}_q(f'_q) = 0$ as required.

Now we show that $\widehat{G}^{(q)}_{\boldsymbol{f}} = 1$ for all $\boldsymbol{f} \in \text{FrequencyCone}_{T^{(q)}}(v_{q\cdot\log_2 n})$. Note that $v_{q\cdot\log_2 n}$ is a leaf in T^v_q. Hence, for every $\boldsymbol{f} \in \text{FrequencyCone}_{T^{(q)}}(v_{q\cdot\log_2 n})$, it is true that $f_q \in \text{FrequencyCone}_{T^v_q}(v_{q\cdot\log_2 n})$. Since $\widehat{G}_q$ is a $(v_{q\cdot\log_2 n}, T^v_q)$-isolating filter, $\widehat{G}_q(f_q) = 1$. Now, note that

$$\text{FrequencyCone}_{T^{(q)}}(v_{q\cdot\log_2 n}) \subseteq \text{FrequencyCone}_{T^{(q-1)}}(v_{(q-1)\cdot\log_2 n}),$$

Thus for every $\boldsymbol{f} \in \text{FrequencyCone}_{T^{(q)}}(v_{q\cdot\log_2 n})$ it is true that $\boldsymbol{f} \in \text{FrequencyCone}_{T^{(q-1)}}(v_{(q-1)\cdot\log_2 n})$. By the inductive hypothesis we have that $\widehat{G}^{(q-1)}$ is $(v_{(q-1)\log_2 n}, T^{(q-1)})$-isolating, and hence $\widehat{G}^{(q-1)}(\boldsymbol{f}) = 1$, and thus $\widehat{G}^{(q)}(\boldsymbol{f}) = \widehat{G}^{(q-1)}(\boldsymbol{f}) \cdot \widehat{G}_q(f_q) = 1$ as required.

It remains to note that $w_T(v) = \sum_{q=1}^{q^*} w_{T^v_q}(v_{q\cdot\log_2 n})$. By Lemma 3, for every $q \in \{1, ..., q^*\}$ one has $|\text{supp } G_q| = 2^{w_{T^v_q}(v_{q\cdot\log_2 n})}$, so $|\text{supp } G| = 2^{w_T(v)}$, as required (note that the support size of the convolution of two filters is at most the product of support sizes of each filter).

The total runtime for constructing this filter has two parts; First part is the computation time of G_q's for all $q \in \{1, ..., q^*\}$ which takes $\sum_{q=1}^{q^*} O\left(2^{w_{T^v_q}(v_{q\cdot\log_2 n})} + \log_2 n\right) = O\left(2^{w_T(v)} + d\log_2 n\right)$ by Lemma 3. Second part is the time needed for computing the tensor product of all G_q's which is $O\left(\|G_1\|_0 \cdot ... \cdot \|G_{q^*}\|_0\right) = O(2^{w_T(v)})$. Therefore the total runtime is $O\left(2^{w_T(v)} + d\log_2 n\right)$. Moreover, the total time for computing $\widehat{G}(\boldsymbol{\xi})$ is the sum of the times needed for computing all $\widehat{G}_q(\xi_q)$'s for $q = 1, \cdots, q^*$, which is $O(d\log_2 n)$ by Lemma 3.

□

Proof of Lemma 6: Let $N = n^d$. Recall that for every $\boldsymbol{t} \in [n]^d$,

$$x_{\boldsymbol{t}} = \frac{1}{N} \sum_{\boldsymbol{f} \in [n]^d} \widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{t}}{n}}$$

Because all $\widehat{x}_{\boldsymbol{f}}$'s are zero mean independent random variables, for every fixed $\boldsymbol{t} \in [n]^d$ one has that for every $\boldsymbol{f} \in [n]^d$ the random variables $\widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{t}}{n}}$ are zero mean and independent. Observe that for all $\boldsymbol{f} \in [n]^d$, we have $|\widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{t}}{n}}| = |\beta_{\boldsymbol{f}}| \leq \|\beta\|_\infty$ and also $\mathbb{E}\left[|\widehat{x}_{\boldsymbol{f}} \cdot e^{2\pi i \frac{\boldsymbol{f}^T \boldsymbol{t}}{n}}|^2\right] = |\beta_{\boldsymbol{f}}|^2$. Therefore by Bernstein's inequality we have that for every fixed $\boldsymbol{t} \in [n]^d$,

$$\begin{aligned}\Pr\left[|x_{\boldsymbol{t}}| > \frac{\theta}{N}\right] &\leq 2\exp\left(-\frac{\frac{1}{2}\theta^2}{\|\beta\|_2^2 + \frac{1}{3}\|\beta\|_\infty \cdot \theta}\right) \\ &\leq 2\exp\left(-\frac{\frac{1}{2}\theta^2}{\|\beta\|_2^2 + \frac{1}{3}\|\beta\|_2 \cdot \theta}\right)\end{aligned}$$

If we choose $\theta = C_1 \log_2 N \cdot \|\beta\|_2$ for some absolute constant $C_1 > 0$,

$$\begin{aligned}\Pr\left[|x_{\boldsymbol{t}}| > \frac{C_1 \log_2 N \cdot \|\beta\|_2}{N}\right] &\leq 2\exp\left(-\frac{\frac{1}{2}C_1^2 \log_2^2 N}{1 + \frac{1}{3}C_1 \log_2 N}\right) \\ &\leq \frac{1}{2N^5}\end{aligned}$$

for large enough constant C_1. By a union bound over all $\boldsymbol{t} \in [n]^d$ we get that, $|x_{\boldsymbol{t}}|^2 \leq \frac{C_1^2 \log_2^2 N}{N^2}\|\beta\|_2^2$ for all $\boldsymbol{t} \in [n]^d$ with probability $1 - \frac{1}{2N^4}$.

Now note that by Parseval's theorem, Claim 1,

$$\sum_{\boldsymbol{j} \in [n]^d} |x_{\boldsymbol{j}}|^2 = \frac{1}{N} \sum_{\boldsymbol{f} \in [n]^d} |\beta_{\boldsymbol{f}}|^2.$$

Conditioning on $\|x\|_\infty \leq \frac{C_1^2 \log_2^2 N}{N^2}\|\beta\|_2^2$, by Chernoff-Hoeffding Bound we have,

$$\begin{aligned}\Pr\left[\frac{1}{2} \cdot \frac{\|\beta\|_2^2}{N^2} \leq \frac{1}{s}\sum_{j=1}^{s} |x_{\boldsymbol{t}_j}|^2 \leq \frac{3}{2} \cdot \frac{\|\beta\|_2^2}{N^2}\right] &\geq 1 - 2e^{-\frac{C_2 \cdot s \cdot \|\beta\|_2^2}{N^2 \|x\|_\infty}} \\ &\geq 1 - 2e^{-\frac{C_2 \cdot s}{C_1^2 \log_2^2 N}}\end{aligned}$$

where the probability is over the i.i.d. random variables $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_s \sim \mathrm{Unif}([n]^d)$ and C_2 is some positive constant. Therefore, by the choice of $s = C \log_2^3 N$ for some large enough constant C we have that,

$$\Pr\left[\frac{1}{2} \cdot \frac{\|\beta\|_2^2}{N^2} \leq \frac{1}{s}\sum_{j=1}^{s} |x_{\boldsymbol{t}_j}|^2 \leq \frac{3}{2} \cdot \frac{\|\beta\|_2^2}{N^2}\right] \geq 1 - \frac{1}{2N^4}.$$

By a union bound over these two events we have that $\frac{1}{2} \cdot \frac{\|\beta\|_2^2}{N^2} \leq \frac{1}{s}\sum_{j=1}^{s} |x_{\boldsymbol{t}_j}|^2 \leq \frac{3}{2} \cdot \frac{\|\beta\|_2^2}{N^2}$ with probability at least $1 - \frac{1}{N^4}$.

□

B Proof of Lemma 8

Lemma 12. *For every power of two integer n and positive integer d, if $x \in \mathbb{C}^{n^d}$ is a random support signal as per Definition 11, the following conditions hold. If $N = n^d$, $\boldsymbol{B} = (B_1, B_2, \ldots, B_d)$ is a vector of powers of two such that $B_j \mid n$ for all $j = 1, 2, \ldots, d$ and $|\boldsymbol{B}| \geq 4k$, then, with probability at least $1 - \frac{1}{N^3}$ over x,*

$$|S_x(\boldsymbol{B}, \boldsymbol{b})| = O(\log N)$$

for all $\boldsymbol{b} \in [\boldsymbol{B}]$ (where $S_x(\boldsymbol{B}, \boldsymbol{b})$ is the set from Definition 25).

Proof. Note that for every $\boldsymbol{b} \in [\boldsymbol{B}]$, we have that for each $\boldsymbol{f} \in [n]^d$ with $\boldsymbol{f} \equiv \boldsymbol{b} \pmod{[\boldsymbol{B}]}$, $\Pr[\boldsymbol{f} \in S] = k/N$. Then, since there are $N/|\boldsymbol{B}|$ such $\boldsymbol{f}$ for every fixed $\boldsymbol{b} \in [\boldsymbol{B}]$, it follows that,

$$\mathbb{E}\left[|S_x(\boldsymbol{B}, \boldsymbol{b})|\right] \leq \frac{k}{N} \cdot \frac{N}{|\boldsymbol{B}|} \leq \frac{1}{4}.$$

Hence, by the Chernoff bound, it follows that $|S_x(\boldsymbol{B}, \boldsymbol{b})| = O(\log N)$ with probability $1 - N^{-4}$ for any fixed $\boldsymbol{b} \in [\boldsymbol{B}]$. Finally, by a union bound over all $\boldsymbol{b} \in [\boldsymbol{B}]$, we have the desired result. □

We now prove a lemma about the size of the sets $S^{(\boldsymbol{B})}$:

Lemma 13. *For any power of two integers n and k, positive integer d and $\boldsymbol{B} = (B_1, B_2, \ldots, B_d)$ such that $B_1, B_2, \ldots, B_d \mid n$ and $\boldsymbol{f} = (f_1, \ldots, f_d) \in [\boldsymbol{B}]$, we have that $\Pr[\mathbf{f} \in S^{(\boldsymbol{B})}] \leq \left(\frac{k}{|\boldsymbol{B}|}\right)^2$, where $S^{(\boldsymbol{B})}$ is defined as in Definition 27.*

Proof. Suppose $\boldsymbol{f} \in [\boldsymbol{B}]$. Then, observe that there are $\binom{n}{B_1}\binom{n}{B_2} \cdot \binom{n}{B_d} = \frac{N}{B_1 B_2 \cdots B_d}$ elements $\boldsymbol{g} = (g_1, \ldots, g_d) \in [n]^d$ such that $\boldsymbol{f} \equiv \boldsymbol{g} \pmod{\boldsymbol{B}}$. Note that $\boldsymbol{f} \in S^{(\boldsymbol{B})}$ if at least two of these elements lies in S. Thus, for every $\boldsymbol{f} \in [\boldsymbol{B}]$ we have,

$$\begin{aligned}
\Pr[\boldsymbol{f} \in S^{(\boldsymbol{B})}] &= 1 - \left(1 - \frac{k}{N}\right)^{\frac{N}{|\boldsymbol{B}|}} - \frac{N}{|\boldsymbol{B}|} \cdot \frac{k}{N}\left(1 - \frac{k}{N}\right)^{\frac{N}{|\boldsymbol{B}|}-1} \\
&\leq 1 - \left(1 - \frac{k}{N}\right)^{\frac{N}{|\boldsymbol{B}|}-1}\left(1 - \frac{k}{N} + \frac{k}{|\boldsymbol{B}|}\right) \\
&\leq 1 - \left(1 - \frac{k}{|\boldsymbol{B}|} + \frac{k}{N}\right)\left(1 - \frac{k}{N} + \frac{k}{|\boldsymbol{B}|}\right) \\
&= 1 - \left(1 - \left(\frac{k}{|\boldsymbol{B}|} - \frac{k}{N}\right)^2\right) \\
&= \left(\frac{k}{|\boldsymbol{B}|} - \frac{k}{N}\right)^2 \\
&\leq \left(\frac{k}{|\boldsymbol{B}|}\right)^2,
\end{aligned}$$

since $|\boldsymbol{B}| \leq n^d = N$. □

As a consequence, we have a bound on the expected size of $S^{(\boldsymbol{B})}$.

Lemma 14. *For any power of two integers n and k, any $\boldsymbol{B} = (B_1, B_2, \ldots, B_d)$ such that $B_1, B_2, \ldots, B_d \mid n$, we have $\mathbb{E}\left[|S^{(\boldsymbol{B})}|\right] \leq \frac{k^2}{|\boldsymbol{B}|}$.*

Proof. Simply note that

$$\begin{aligned}\mathbb{E}\left[|S^{(\boldsymbol{B})}|\right] &= \sum_{\boldsymbol{f} \in [\boldsymbol{B}]} \Pr[\boldsymbol{f} \in S^{(\boldsymbol{B})}] \\ &\leq |\boldsymbol{B}| \cdot \left(\frac{k}{|\boldsymbol{B}|}\right)^2 \\ &= \frac{k^2}{|\boldsymbol{B}|},\end{aligned}$$

by Lemma 13. □

We are now ready to proof Lemma 8.

Proof of Lemma 8: Consider a fixed $\boldsymbol{b} \in [\boldsymbol{B}']$. Note that there are $m = \frac{B_1 B_2 \cdots B_d}{B_1' B_2' \cdots B_d'}$ values of $\boldsymbol{f} \in [\boldsymbol{B}]$ such that $\boldsymbol{f} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}'}$. Moreover, by Lemma 13, each such $\boldsymbol{f}$ lies in $S^{(\boldsymbol{B})}$ with identical probability

$$p \leq \left(\frac{k}{|\boldsymbol{B}|}\right)^2,$$

and these events are all independent. Thus,

$$\begin{aligned}\mathbb{E}\left[\left|S^{(\boldsymbol{B})} \cap \{\boldsymbol{f} \in [\boldsymbol{B}] : \boldsymbol{f} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}'}\}\right|\right] &\leq mp \\ &\leq \frac{k^2}{|\boldsymbol{B}| \cdot |\boldsymbol{B}'|} \\ &\leq 1.\end{aligned}$$

Thus, by the Chernoff bound, we have that

$$\left|S^{(\boldsymbol{B})} \cap \{\boldsymbol{f} \in [\boldsymbol{B}] : \boldsymbol{f} \equiv \boldsymbol{b} \pmod{\boldsymbol{B}'}\}\right| = O(\log N)$$

with probability at least $1 - \frac{1}{N^4}$, as desired. Finally, taking a union bound over all $|\boldsymbol{B}'| \leq N$ values of $\boldsymbol{b} \in [\boldsymbol{B}]$ gives the desired result. □