Simple and practical algorithm for sparse fourier transform

Simple and Practical Algorithm for Sparse Fourier Transform

Haitham Hassanieh MIT
Piotr Indyk MIT
Dina Katabi MIT
Eric Price MIT

{haithamh,indyk,dk,ecprice}@mit.edu

Abstract

We consider the sparse Fourier transform problem: given a complex vector x of length n, and a parameter k, estimate the k largest (in magnitude) coefficients of the Fourier transform of x. The problem is of key interest in several areas, including signal processing, audio/image/video compression, and learning theory.

We propose a new algorithm for this problem. The algorithm leverages techniques from digital signal processing, notably Gaussian and Dolph-Chebyshev filters. Unlike the typical approach to this problem, our algorithm is not *iterative*. That is, instead of estimating "large" coefficients, subtracting them and recursing on the reminder, it identifies and estimates the k largest coefficients in "one shot", in a manner akin to sketching/streaming algorithms. The resulting algorithm is structurally simpler than its predecessors. As a consequence, we are able to extend considerably the range of sparsity, k, for which the algorithm is faster than FFT, both in theory and practice.

1 Introduction

The Fast Fourier Transform (FFT) is one of the most fundamental numerical algorithms. It computes the Discrete Fourier Transform (DFT) of an n-dimensional signal in $O(n \log n)$ time. The algorithm plays a central role in several application areas, including signal processing and audio/image/video compression. It is also a fundamental subroutine in integer multiplication and encoding/decoding of error-correcting codes.

Any algorithm for computing the exact DFT must take time at least proportional to its output size, which is $\Theta(n)$. In many applications, however, most of the Fourier coefficients of a signal are small or equal to zero, i.e., the output of the DFT is (approximately) *sparse*. For example, a typical 8x8 block in a video frame has on average 7 non-negligible coefficients (i.e., 89% of the coefficients are negligible) [CGX96]. Images and audio data are equally sparse. This sparsity provides the rationale underlying compression schemes such as MPEG and JPEG. Other applications of sparse Fourier analysis include computational learning theory [KM91, LMN93], analysis of boolean functions [KKL88, O'D08], multiscale analysis [DRZ07], compressed sensing [Don06, CRT06], similarity search in databases [AFS93], spectrum sensing for wideband channels [LVS11], and datacenter monitoring [MNL10].

When the output of the DFT is sparse or approximately sparse, one can hope for an "output-sensitive" algorithm, whose runtime depends on k, the number of computed "large" coefficients. Formally, given a complex vector x whose Fourier transform is $\hat{x}$, we require the algorithm to output an approximation $\hat{x}'$ to $\hat{x}$ that satisfies the following ℓ_2/ℓ_2 guarantee:[1]

$$\|\hat{x} - \hat{x}'\|_2 \leq C \min_{k\text{-sparse } y} \|\hat{x} - y\|_2, \tag{1.1}$$

where C is some approximation factor and the minimization is over k-sparse signals. Note that the best k-sparse approximation can be obtained by setting all

[1] The algorithm in this paper has a somewhat stronger guarantee; see "Results" for more details.

but the largest k coefficients of $\hat{x}$ to 0. Such a vector can be represented using only $O(k)$ numbers. Thus, if k is small, the output of the algorithm can be expressed succinctly, and one can hope for an algorithm whose runtime is *sublinear* in the signal size n.

The first such sublinear algorithm (for the Hadamard transform) was presented in [KM91]. Shortly after, several sublinear algorithms for the Fourier transform over the complex field were discovered [Man92, GGI+02, AGS03, GMS05, Iwe10a, Aka10].[2] These algorithms have a runtime that is polynomial in k and $\log n$.[3] The exponents of the polynomials, however, are typically large. The fastest among these algorithms have a runtime of the form $O(k^2 \log^c n)$ (as in [GGI+02, Iwe10a]) or the form $O(k \log^c n))$(as in [GMS05]), for some constant c > 2.

In practice, the exponents in the runtime of these algorithms and their complex structures limit their applicability to only very sparse signals. In particular, the more recent algorithms were implemented and evaluated empirically against FFTW, an efficient implementation of the FFT with a runtime $O(n \log n)$ [Iwe10a, IGS07]. The results show that the algorithm in [GMS05] is competitive with FFTW for $n = 2^{22}$ and $k \leq 135$ [IGS07]. The algorithms in [GGI+02, Iwe10a] require an even sparser signal (i.e., larger n and smaller k) to be competitive with FFTW.

Results. In this paper, we propose a new sublinear algorithm for sparse Fourier transform over the complex field. The key feature of our algorithm is its simplicity: the algorithm has a simple structure, which leads to efficient runtime with low big-Oh constant. Specifically, for the typical case of n a power of 2, our algorithm has the runtime of

$$O(\log n \sqrt{nk \log n}).$$

Thus, the algorithm is faster than FFTW for k up to $O(n/\log n)$. In contrast, earlier algorithms required asymptotically smaller bounds on k.

This asymptotic improvement is also reflected in empirical runtimes. For example, for $n = 2^{22}$, our algorithm outperforms FFTW for k up to about 2200, which is an order of magnitude higher than previously achieved.

The estimations provided by our algorithm satisfy the so-called ℓ_∞/ℓ_2 guarantee. Specifically, let y be the minimizer of $\|\hat{x} - y\|_2$. For a precision parameter $\delta = 1/n^{O(1)}$, and a constant $\epsilon > 0$, our (randomized) algorithm outputs $\hat{x}'$ such that:

$$\|\hat{x} - \hat{x}'\|_\infty^2 \leq \epsilon \|\hat{x} - y\|_2^2/k + \delta \|x\|_1^2 \tag{1.2}$$

with probability $1 - 1/n$. The additive term that depends on δ appears in all past algorithms [Man92, GGI+02, AGS03, GMS05, Iwe10a, Aka10], although typically (with the exception of [Iwe10b]) it is eliminated by assuming that all coordinates are integers in the range $\{-n^{O(1)} \ldots n^{O(1)}\}$. In this paper, we keep the dependence on δ explicit.

The ℓ_∞/ℓ_2 guarantee of Equation (1.2) is *stronger* than the ℓ_2/ℓ_2 guarantee of Equation 1.1. In particular, the ℓ_∞/ℓ_2 guarantee with a constant approximation factor C implies the ℓ_2/ℓ_2 guarantee with a constant approximation factor C', if one sets all but the k largest entries in $\hat{x}'$ to 0.[4] Furthermore, instead of bounding only the collective error, the ℓ_∞/ℓ_2 guarantee ensures that every Fourier coefficient is well-approximated.

Techniques. We start with an overview of the techniques used in prior works, then describe our contribution in that context. At a high level, sparse Fourier algorithms work by binning the Fourier coefficients into a small number of buckets. Since the signal is sparse in the frequency domain, each bucket is likely[5] to have only one large coefficient, which can then be located (to find its position) and estimated (to find its value). For the algorithm to be sublinear, the binning has to be done in sublinear time. To achieve this goal, these algorithms bin the Fourier coefficient using an n-dimensional filter vector G that is concentrated both in time and frequency, i.e., G is zero except at a small *number* of time coordinates, and its Fourier transform $\hat{G}$ is negligible except at a small *fraction* (about $1/k$) of the frequency coordinates (the "pass" region). Depending on the choice of the filter G, past algorithms can be classified as: iteration-based or interpolation-based.

Iteration-based algorithms use a filter that has a significant mass outside its pass region[Man92, GGI+02, GMS05]. For example, the papers [GGI+02, GMS05] set G to the box-car function, in which case $\hat{G}$ is the Dirichlet kernel, whose tail decays in an inverse linear fashion. Since the tail decays slowly, the Fourier coefficients binned to a particular bucket "leak" into other buckets. On the other hand, the paper [Man92] estimates the convolution in time domain via random sampling, which also leads to a large estimation error. To reduce these errors and obtain the ℓ_2/ℓ_2 guarantee, these algorithms have to perform multiple iterations, where

[2] See [GST08] for a streamlined exposition of some of the algorithms.

[3] Assuming all inputs are represented using $O(\log n)$ bits.

[4] This fact was implicit in [CM06]. For an explicit statement and proof see [GI10], remarks after Theorem 2.

[5] One can randomize the positions of the frequencies by sampling the signal in time domain appropriately [GGI+02, GMS05]. See section 3 part (b) for the description.

each iteration estimates the largest Fourier coefficient (the one least impacted by leakage) and subtracts its contribution to the time signal. The iterative update of the time signal causes a large increase in runtime. The algorithms in [GGI+02, Man92] perform this update by going through $O(k)$ iterations each of which updates at least $O(k)$ time samples, resulting in an $O(k^2)$ term in the runtime. The algorithm [GMS05], introduced a "bulk sampling" algorithm that amortizes this process but it requires solving instances of a non-uniform Fourier transform, which is expensive in practice.

Interpolation-based algorithms are less common and limited to the design in [Iwe10a]. This approach uses a leakage-free filter, G, to avoid the need for iteration. Specifically, the algorithm in [Iwe10a] uses for G a filter in which $G_i = 1$ iff $i \bmod n/p = 0$ and $G_i = 0$ otherwise. The Fourier transform of this filter is a "spike train" with period p. This filter does not leak: it is equal to 1 on $1/p$ fraction of coordinates and is zero elsewhere. Unfortunately, however, such a filter requires that p divides n; moreover, the algorithm needs several different values of p. Since in general one cannot assume that n is divisible by all numbers p, the algorithm treats the signal as a continuous function and *interpolates* it at the required points. Interpolation introduces additional complexity and increases the exponents in the runtime.

Our approach. The key feature of our algorithm is the use of a different type of filter. In the simplest case, we use a filter obtained by convolving a Gaussian function with a box-car function. A more efficient filter can be obtained by replacing the Gaussian function with a Dolph-Chebyshev function. (See Fig. 1 for an illustration.)

Because of this new filter, our algorithm does not need to either iterate or interpolate. Specifically, the frequency response of our filter $\hat{G}$ is nearly flat inside the pass region and has an *exponential* tail outside it. This means that leakage from frequencies in other buckets is negligible, and hence, our algorithm need not iterate. Also, filtering can be performed using the existing input samples x_i, and hence our algorithm need not interpolate the signal at new points. Avoiding both iteration and interpolation is the key feature that makes our algorithm efficient.

Further, once a large coefficient is isolated in a bucket, one needs to identify its frequency. In contrast to past work which typically uses binary search for this task, we adopt an idea from [PS10] and tailor it to our problem. Specifically, we simply select the set of "large" bins which are likely to contain large coefficients, and directly estimate all frequencies in those bins. To balance the cost of the bin selection and estimation steps, we make the number of bins somewhat larger than the typical value of $O(k)$. Specifically, we use $B \approx \sqrt{nk}$, which leads to the stated runtime. [6]

2 Notation

Transform-Related Notations. For an input signal $x \in \mathbb{C}^n$, its Fourier spectrum is denoted by $\hat{x}$. We will use $x * y$ to denote the convolution of x and y, and $x \cdot y$ to denote the coordinate-wise product of x and y. Recall that $\widehat{x \cdot y} = \hat{x} * \hat{y}$. We define $\omega = e^{2\pi i/n}$ to be a primitive nth root of unity (here $i = \sqrt{-1}$, but in the rest of the paper, i is an index).

Indices-Related Notations. All operations on indices in this paper are taken modulo n. Therefore we might refer to an n-dimensional vector as having coordinates $\{0, 1, \ldots, n-1\}$ or $\{0, 1, \ldots, n/2, -n/2+1, \ldots, -1\}$. interchangeably. Finally, $[s]$ refers to the set of indices $\{0, \ldots, s-1\}$, whereas $\mathrm{supp}(x)$ refers to the support of vector x, i.e., the set of non-zero coordinates.

In this paper we assume that n is an integer power of 2.

3 Basics

(a) Window Functions. In digital signal processing [OSB99] one defines *window functions* in the following manner:

DEFINITION 3.1. *We define a* (ϵ, δ, w) standard window function *to be a symmetric vector* $F \in \mathbb{R}^n$ *with* $\mathrm{supp}(F) \subseteq [-w/2, w/2]$ *such that* $\hat{F}_0 = 1$, $\hat{F}_i > 0$ *for all* $i \in [-\epsilon n, \epsilon n]$, *and* $|\hat{F}_i| < \delta$ *for all* $i \notin [-\epsilon n, \epsilon n]$.

CLAIM 3.2. *For any* ϵ *and* δ, *there exists an* $(\epsilon, \delta, O(\frac{1}{\epsilon}\log(1/\delta)))$ *standard window function.*

Proof. This is a well known fact [Smi11]. For example, for any ϵ and δ, one can obtain a standard window by taking a Gaussian with standard deviation $\Theta(\sqrt{\log(1/\delta)}/\epsilon)$ and truncating it at $w = O(\frac{1}{\epsilon}\log(1/\delta)))$. The Dolph-Chebyshev window function also has the claimed property but with minimal big-Oh constant [Smi11] (in particular, half the constant of the truncated Gaussian). □

The above definition shows that a standard window function acts like a filter, allowing us to focus on a subset of the Fourier coefficients. Ideally, however, we would like the pass region of our filter to be as flat as possible.

[6] Although it is plausible that one could combine our filters with the binary search technique of [GMS05] and achieve an algorithm with a $O(k \log^c n)$ runtime, our preliminary analysis indicates that the resulting algorithm would be slower. Intuitively, observe that for $n = 2^{22}$ and $k = 2^{11}$, the values of $\sqrt{nk} = 2^{16.5} \approx 92681$ and $k \log_2 n = 45056$ are quite close to each other.

DEFINITION 3.3. *We define a* $(\epsilon, \epsilon', \delta, w)$ flat window function *to be a symmetric vector* $F \in \mathbb{R}^n$ *with* $\mathrm{supp}(F) \subseteq [-w/2, w/2]$ *such that* $\hat{F}_i \in [1-\delta, 1+\delta]$ *for all* $i \in [-\epsilon' n, \epsilon' n]$ *and* $|\hat{F}_i| < \delta$ *for all* $i \notin [-\epsilon n, \epsilon n]$.

A flat window function (like the one in Fig. 1) can be obtained from a standard window function by convolving it with a "box car" window function, i.e., an interval. Specifically, we have the following.

CLAIM 3.4. *For any* ϵ, ϵ', *and* δ *with* $\epsilon' < \epsilon$, *there exists an* $(\epsilon, \epsilon', \delta, O(\frac{1}{\epsilon - \epsilon'} \log \frac{n}{\delta}))$ *flat window function.*

Note that in our applications we have $\delta < 1/n^{O(1)}$ and $\epsilon = 2\epsilon'$. Thus the window lengths w of the flat window function and the standard window function are the same up to a constant factor.

Proof. Let $f = (\epsilon - \epsilon')/2$, and let F be an $(f, \frac{\delta}{(\epsilon'+\epsilon)n}, w)$ standard window function with minimal $w = O(\frac{2}{\epsilon - \epsilon'} \log \frac{(\epsilon+\epsilon')n}{\delta})$. We can assume $\epsilon, \epsilon' > 1/(2n)$ (because $[-\epsilon n, \epsilon n] = \{0\}$ otherwise), so $\log \frac{(\epsilon+\epsilon')n}{\delta} = O(\log \frac{n}{\delta})$. Let $\hat{F}'$ be the sum of $1 + 2(\epsilon' + f)n$ adjacent copies of $\hat{F}$, normalized to $\hat{F}'_0 \approx 1$. That is, we define

$$\hat{F}'_i = \frac{\sum_{j=-(\epsilon'+f)n}^{(\epsilon'+f)n} \hat{F}_{i+j}}{\sum_{j=-fn}^{fn} \hat{F}_j}$$

so by the shift theorem, in the time domain

$$F'_a \propto F_a \sum_{j=-(\epsilon'+f)n}^{(\epsilon'+f)n} \omega^j.$$

Since $\hat{F}_i > 0$ for $|i| \leq fn$, the normalization factor $\sum_{j=-fn}^{fn} \hat{F}_j$ is at least 1. For each $i \in [-\epsilon' n, \epsilon' n]$, the sum on top contains all the terms from the sum on bottom. The other $2\epsilon' n$ terms in the top sum have magnitude at most $\delta/((\epsilon' + \epsilon)n) = \delta/(2(\epsilon' + f)n)$, so $|\hat{F}'_i - 1| \leq 2\epsilon' n(\delta/(2(\epsilon' + f)n)) < \delta$. For $|i| > \epsilon n$, however, $\hat{F}'_i \leq 2(\epsilon' + f)n\delta/(2(\epsilon' + f)n) < \delta$. Thus F' is an $(\epsilon, \epsilon', \delta, w)$ flat window function, with the correct w. □

(b) Permutation of Spectra. Following [GMS05], we can permute the Fourier spectrum as follows by permuting the time domain:

CLAIM 3.5. *Define the transform* $P_{\sigma,\tau}$ *such that, given an* n*-dimensional vector* x, *an integer* σ *that is invertible mod* n, *and an integer* $\tau \in [n]$, $(P_{\sigma,\tau} x)_i = x_{\sigma i + \tau}$. *Then* $\widehat{(P_{\sigma,\tau} x)}_{\sigma i} = \hat{x}_i \omega^{-\tau i}$.

Proof. For all a, $\widehat{(P_{\sigma,\tau} x)}_a = \sum_{j=1}^n x_{\sigma j + \tau} \omega^{aj} = \sum_{j=1}^n x_j \omega^{a(j-\tau)\sigma^{-1}} = \hat{x}_{a\sigma^{-1}} \omega^{-\tau a \sigma^{-1}}$. □

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

Proof. If $j = a2^b$ for some odd a, then the orbit of σj is $a'2^b$ for all odd a'. There are thus $2 \cdot \mathrm{round}(C/2^{b+1}) < 4C/2^{b+1}$ possible values in $[-C, C]$ out of $n/2^{b+1}$ such elements in the orbit, for a chance of at most $4C/n$. □

Note that for simplicity, we will only analyze our algorithm when n is a power of two. For general n, the analog of Lemma 3.6 would lose an $n/\varphi(n) = O(\log \log n)$ factor, where φ is Euler's totient function. This will correspondingly increase the running time of the algorithm on general n.

Claim 3.5 allows us to change the set of coefficients binned to a bucket by changing the permutation; Lemma 3.6 bounds the probability of non-zero coefficients falling into the same bucket.

(c) Subsampled FFT. Suppose we have a vector $x \in \mathbb{C}^n$ and a parameter B dividing n, and would like to compute $\hat{y}_i = \hat{x}_{i(n/B)}$ for $i \in [B]$.

CLAIM 3.7. $\hat{y}$ *is the* B*-dimensional Fourier transform of* $y_i = \sum_{j=0}^{n/B-1} x_{i+Bj}$.

Therefore $\hat{y}$ *can be computed in* $O(|\mathrm{supp}(x)| + B \log B)$ *time.*

Proof.

$$\hat{x}_{i(n/B)} = \sum_{j=0}^{n-1} x_j \omega^{ij(n/B)} = \sum_{a=0}^{B-1} \sum_{j=0}^{n/B-1} x_{Bj+a} \omega^{i(Bj+a)n/B}$$
$$= \sum_{a=0}^{B-1} \sum_{j=0}^{n/B-1} x_{Bj+a} \omega^{ian/B} = \sum_{a=0}^{B-1} y_a \omega^{ian/B} = \hat{y}_i,$$

as desired. □

4 Algorithm

A key element of our algorithm is the *inner loop*, which finds and estimates each "large" coefficient with constant probability. In §4.1 we describe the inner loop, and in §4.2 we show how to use it to construct the full algorithm.

4.1 Inner Loop Let B be a parameter that divides n, to be determined later. Let G be a $(1/B, 1/(2B), \delta, w)$ flat window function for some δ and $w = O(B \log \frac{n}{\delta})$. We will have $\delta \approx 1/n^c$, so one can think of it as negligibly small.

There are two versions of the inner loop: *location* loops and *estimation* loops. Location loops are given a parameter d, and output a set $I \subset [n]$ of dkn/B coordinates that contains each large coefficient with

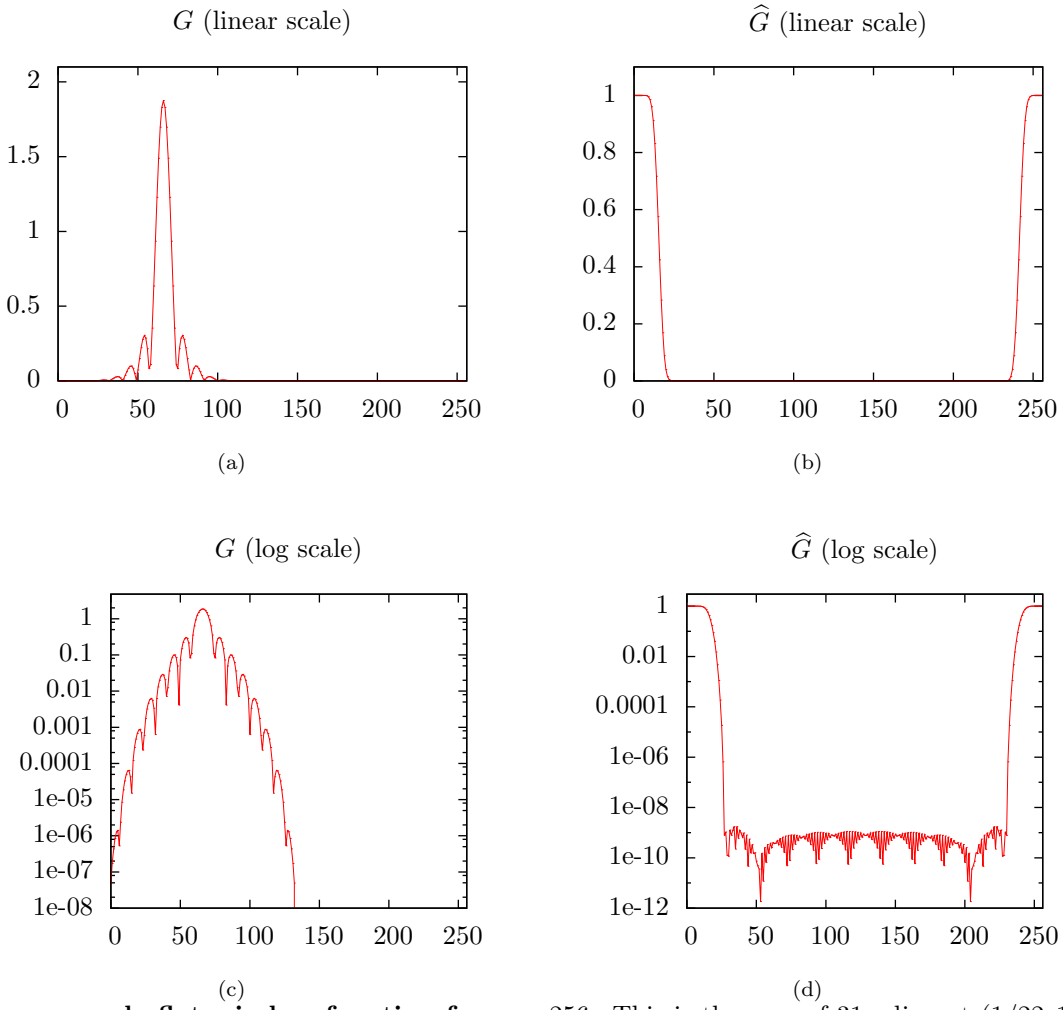


Figure 1: **An example flat window function for** $n = 256$**.** This is the sum of 31 adjacent $(1/22, 10^{-8}, 133)$ Dolph-Chebyshev window functions, giving a $(0.11, 0.06, 2 \times 10^{-9}, 133)$ flat window function (although our proof only guarantees a tolerance $\delta = 29 \times 10^{-8}$, the actual tolerance is better). The top row shows G and $\widehat{G}$ in a linear scale, and the bottom row shows the same with a log scale.

"good" probability. Estimation loops are given a set $I \subset [n]$ and estimate x_I such that each coordinate is estimated well with "good" probability.

1. Choose a random $\sigma, \tau \in [n]$ with σ odd.
2. Define $y = G \cdot (P_{\sigma,\tau} x)$, so $y_i = G_i x_{\sigma i+\tau}$. Then $\mathrm{supp}(y) \subseteq \mathrm{supp}(G) = [w]$.
3. Compute $\hat{z}_i = \hat{y}_{i(n/B)}$ for $i \in [B]$. By Claim 3.7, this is the DFT of $z_i = \sum_{j=0}^{\lceil w/B \rceil - 1} y_{i+jB}$.
4. Define the "hash function" $h_\sigma \colon [n] \to [B]$ by $h_\sigma(i) = \mathrm{round}(\sigma i B/n)$ and the "offset" $o_\sigma \colon [n] \to [-n/(2B), n/(2B)]$ by $o_\sigma(i) = \sigma i - h_\sigma(i)(n/B)$.
5. Location loops: let J contain the dk coordinates of maximum magnitude in $\hat{z}$. Output $I = \{i \in [n] \mid h_\sigma(i) \in J\}$, which has size dkn/B.
6. Estimation loops: for $i \in I$, estimate $\hat{x}_i$ as $\hat{x}'_i = \hat{z}_{h_\sigma(i)} \omega^{\tau i} / \hat{G}_{o_\sigma(i)}$.

By Claim 3.7, we can compute $\hat{z}$ in $O(w + B \log B) = O(B \log \frac{n}{\delta})$ time. Location loops thus take $O(B \log \frac{n}{\delta} + dkn/B)$ time and estimation loops take $O(B \log \frac{n}{\delta} + |I|)$ time. Fig. 2 illustrates the inner loop.

For estimation loops, we get the following guarantee:

LEMMA 4.1. *Let S be the support of the largest k coefficients of $\hat{x}$, and $\hat{x}_{-S}$ contain the rest. Then for any $\epsilon \le 1$,*

$$\Pr_{\sigma,\tau}[|\hat{x}'_i - \hat{x}_i|^2 \ge \frac{\epsilon}{k} \|\hat{x}_{-S}\|_2^2 + 3\delta^2 \|\hat{x}\|_1^2] < O(\frac{k}{\epsilon B}).$$

Proof. By the definitions,

$$\hat{x}'_i = \hat{z}_{h_\sigma(i)} \omega^{\tau i} / \hat{G}_{o_\sigma(i)} = \hat{y}_{\sigma i - o_\sigma(i)} \omega^{\tau i} / \hat{G}_{o_\sigma(i)}.$$

Consider the case that $\hat{x}$ is zero everywhere but at i, so $\mathrm{supp}(\widehat{P_{\sigma,\tau} x}) = \{\sigma i\}$. Then $\hat{y}$ is the convolution of $\hat{G}$ and $\widehat{P_{\sigma,\tau} x}$, and $\hat{G}$ is symmetric, so

$$\begin{aligned}\hat{y}_{\sigma i - o_\sigma(i)} &= (\hat{G} * \widehat{P_{\sigma,\tau} x})_{\sigma i - o_\sigma(i)} = \hat{G}_{-o_\sigma(i)} \widehat{P_{\sigma,\tau} x}_{\sigma i} \\ &= \hat{G}_{o_\sigma(i)} \hat{x}_i \omega^{-\tau i}\end{aligned}$$

which shows that $\hat{x}'_i = \hat{x}_i$ in this case. But $\hat{x}'_i - \hat{x}_i$ is linear in $\hat{x}$, so in general we can assume $\hat{x}_i = 0$ and bound $|\hat{x}'_i|$.

Since $|\hat{x}'_i| = |\hat{z}_{h_\sigma(i)} / \hat{G}_{o_\sigma(i)}| < |\hat{z}_{h_\sigma(i)}/(1-\delta)| = |\hat{y}_{\sigma i - o_\sigma(i)}|/(1-\delta)$, it is sufficient to bound $|\hat{y}_{\sigma i - o_\sigma(i)}|$.

Define $T = \{j \in [n] \mid \sigma(i-j) \in [-2n/B, 2n/B]\}$. We have that

$$\begin{aligned}|\hat{y}_{\sigma i - o_\sigma(i)}|^2 &= \left| \sum_{t=0}^{n-1} (\widehat{P_{\sigma,\tau} x})_t \hat{G}_{\sigma i - t - o_\sigma(i)} \right|^2 \\ &= \left| \sum_{j=0}^{n-1} (\widehat{P_{\sigma,\tau} x})_{\sigma j} \hat{G}_{\sigma(i-j) - o_\sigma(i)} \right|^2 \\ &\le 2 \left| \sum_{j \in T} (\widehat{P_{\sigma,\tau} x})_{\sigma j} \hat{G}_{\sigma(i-j) - o_\sigma(i)} \right|^2 \\ &\quad + 2 \left| \sum_{j \notin T} (\widehat{P_{\sigma,\tau} x})_{\sigma j} \hat{G}_{\sigma(i-j) - o_\sigma(i)} \right|^2 \\ &\le 2(1+\delta)^2 \left| \sum_{j \in T} (\widehat{P_{\sigma,\tau} x})_{\sigma j} \right|^2 + 2\delta^2 \left| \sum_{j \notin T} (\widehat{P_{\sigma,\tau} x})_{\sigma j} \right|^2\end{aligned}$$

In the last step, the left term follows from $|\hat{G}_a| \le 1 + \delta$ for all a. The right term is because for $j \notin T$, $|\sigma(i-j) - o_\sigma(i)| \ge |\sigma(i-j)| - |o_\sigma(i)| > 2n/B - n/(2B) > n/B$, and $|\hat{G}_a| \le \delta$ for $|a| > n/B$. By Claim 3.5, this becomes:

$$|\hat{y}_{\sigma i - o_\sigma(i)}|^2 \le 2(1+\delta)^2 \left| \sum_{j \in T} \hat{x}_j \omega^{-\tau j} \right|^2 + 2\delta^2 \|\hat{x}\|_1^2 .$$

Define $V = \left| \sum_{j \in T} \hat{x}_j \omega^{-\tau j} \right|^2$. As a choice over τ, V is the energy of a random Fourier coefficient of the vector $\hat{x}_T$ so we can bound the expectation over τ:

$$\mathop{\mathbb{E}}_{\tau}[V] = \|\hat{x}_T\|_2^2 .$$

Now, for each coordinate $j \ne i$, $\Pr_\sigma[j \in T] \le 8/B$ by Lemma 3.6. Thus $\Pr_\sigma[S \cap T \ne \emptyset] \le 8k/B$, so with probability $1 - 8k/B$ over σ we have

$$\|\hat{x}_T\|_2^2 = \left\|\hat{x}_{T \setminus S}\right\|_2^2 .$$

Let E be the event that this occurs, so E is 0 with probability $8k/B$ and 1 otherwise. We have

$$\mathop{\mathbb{E}}_{\sigma,\tau}[EV] = \mathop{\mathbb{E}}_{\sigma}[E \|\hat{x}_T\|_2^2] = \mathop{\mathbb{E}}_{\sigma}[E \left\|\hat{x}_{T \setminus S}\right\|_2^2] \le \mathop{\mathbb{E}}_{\sigma}[\left\|\hat{x}_{T \setminus S}\right\|_2^2].$$

Furthermore, we know

$$\begin{aligned}\mathop{\mathbb{E}}_{\sigma,\tau}[EV] &\le \mathop{\mathbb{E}}_{\sigma}[\left\|\hat{x}_{T \setminus S}\right\|_2^2] = \sum_{j \notin S} |\hat{x}_j|^2 \Pr_\sigma[\sigma(i-j) \in [-2n/B, 2n/B]] \\ &\le \frac{8}{B} \|\hat{x}_{-S}\|_2^2\end{aligned}$$

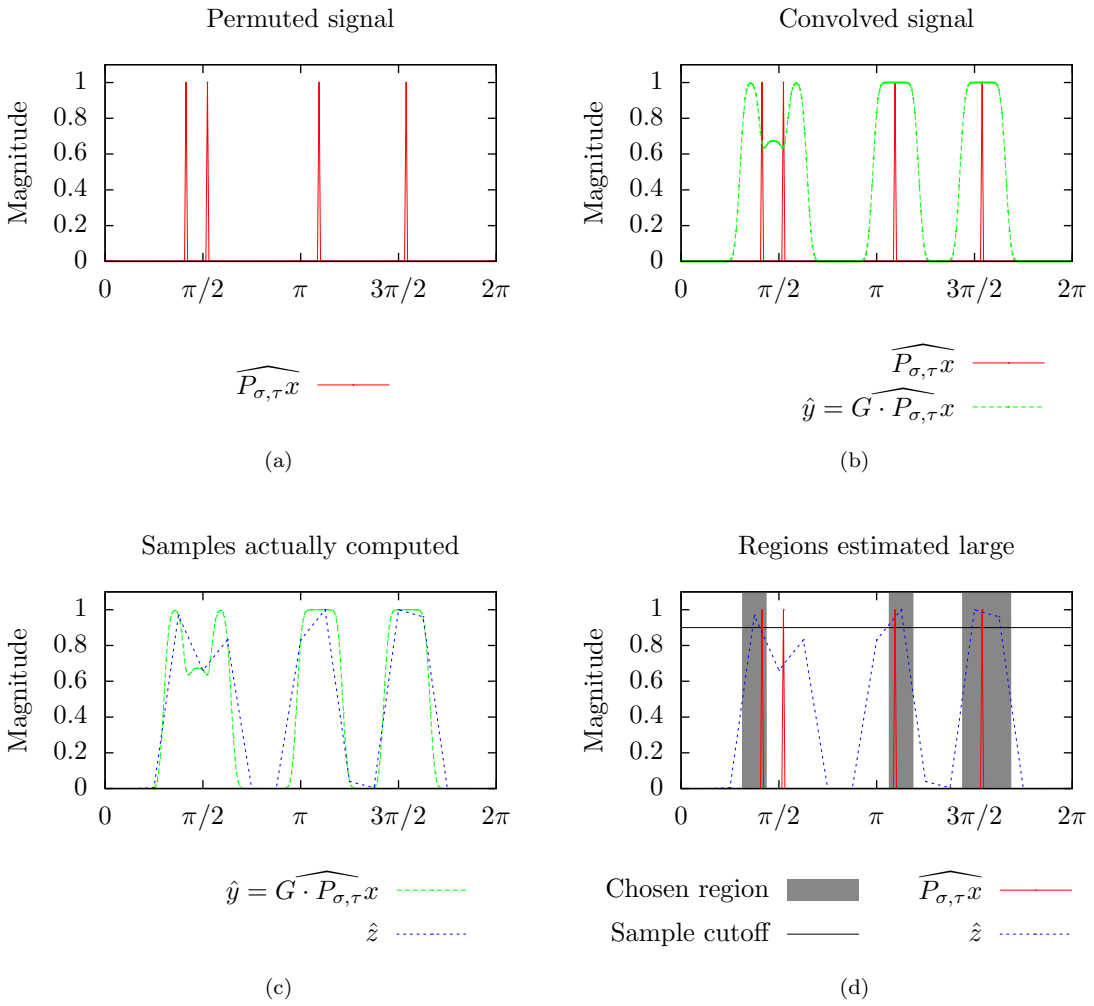


Figure 2: **Example inner loop of the algorithm on sparse input.** This run has parameters $n = 256$, $k = 4$, G being the $(0.11, 0.06, 2 \times 10^{-9}, 133)$ flat window function in Fig. 1, and selecting the top 4 of $B = 16$ samples. In part (a), the algorithm begins with time domain access to $P_{\sigma,\tau}x$ given by $(P_{\sigma,\tau}x)_i = x_{\sigma i+\tau}$, which permutes the spectrum of x by permuting the samples in the time domain. In part (b), the algorithm computes the time domain signal $y = G \cdot P_{\sigma,\tau}x$. The spectrum of y (pictured) is large around the large coordinates of $P_{\sigma,\tau}x$. The algorithm then computes $\hat{z}$, which is the rate B subsampling of $\hat{y}$ as pictured in part (c). During estimation loops, the algorithm estimates $\hat{x}_i$ based on the value of the nearest coordinate in $\hat{z}$, namely $\hat{z}_{h_\sigma(i)}$. During location loops (part (d)), the algorithm chooses J, the top dk (here, 4) coordinates of $\hat{z}$, and selects the elements of $[n]$ that are closest to those coordinates (the shaded region of the picture). It outputs the set I of preimages of those elements. In this example, the two coordinates on the left landed too close in the permutation and form a "hash collision". As a result, the algorithm misses the second from the left coordinate in its output. Our guarantee is that each large coordinate has a low probability of being missed if we select the top $O(k)$ samples.

by Lemma 3.6. Therefore, by Markov's inequality and a union bound, we have for any $C > 0$ that

$$\Pr_{\sigma,\tau}[V > 8\frac{C}{B}\left\|\hat{x}_{-S}\right\|_2^2] \leq \Pr_{\sigma,\tau}[E = 0 \cup EV > C \mathop{\mathbb{E}}_{\sigma,\tau}[EV]] \leq 8\frac{k}{B} + 1/C.$$

Hence

$$\Pr_{\sigma,\tau}[|\hat{y}_{\sigma i - o_\sigma(i)}|^2 \geq 16\frac{C}{B}(1+\delta)^2\left\|\hat{x}_{-S}\right\|_2^2 + 2\delta^2\left\|\hat{x}\right\|_1^2] < 8\frac{k}{B} + 1/C.$$

Replacing C with $\epsilon B/(32k)$ and using $|\hat{x}'_i - \hat{x}_i| < |\hat{y}_{\sigma i - o_\sigma(i)}|/(1-\delta)$ gives

$$\Pr_{\sigma,\tau}[|\hat{x}'_i - \hat{x}_i|^2 \geq \frac{\epsilon}{2k}(\frac{1+\delta}{1-\delta})^2\left\|\hat{x}_{-S}\right\|_2^2 + \frac{2}{1-\delta}\delta^2\left\|\hat{x}\right\|_1^2] < (8 + 32/\epsilon)\frac{k}{B}.$$

which implies the result. □

Furthermore, since $|\hat{G}_{o_\sigma(i)}| \in [1-\delta, 1+\delta]$, $|\hat{z}_{h_\sigma(i)}|$ is a good estimate for $|\hat{x}_i|$—the division is mainly useful for fixing the phase. Therefore in location loops, we get the following guarantee:

LEMMA 4.2. *Define* $E = \sqrt{\frac{\epsilon}{k}\left\|\hat{x}_{-S}\right\|_2^2 + 3\delta^2\left\|\hat{x}\right\|_1^2}$ *to be the error tolerated in Lemma 4.1. Then for any* $i \in [n]$ *with* $|\hat{x}_i| \geq 4E$,

$$\Pr[i \notin I] \leq O(\frac{k}{\epsilon B} + \frac{1}{\epsilon d})$$

Proof. With probability at least $1 - O(\frac{k}{\epsilon B})$, $|\hat{x}'_i| \geq |\hat{x}_i| - E \geq 3E$. In this case $|\hat{z}_{h_\sigma(i)}| \geq 3(1-\delta)E$. Thus it is sufficient to show that, with probability at least $1 - O(\frac{1}{\epsilon d})$, there are at most dk locations $j \in [B]$ with $|\hat{z}_j| \geq 3(1-\delta)E$. Since each $\hat{z}_j$ corresponds to n/B locations in $[n]$, we will show that with probability at least $1 - O(\frac{1}{\epsilon d})$, there are at most dkn/B locations $j \in [n]$ with $|\hat{x}'_j| \geq 3(1-\delta)^2E$.

Let $U = \{j \in [n] \mid |\hat{x}_j| \geq E\}$, and $V = \{j \in [n] \mid |\hat{x}'_j - \hat{x}_j| \geq E\}$. Therefore $\{j \in [n] \mid |\hat{x}'_j| \geq 2E\} \subseteq U \cup V$. Since $2 \leq 3(1-\delta)^2$, we have

$$\left|\{j \mid |\hat{x}'_j| \geq 3(1-\delta)^2E\}\right| \leq |U \cup V|.$$

We also know

$$|U| \leq k + \frac{\left\|x_{-S}\right\|_2^2}{E^2} \leq k(1 + 1/\epsilon)$$

and by Lemma 4.1,

$$\mathbb{E}[|V|] \leq O(\frac{kn}{\epsilon B}).$$

Thus by Markov's inequality $\Pr[|V| \geq dk\frac{n}{B}] \leq O(\frac{1}{\epsilon d})$, or

$$\Pr[|U \cup V| \geq dk\frac{n}{B} + k(1 + 1/\epsilon)] \leq O(\frac{1}{\epsilon d}).$$

Since the RHS is only meaningful for $d > 1/\epsilon$ we have $dk\frac{n}{B} > k(1+1/\epsilon)$. Therefore

$$\Pr[|U \cup V| \geq dk\frac{n}{B}] \leq O(\frac{1}{\epsilon d}).$$

and thus

$$\Pr[|\{j \in [B] \mid |\hat{z}_j| \geq 3(1-\delta)E\}| > kd] \leq O(\frac{1}{\epsilon d}).$$

Hence a union bound over this and the probability that $|\hat{x}'_i - \hat{x}_i| < E$ gives

$$\Pr[i \notin I] \leq O(\frac{k}{\epsilon B} + \frac{1}{\epsilon d})$$

as desired. □

4.2 Outer Loop Our algorithm is parameterized by ϵ and δ. It runs $L = O(\log n)$ iterations of the inner loop, with parameters $B = O(\sqrt{\frac{nk}{\epsilon \log(n/\delta)}})$ [7] and $d = O(1/\epsilon)$ as well as δ.

1. For $r \in \{1, \ldots, L\}$, run a location inner loop to get I_r.
2. For each $i \in I = I_1 \cup \cdots \cup I_L$, let $s_i = |\{r \mid i \in I_r\}|$.
3. Let $I' = \{i \in I \mid s_i \geq L/2\}$.
4. For $r \in \{1, \ldots, L\}$, run an estimation inner loop on I' to get $\hat{x}^r_{I'}$.
5. For $i \in I'$, estimate $\hat{x}'_i = \text{median}(\{\hat{x}^r_i \mid i \in I'\})$, where we take the median in real and imaginary coordinates separately.

LEMMA 4.3. *The algorithm runs in time* $O(\sqrt{\frac{nk\log(n/\delta)}{\epsilon}}\log n)$.

Proof. To analyze this algorithm, note that

$$|I'|\frac{L}{2} \leq \sum_i s_i = \sum_r |I_r| = Ldkn/B$$

[7] Note that B is chosen in order to minimize the running time. For the purpose of correctness, it suffices that $B \geq ck/\epsilon$ for some constant c. We will use this fact later in the experimental section.

or $|I'| \leq 2dkn/B$. Therefore the running time of both the location and estimation inner loops is $O(B \log \frac{n}{\delta} + dkn/B)$. Computing I' and computing the medians both take linear time, namely $O(Ldkn/B)$. Thus the total running time is $O(LB \log \frac{n}{\delta} + Ldkn/B)$. Plugging in $B = O(\sqrt{\frac{nk}{\epsilon \log(n/\delta)}})$ and $d = O(1/\epsilon)$, this running time is $O(\sqrt{\frac{nk \log(n/\delta)}{\epsilon}} \log n)$. We require $B = \Omega(k/\epsilon)$, however; for $k > \epsilon n / \log(n/\delta)$, this would cause the run time to be larger. But in this case, the predicted run time is $\Omega(n \log n)$ already, so the standard FFT is faster and we can fall back on it. □

THEOREM 4.4. *Running the algorithm with parameters $\epsilon, \delta < 1$ gives $\hat{x}'$ satisfying*

$$\|\hat{x}' - \hat{x}\|_\infty^2 \leq \frac{\epsilon}{k} \|\hat{x}_{-S}\|_2^2 + \delta^2 \|\hat{x}\|_1^2 .$$

with probability $1 - 1/n$ and running time $O(\sqrt{\frac{nk \log(n/\delta)}{\epsilon}} \log n)$.

Proof. Define

$$E = \sqrt{\frac{\epsilon}{k} \|\hat{x}_{-S}\|_2^2 + 3\delta^2 \|\hat{x}\|_1^2}$$

Lemma 4.2 says that in each location iteration r, for any i with $|\hat{x}_i| \geq 4E$,

$$\Pr[i \notin I^r] \leq O(\frac{k}{\epsilon B} + \frac{1}{\epsilon d}) \leq 1/4.$$

Thus $\mathbb{E}[s_i] \geq 3L/4$, and each iteration is an independent trial, so by a Chernoff bound the chance that $s_i < L/2$ is at most $1/2^{\Omega(L)} < 1/n^3$. Therefore by a union bound, with probability at least $1 - 1/n^2$, $i \in I'$ for all i with $|\hat{x}_i| \geq 4E$.

Next, Lemma 4.1 says that for each estimation iteration r and index i,

$$\Pr[|\hat{x}_i^r - \hat{x}_i| \geq E] \leq O(\frac{k}{\epsilon B}) < 1/4.$$

Therefore, with probability $1 - 2^{-\Omega(L)} \geq 1 - 1/n^3$, $|\hat{x}_i^r - \hat{x}_i| \leq E$ in at least $2L/3$ of the iterations.

Since $\text{real}(\hat{x}_i')$ is the median of the $\text{real}(\hat{x}_i^r)$, there must exist two r with $|\hat{x}_i^r - \hat{x}_i| \leq E$ but one $\text{real}(\hat{x}_i^r)$ above $\text{real}(\hat{x}_i')$ and one below. Hence one of these r has $|\text{real}(\hat{x}_i' - \hat{x}_i)| \leq |\text{real}(\hat{x}_i^r - \hat{x}_i)| \leq E$, and similarly for the imaginary axis. Then

$$|\hat{x}_i' - \hat{x}_i| \leq \sqrt{2} \max(|\text{real}(\hat{x}_i' - \hat{x}_i)|, |\text{imag}(\hat{x}_i' - \hat{x}_i)|) \leq \sqrt{2}E.$$

By a union bound over I', with probability at least $1 - 1/n^2$ we have $|\hat{x}_i' - \hat{x}_i| \leq \sqrt{2}E$ for all $i \in I'$. Since all $i \notin I'$ have $\hat{x}_i' = 0$ and $|\hat{x}_i| \leq 4E$ with probability $1 - 1/n^2$, with total probability $1 - 2/n^2$ we have

$$\|\hat{x}' - \hat{x}\|_\infty^2 \leq 16E^2 = \frac{16\epsilon}{k} \|\hat{x}_{-S}\|_2^2 + 48\delta^2 \|\hat{x}\|_1^2 .$$

Rescaling ϵ and δ gives our theorem. □

5 Experimental Evaluation

5.1 Implementation We implement our algorithm in C++ using the Standard Template Library, and refer to this implementation as sFFT (which stands for "sparse FFT"). We have two versions: sFFT 1.0, which implements the algorithm as in §4, and sFFT 2.0, which adds a heuristic to improve the runtime.

sFFT 2.0. The idea of the heuristic is to apply the filter used in [Man92] to restrict the locations of the large coefficients. The heuristic is parameterized by an M dividing n. During a preprocessing stage, it does the following:

1. Choose $\tau \in [n]$ uniformly at random.
2. For $i \in [M]$, set $y_i = x_{i(n/M)+\tau}$.
3. Compute $\hat{y}$.
4. Output $T \subset [M]$ containing the $2k$ largest elements of $\hat{y}$.

Observe that $\hat{y}_i = \sum_{j=0}^{n/M} \hat{x}_{Mj+i} \omega^{-\tau(i+Mj)}$. Thus,

$$\mathop{\mathbb{E}}_{\tau}[|\hat{y}_i|^2] = \sum_{i \equiv j \bmod M} |\hat{x}_j|^2.$$

This means that the filter is very efficient, in that it has no leakage at all. Also, it is simple to compute. Unfortunately, it cannot be "randomized" using $P_{\sigma,\tau}$: after permuting by σ, any two colliding elements j and j' (i.e., such that $j = j' \pmod M$) continue to collide. Nevertheless, if $\hat{x}_j$ is large, then $j \pmod M$ is likely to lie in T—at least heuristically on random input.

sFFT 2.0 completes the algorithm assuming that all "large" coefficients j have $j \bmod M$ in T. That is, in the main algorithm of §4, we then restrict our sets I_r to contain only coordinates i with $(i \bmod M) \in T$. We expect that $|I_r| \approx \frac{2k}{M} dkn/B$ rather than the previous dkn/B. This means that our heuristic will improve the runtime of the inner loops from $O(B \log(n/\delta) + dkn/B)$ to $O(B \log(n/\delta) + \frac{k}{M} dkn/B + M + dk)$, at the cost of $O(M \log M)$ preprocessing.

Note that on worst case input, sFFT 2.0 may give incorrect output with high probability. For example, if $x_i = 1$ when i is a multiple of n/M and 0 otherwise, then $y = 0$ with probability $1 - M/n$ and the algorithm will output 0 over $\text{supp}(x)$. However, in practice the algorithm works for "sufficiently random" x.

CLAIM 5.1. *As a heuristic approximation, sFFT 2.0 runs in* $O((k^2 n \log(n/\delta)/\epsilon)^{1/3} \log n)$ *as long as* $k \leq \epsilon^2 n \log(n/\delta)$.

Justification. First we will show that the heuristic improves the inner loop running time to $O(B \log(n/\delta) + \frac{k}{M} dkn/B + M + dk)$, then optimize the parameters M and B.

Heuristically, one would expect each of the I_r to be a $\frac{|T|}{M}$ factor smaller than if we did not require the elements to lie in T modulo M. Hence, we expect each of the I_r and I' to have size $\frac{|T|}{M} dkn/B = O(\frac{k}{M} dkn/B)$. Then in each location loop, rather than spending $O(dkn/B)$ time to list our output, we spend $O(\frac{k}{M} dkn/B)$ time—plus the time required to figure out where to start listing coordinates from each of the dk chosen elements J of $\hat{z}$. We do this by sorting J and $\{\sigma i \mid i \in T\}$ (mod M), then scanning through the elements. It takes $O(M+dk)$ time to sort $O(dk)$ elements in $[M]$, so the total runtime of each location loop is $O(B \log(n/\delta) + \frac{k}{M} dkn/B + M + dk)$. The estimation loops are even faster, since they benefit from $|I'|$ being smaller but avoid the $M + dk$ penalty.

The full algorithm does $O(M \log M)$ preprocessing and runs the inner loop $L = O(\log n)$ times with $d = O(1/\epsilon)$. Therefore, given parameters B and M, the algorithm takes $O(M \log M + B \log \frac{n}{\delta} \log n + \frac{k}{M} \frac{kn}{\epsilon B} \log n + M \log n + \frac{k}{\epsilon} \log n)$ time. Optimizing over B, we take

$$O(M \log n + k\sqrt{\frac{n}{M\epsilon} \log(n/\delta)} \log n + \frac{k}{\epsilon} \log n)$$

time. Then, optimizing over M, this becomes

$$O((k^2 n \log(n/\delta)/\epsilon)^{1/3} \log n + \frac{k}{\epsilon} \log n)$$

time. If $k < \epsilon^2 n \log(n/\delta)$, the first term dominates.

Note that this is an $(\frac{n \log(n/\delta)}{\epsilon k})^{1/6}$ factor smaller than the running time of sFFT 1.0.

5.2 Numerical Results We evaluate the performance of sFFT 1.0 and sFFT 2.0, and compare them against two baselines: 1) FFTW 3.2.2 [FJ], which is the fastest public implementation for computing the DFT and has a runtime of $O(n \log(n))$, and 2) AAFFT 0.9 [Iwe08], which is the prior sublinear algorithm with the fastest empirical runtime [IGS07]. For completeness, we compare against two variants of FFTW: basic and optimized. The optimized version requires preprocessing, during which the algorithm is tuned to a particular machine hardware. In contrast, our current implementations of sFFT variants do not perform hardware specific optimizations.

Experimental Setup. The test signals are generated in a manner similar to that in [IGS07]. For the runtime experiments, k frequencies are selected uniformly at random from $[n]$ and assigned a magnitude of 1 and a uniformly random phase. The rest are set to zero. For the tolerance to noise experiments, the test signals are generated as before but they are combined with additive white Gaussian noise, whose variance varies depending on the desired SNR. Each point in the graphs is the average over 100 runs with different instances of test signals and different instances of noise. In all experiments, the parameters of sFFT 1.0, sFFT 2.0, and AAFFT 0.9 are chosen so that the average L^1 error in the absence of noise is between 10^{-7} and 10^{-8} per non-zero frequency.[8] Finally, all experiments are run on a Dual Core Intel 3.0 GHz CPU running Ubuntu Linux 10.04 with a cache size of 6144 KB and 8 GB of RAM.

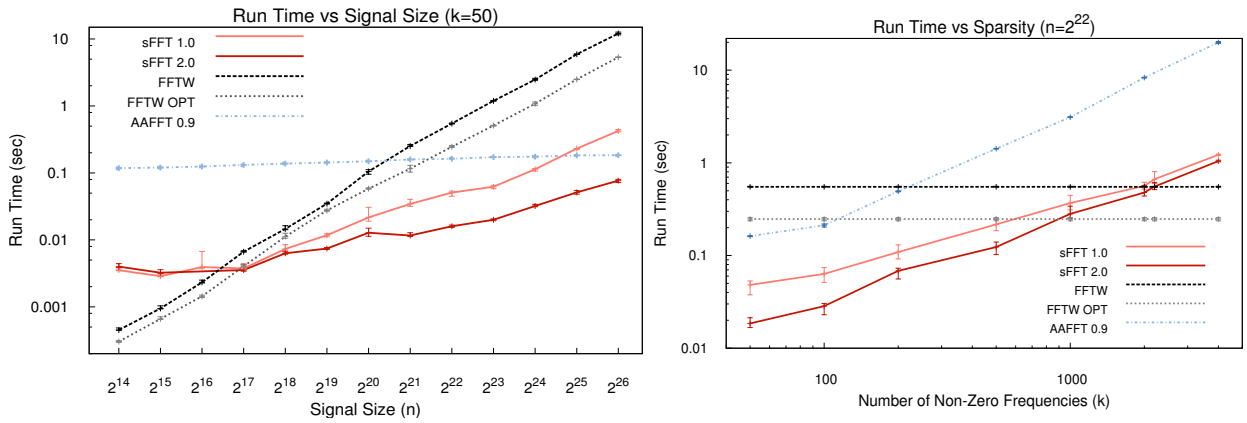
Runtime vs. Signal Size. In this experiment, we fix the sparsity parameter $k = 50$ and report the runtime of the compared algorithms for 12 different signal sizes $n : 2^{14}, 2^{15}, ..., 2^{26}$. We plot, in Fig. 3(a), the mean, maximum, and minimum runtimes for sFFT 1.0, sFFT 2.0, AAFFT 0.9, FFTW, and FFTW OPT, over 100 runs. The relative runtimes of AAFFT 0.9 and FFTW are consistent with those reported in [IGS07], Fig. 3.1.

As expected, Fig. 3(a) shows that the runtimes of sFFT and FFTW (and their variants) are approximately linear in the log scale. However, the slope of the line for sFFT is less than the slope for FFTW, which is a result of sFFT's sub-linear runtime. Further, the figure shows that for signal sizes $n > 100,000$ both sFFT 1.0 and sFFT 2.0 are faster than both variants of FFTW at recovering the exact 50 non-zero coefficients. On the other hand, the runtime of AAFFT 0.9 is proportional to polylog(n) and thus it appears almost constant as we increase the signal size. For $n = 2^{26}$ (i.e., 67,108,864), AAFFT 0.9 eventually beats the runtime of sFFT 1.0 and is only 2 times slower than sFFT 2.0. Overall, for a large range of signal sizes from about 100,000 to 67,108,864, sFFT has the fastest runtime.

Runtime vs. Number of Non-Zero Frequencies. In this experiment, we fix the signal size to $n = 2^{22}$ (i.e. 4,194,304) and evaluate the run time of sFFT vs. the number of non-zero frequencies k. For each value of k, the experiment is repeated 100 times. Fig. 3(b) illustrates the mean, maximum, and minimum runtimes for the compared algorithms.

Fig. 3(b) shows that sFFT 1.0 and sFFT 2.0 have a faster runtime than basic FFTW for k up to 2000 and

[8] For the values of k and n that are close to the ones considered in [IGS07], we use the parameters therein. For other ranges, we follow the guidelines in the AAFFT 0.9 documentation [Iwe08].



(a) (b)

Figure 3: **(a) Runtime vs. signal size.** The figure shows that for a large range of signal sizes, $n \in [2^{17}, 2^{26}]$, sFFT is faster than FFTW and the state-of-the-art sublinear algorithm. **(b) Runtime vs. sparsity parameter.** The figure shows that sFFT significantly extends the range of applications for which sparse approximation of DFT is practical, and beats the runtime of FFTW for values of k order of magnitude larger than those achieved by past work.

2200, respectively. When compared to the optimized FFTW, the crossing values become 500 and 1000. Thus, sFFT's crossing values are around $\sqrt{n}$. In comparison, AAFFT 0.9 is faster than FFTW variants for k between 100 and 200. Further, the relative runtimes of AAFFT 0.9, and FFTW 3.2.2 are close to those reported in [IGS07], Fig. 3.2. Finally, FFTW has a runtime of $O(n \log(n))$, which is independent of the number of non-zero frequencies k, as can be seen in Fig. 3(b). Thus, as the sparsity of the signal decreases (i.e., k increases), FFTW eventually becomes faster than sFFT and AAFFT. Nonetheless, the results show that in comparison with the fastest prior sublinear algorithm [IGS07], sFFT significantly extends the range of applications for which sparse approximation of DFT is practical.

Robustness to Noise. Last, we would like to check that sFFT's reduced runtime does not come at the expense of reducing robustness to noise. Thus, we compare the performance of sFFT 1.0 and sFFT 2.0 against AAFFT 0.9, for different levels of white Gaussian noise. Specifically, we fix the $n = 2^{22}$ and $k = 50$, and experiment with different signal SNRs.[9] We change the SNR by changing the variance of the Gaussian noise. For each noise variance, we run multiple experiments by regenerating new instances of the signal and noise vectors. For each run, we compute the error metric per as the average L_1 error between the output approximation $\hat{x}'$ (restricted to its k largest entries) and the best k-sparse approximation of $\hat{x}$ referred to as $\hat{y}$:

$$\text{Average } L_1 \text{ Error } = \frac{1}{k} \sum_{0<i\leq n} |\hat{x}'_i - \hat{y}_i|.$$

Fig. 4 plots the average error per non-zero frequency for sFFT 1.0, sFFT 2.0, and AAFFT 0.9. The figure shows that all three algorithms are stable under noise. Further, sFFT variants appear to be more robust to noise than AAFFT 0.9.

[9]The SNR is defined as $SNR = 20 \log \frac{||x||_2}{||z||_2}$, where z is an n-dimensional noise vector.

References

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Int. Conf. on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

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

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

[CGX96] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos. Data driven signal processing: An approach for energy efficient computing. *International Symposium on Low Power Electronics and Design*, 1996.

[CM06] G. Cormode and S. Muthukrishnan. Combinatorial algorithms for compressed sensing. *SIROCCO*, 2006.

[CRT06] E. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE*

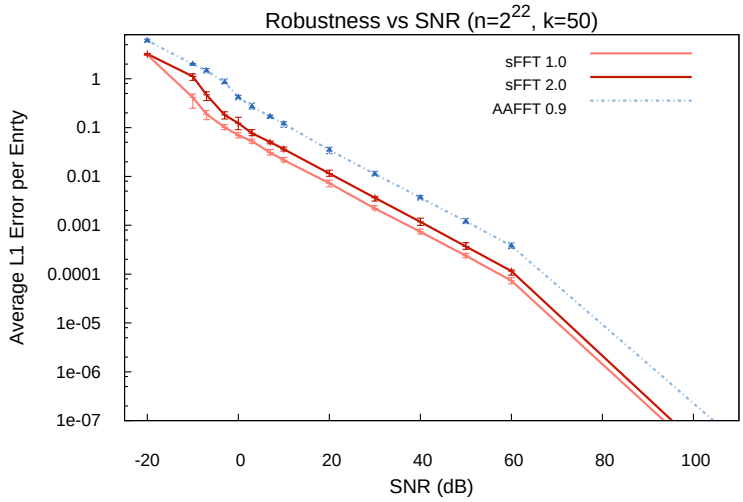


Figure 4: **Robustness to Noise** ($n = 2^{22}, k = 50$)**.** The figure shows that both sFFT and AAFFT 0.9 are stable in the presence of noise but sFFT has lower errors.

Transactions on Information Theory, 52:489 – 509, 2006.

[Don06] D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289 – 1306, 2006.

[DRZ07] I. Daubechies, O. Runborg, and J. Zou. A sparse spectral method for homogenization multiscale problems. *Multiscale Model. Sim.*, 6(3):711–740, 2007.

[FJ] M. Frigo and S. G. Johnson. Fftw3.2.3. `http://www.fftw.org/`.

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

[GI10] A. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of IEEE*, 2010.

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

[GST08] A.C. Gilbert, M.J. Strauss, and J. A. Tropp. A tutorial on fast fourier sampling. *Signal Processing Magazine*, 2008.

[IGS07] M. A. Iwen, A. Gilbert, and M. Strauss. Empirical evaluation of a sub-linear time sparse dft algorithm. *Communications in Mathematical Sciences*, 5, 2007.

[Iwe08] M. Iwen. AAFFT (Ann Arbor Fast Fourier Transform). `http://sourceforge.net/projects/aafftannarborfa/`, 2008.

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

[Iwe10b] M.A. Iwen. Improved approximation guarantees for sublinear-time fourier algorithms. *Arxiv preprint arXiv:1010.0014*, 2010.

[KKL88] J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions. *FOCS*, 1988.

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

[LMN93] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. *Journal of the ACM (JACM)*, 1993.

[LVS11] Mengda Lin, A. P. Vinod, and Chong Meng Samson See. A new flexible filter bank for low complexity spectrum sensing in cognitive radios. *Journal of Signal Processing Systems*, 62(2):205–215, 2011.

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

[MNL10] Abdullah Mueen, Suman Nath, and Jie Liu. Fast approximate correlation for massive time-series data. In *SIGMOD Conference*, pages 171–182, 2010.

[O'D08] R. O'Donnell. Some topics in analysis of boolean functions (tutorial). *STOC*, 2008.

[OSB99] A. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time signal processing*. Upper Saddle River, N.J.: Prentice Hall. ISBN 0-13-754920-2., 1999.

[PS10] E. Porat and M. Strauss. Sublinear time, measurement-optimal, sparse recovery for all. *Manuscript*, 2010.

[Smi11] Julius O. Smith. *Spectral Audio Signal Processing, October 2008 Draft*. `http://ccrma.stanford.edu/~jos/sasp/`, accessed 2011-07-11. online book.