Recent Developments in the Sparse Fourier Transform

Anna C. Gilbert*
University of Michigan
annacg@umich.edu

Piotr Indyk†
MIT
indyk@mit.edu

Mark Iwen‡
Michigan State University
markiwen@math.msu.edu

Ludwig Schmidt
MIT
ludwigs@mit.edu

Abstract

The Discrete Fourier Transform (DFT) is a fundamental component of numerous computational techniques in signal processing and scientific computing. The most popular means of computing the DFT is the Fast Fourier Transform (FFT). However, with the emergence of big data problems, in which the size of the processed data sets can easily exceed terabytes, the "Fast" in Fast Fourier Transform is often no longer fast enough. In addition, in many big data applications it is hard to acquire a sufficient amount of data in order to compute the desired Fourier transform in the first place. The Sparse Fourier Transform (SFT) addresses the big data setting by computing a compressed Fourier transform using only a subset of the input data, in time *sub-linear* in the data set size. The goal of this article is to survey these recent developments, to explain the basic techniques with examples and applications in big data, to demonstrate trade-offs in empirical performance of the algorithms, and to discuss the connection between the SFT and other techniques for massive data analysis such as streaming algorithms and compressive sensing.

*Supported in part by NSF CIF 0910765 and NSF CCF 1161233.
†Supported in part by the Simons Foundation and NSF CCF 1065125.
‡Supported in part by NSA grant H98230-13-1-0275.

1 Introduction

The Discrete Fourier Transform (DFT) is one of the main mathematical workhorses of signal processing. The most popular approach for computing the discrete Fourier transform is the Fast Fourier Transform (FFT) algorithm. Invented in the 1960s, the FFT computes the frequency representation of a signal of size N in $O(N \log N)$ time. The FFT is widely used and was considered to be one of the most influential and important algorithmic developments of the 20th century. However, with the emergence of big data problems, in which the size of the processed data sets can easily exceed terabytes, the Fast Fourier Transform is no longer always fast enough. Furthermore, in many applications it is hard to acquire a sufficient amount of data to compute the desired Fourier transform. For example, in medical imaging, it is highly desirable to reduce the time that the patient spends in an MRI machine. This motivates the need for algorithms that can compute the Fourier transform in sub-linear time (in an amount of time that is considerably smaller than the size of the data), and that use only a subset of the input data. The Sparse Fourier Transform (SFT) provides precisely this functionality.

Developed over the last decade, sparse Fourier transform algorithms compute an approximation or compressed version of the DFT in time proportional to the sparsity of the spectrum of the signal (i.e., the number of dominant frequencies), as opposed to the length of the signal. The algorithms use only a small subset of the input data and run in time proportional to the sparsity or desired compression, considerably faster than in time proportional to the signal length. This is made possible by requiring that the algorithms report only the non-zero or large frequencies and their complex amplitudes, rather than a vector containing this information for all frequencies. Since most video, audio, medical images, spectroscopic measurements (e.g., NMR), GPS signals, seismic data, and many more massive data sets are compressible or are sparse, these results promise a significant practical impact in many big data domains.

The first algorithms of this type were designed for the Hadamard Transform; i.e., the Fourier transform over the Boolean cube [KM91, Lev93] (cf. [GL89]). Shortly thereafter, algorithms for the complex Fourier transform were discovered as well [Man92, GGI$^+$02, AGS03, GMS05]. The most efficient of those algorithms [GMS05], computed the DFT in time $k \log^{O(1)} N$, where k is the sparsity of the signal spectrum. All of the algorithms are randomized and have a constant probability of error. These developments were covered in [GST08].

Over the last few years, the topic has been the subject of extensive research, from the algorithmic [Iwe10, Iwe13, Aka10, HIKP12b, HIKP12a, HAKI12, LWC13, BCG$^+$12, GHI$^+$13, HKPV13, PR13, SHV13, IKP14], implementation [Sch13, WAPC$^+$13] and hardware [YRR$^+$12,

AHH+14] perspectives. These developments include the first *deterministic* algorithms that make *no* errors [Iwe10, Iwe13, Aka10], as well as algorithms that, given a signal with k-sparse spectrum, compute the non-zero coefficients in time $O(k \log N)$ [HIKP12a] or even $O(k \log k)$ [LWC13, PR13, GHI+13, SHV13]. The goal of this article is to survey these developments. We focus on explaining the basic components and techniques used in the aforementioned algorithms, coupled with illustrative examples and concrete applications. We do not cover the analysis of sampling rates and running times of the algorithms (the reader is referred to the original papers for proofs and analysis). However, we present an empirical analysis of the performance of the algorithms. Finally, we discuss the connection between the SFT and other techniques for massive data analysis such as streaming algorithms and compressive sensing.

2 Definitions

Let $F \in \mathbf{C}^{N \times N}$ be the discrete Fourier transform matrix of size N, defined entrywise by

$$F_{\omega,j} := \frac{e^{-2\pi i \omega j/N}}{N} \tag{1}$$

for $0 \leq \omega, j < N$. The DFT of a vector $\mathbf{f} \in \mathbf{C}^N$ is simply

$$\hat{\mathbf{f}} := F\mathbf{f}. \tag{2}$$

Equivalently, one may define $\hat{\mathbf{f}} \in \mathbf{C}^N$ componentwise by

$$\hat{f}_\omega = \frac{1}{N} \sum_{j=0}^{N-1} f_j \, e^{-2\pi i \omega j/N}. \tag{3}$$

In this section it will also be useful to consider the inverse of the DFT matrix above. Its entries are given by

$$F^{-1}_{j,\omega} := e^{2\pi i \omega j/N} \tag{4}$$

for $0 \leq \omega, j < N$. The inverse DFT of a vector $\hat{\mathbf{f}} \in \mathbf{C}^N$ is just

$$\mathbf{f} := F^{-1}\hat{\mathbf{f}} = F^{-1}\left(F\mathbf{f}\right). \tag{5}$$

This allows one to write $\mathbf{f} \in \mathbf{C}^N$ in terms of its Fourier components by the formula

$$f_j = \sum_{\omega=0}^{N-1} \hat{f}_\omega \, e^{2\pi i \omega j / N}. \tag{6}$$

3 Techniques

In this section we outline the basic components and techniques used in sparse FFT algorithms. We start from the simple case when the spectrum of the signal consists of, or is dominated by, only a *single* non-zero frequency. In this case we show that the position and the value of the non-zero frequency can be found using only two samples of the signal (if the signal is a pure tone without any noise) or a logarithmic number of samples (if the signal is contaminated by noise). These techniques are described in Section 3.1. Second, we address the general case, by reducing it to several sub-problems involving a single non-zero frequency. This is achieved by grouping subsets of Fourier space together into a small number of bins. In the simplest case, each bin corresponds to a frequency band, but other groupings are also possible. If each bin contains only a single frequency (i.e., if a frequency is *isolated*), then we can solve the problem separately for each bin using the techniques mentioned earlier. This leads to the sample complexity and the running time proportional to the number of bins, which is lower bounded by the sparsity parameter k. The binning techniques are described in Section 3.2. Finally, in Section 3.3, we show how to deal with spectra in which two non-zero frequencies are very close to each other, and thus cannot be easily isolated via binning. Specifically, we show how to permute the spectrum of the signal in a pseudo-random fashion, by pseudo-randomly permuting the time domain signal. Since the positions of the non-zero frequencies in the permuted signals are (pseudo)-random, they are likely to be isolated, and then recovered by the binning procedure. To ensure that all non-zero coefficients are recovered, the permutation and binning procedure is repeated several times, using fresh randomness every time.

All sparse FFT algorithms follow this general approach, as discussed in more detail in Section 4. However, they differ in the implementations of the specific modules, as well as in the methods they use for aggregating the information gathered from different invocations of the permutation and binning procedure.

3.1 Single Frequency Recovery

In the simplest possible case, a vector $\mathbf{f} \in \mathbf{C}^N$ contains a single pure frequency. In such a setting an SFT must be able to rapidly determine the single tone much more quickly than the $O(N \log N)$-time FFT. In this section we will illustrate several different techniques for accomplishing this fundamental task. In later sections, we will demonstrate techniques for filtering more general vectors in order to produce the type of single-frequency vectors considered here. For now, however, we will assume that our vector is of the following form:

$$f_j := \hat{f}_\omega \cdot e^{2\pi i \omega j / N}. \tag{7}$$

for a fixed $\omega \in \{0, 1, \ldots, N-1\}$.

3.1.1 Phase Encoding

Given a simple vector defined as in (7), one can quickly calculate ω by choosing $j \in \{0, \ldots, N-1\}$, and then computing the phase of

$$\frac{f_{j+1}}{f_j} = e^{2\pi i \omega / N} = \cos\left(\frac{2\pi\omega}{N}\right) + i \cdot \sin\left(\frac{2\pi\omega}{N}\right).^1 \tag{8}$$

Furthermore, once ω is known, $\hat{f}_\omega$ can be calculated by computing $f_j \cdot e^{-2\pi i \omega j / N}$. This procedure[2] effectively finds the DFT of the vector from (7) in $O(1)$-time by inspecting only 2 entries of $\mathbf{f}$.

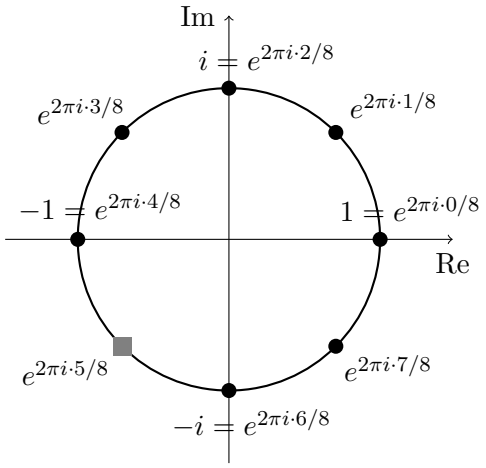
Although fast, this straightforward technique is not generally very robust to noise. Specifically, if $f_j := \hat{f}_\omega \cdot e^{2\pi i \omega j / N} + \epsilon_j$ for all j, the phases calculated from (8) can fail to yield ω unless $|\epsilon_j|$ is much smaller than $|\hat{f}_\omega|/N$. Hence, it is often necessary to use different techniques to find ω from (7).

3.1.2 A Binary Search Technique

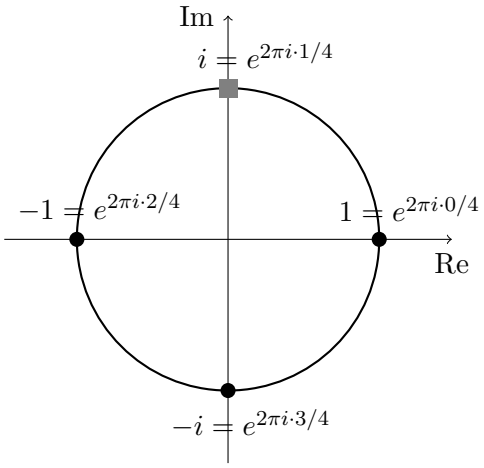
One means of learning ω from (7) in a more noise tolerant fashion is to perform the equivalent of a binary search for ω through the frequency domain. Many variants of such a search can be performed. In this subsection we will illustrate the most basic type of search for the example

[1] If $j+1 = N$, we set $f_{j+1} = f_0$. More generally, we will assume that indices are always taken modulo N when referring to entries of $\mathbf{f} \in \mathbf{C}^N$ below.

[2] The procedure, referred to as "the OFDM trick" in [HIKP12a], was also used in [LWC13]. It can also be seen as a very special case of the Prony Method [HKPV13].



(a) The initial search problem. Our goal is to locate the hidden frequency $\omega = 5$, i.e., the gray square at $e^{2\pi i\omega/8}$. In the first stage of the binary search, we determine that the hidden frequency lies in the third quadrant.



(b) The simplified search problem associated with $\mathbf{f}'$, which is the second stage of the binary search. The unknown frequency $\omega = 5$ has been mapped to $\omega' = 1$ (i.e., the gray square) within a smaller search space.

Figure 1: Recovering a single frequency via a binary search (see Section 3.1.2). Both subfigures show the unit circle in the complex plane. The black dots indicate the potential locations of the dominant, hidden frequency, which is represented by the gray square.

vector

$$f_j := \hat{f}_\omega \cdot e^{2\pi i \cdot \omega j/8}. \tag{9}$$

Note that this is exactly (7) with $N = 8$. Here, ω is unknown. We begin knowing only that

$$\omega \in \{0, 1, 2, 3, 4, 5, 6, 7\}.$$

Our job is to find ω using 3 rounds of testing based on at most 6 entries of $\mathbf{f}$.

Our tests will be based on the following observations: If $0 < \omega < N/2 = 4$, then $e^{2\pi i\omega/8}$ will be closer to $i = e^{2\pi i 2/8}$ than to $-i = e^{2\pi i 6/8}$ (see Figure 1(a)). Conversely, if $\omega > N/2 = 4$, then $e^{2\pi i\omega/8}$ will be closer to $-i$ than to i. Similarly, $\omega < N/4 = 2$ or $\omega > 3N/4 = 6$ implies that $e^{2\pi i\omega/8}$ is closer to 1 than to -1, and $2 = N/4 < \omega < 3N/4 = 6$ implies that $e^{2\pi i\omega/8}$ will be closer to -1 than to 1.

During our first round of testing we will choose $j \in \{0, \ldots, 7\}$ and then test whether both

$$\left|f_j\right| \cdot \left|i - e^{2\pi i \cdot \omega/8}\right| = \left|i \cdot f_j - f_{j+1}\right| < \left|i \cdot f_j + f_{j+1}\right| = \left|f_j\right| \cdot \left|i + e^{2\pi i \cdot \omega/8}\right|, \tag{10}$$

and

$$\left|f_j\right| \cdot \left|1 - e^{2\pi i \cdot \omega/8}\right| = \left|f_j - f_{j+1}\right| < \left|f_j + f_{j+1}\right| = \left|f_j\right| \cdot \left|1 + e^{2\pi i \cdot \omega/8}\right| \tag{11}$$

are true. Note that (10) and (11) will be true if and only if

$$\left|e^{2\pi i\omega/8} - i\right| < \left|e^{2\pi i\omega/8} - (-i)\right|, \tag{12}$$

and

$$\left|e^{2\pi i\omega/8} - 1\right| < \left|e^{2\pi i\omega/8} - (-1)\right| \tag{13}$$

are true, respectively. Hence, (10) and (11) are simply testing which axes of the complex plane are best aligned with $e^{2\pi i\omega/8}$. If (10) holds true we may safely conclude that $\omega \notin \{5, 6, 7\}$ (recall Figure 1(a)). Otherwise, if (10) is false, we conclude that $\omega \notin \{1, 2, 3\}$. Similarly, (11) holding true implies that $\omega \notin \{3, 4, 5\}$, while (11) failing to hold implies that $\omega \notin \{0, 1, 7\}$.

Returning to our example (9), suppose that both (10) and (11) fail to hold. The first test (10) failing tells us that $\omega \notin \{1, 2, 3\}$, and the second failure tells us that $\omega \notin \{0, 1, 7\}$. Taken all together, then, the tests tell us that $\omega \in \{4, 5, 6\}$ in this case (i.e., we learn that $e^{2\pi i \cdot \omega/8}$ is in the third quadrant of the complex plane).

Having learned that $\omega \in \{4, 5, 6\}$ allows us to simplify the problem. In particular, we may now implicitly define a new vector

$$f'_j := e^{-2\pi i \cdot 4 \cdot (2j/8)} \cdot f_{2j} = \hat{f}_\omega \cdot e^{2\pi i \cdot (\omega - 4)j/4} \tag{14}$$

for all $0 \leq j < 4$. Note that $\mathbf{f}' \in \mathbf{C}^4$ was formed by (i) shifting the possible range for ω into the first quadrant (by multiplying $\mathbf{f}$ by $e^{-2\pi i \cdot 4j/8}$), and then (ii) discarding the odd entries. This effectively halves our initial problem: $\mathbf{f}' \in \mathbf{C}^4$ is a vector with one frequency, $\omega' = (\omega - 4) \in \{0, 1, 2\}$ (see Figure 1(b)). Our new goal is to find ω' using 2 entries of $\mathbf{f}'$ (i.e., 2 additional entries of $\mathbf{f}$).

Our second round of tests now proceedes exactly as before. We choose $j \in \{0, \ldots, 3\}$ and then consider both

$$\left|i \cdot f'_j - f'_{j+1}\right| < \left|i \cdot f'_j + f'_{j+1}\right|, \tag{15}$$

and

$$\left|f'_j - f'_{j+1}\right| < \left|f'_j + f'_{j+1}\right|. \tag{16}$$

As above, these tests will collectively determine the quadrant of the complex plane containing

$e^{2\pi i\cdot(\omega-4)/4}$.

Continuing our example, suppose that (15) is true and (16) is false. This means that $(\omega - 4) \in \{1, 2\}$ (i.e., we have ruled out 0 and 3). We can now implicitly form our last new vector for the third round of tests. In particular, we form $\mathbf{f}'' \in \mathbf{C}^2$ by

$$f''_j := e^{-2\pi i\cdot 1\cdot(2j/4)} \cdot f'_{2j} = \hat{f}_\omega \cdot (-1)^{(\omega-5)j} \tag{17}$$

for $j = 0, 1$. Note that we have once again formed our new vector by (i) shifting the possible values of $\omega' = (\omega - 4)$ to the first quadrant of the complex plain, and then (ii) discarding all odd entries of $\mathbf{f}'$. We now know that $\omega'' = (\omega - 5) \in \{0, 1\}$, and may decide which it is by testing $\mathbf{f}''$.

In particular, suppose that

$$\left|f''_j - f''_{j+1}\right| < \left|f''_j + f''_{j+1}\right| \tag{18}$$

holds for a $j \in \{0, 1\}$. Then, we conclude that

$$\omega - 5 = 0 \Rightarrow \omega = 5.$$

This concludes the description of the binary search procedure for identifying the non-zero frequency. Since we learn the position of the frequency bit by bit, the total number of samples used is $O(\log N)$. Furthermore, we note that the binary search is (relatively) robust to noise. Adding small perturbations to each entry of (9) will not stop us from determining that $\omega = 5$. Further details are given, e.g., in [GST08].

3.1.3 An Aliasing-Based Search

We will conclude our discussion of single-frequency recovery techniques with an example of a modified search method that takes advantage of natural aliasing phenomena (see, e.g., [Iwe10, Iwe13]). These ideas are of use when subsampling signals is easy to implement directly, or when N is a product of several smaller relatively prime integers. Suppose, for example, that

$$N = 70 = 2 \cdot 5 \cdot 7,$$

and let $\mathbf{a} \in \mathbf{C}^2$ be the 2-element sub-vector of $\mathbf{f}$ from (7) with

$$a_0 := f_0 = \hat{f}_\omega, \text{ and } a_1 := f_{N/2} = \hat{f}_\omega \cdot (-1)^\omega. \tag{19}$$

Calculating $\widehat{\mathbf{a}} \in \mathbf{C}^2$ using (3) we get that

$$\widehat{a}_0 = \hat{f}_\omega \cdot \frac{1 + (-1)^\omega}{2}, \tag{20}$$

and

$$\widehat{a}_1 = \hat{f}_\omega \cdot \frac{1 + (-1)^{\omega+1}}{2}. \tag{21}$$

Note that since ω is an integer, exactly one element of $\widehat{\mathbf{a}}$ will be non-zero. If $\widehat{a}_0 \neq 0$ then we know that $\omega \equiv 0$ modulo 2. On the other hand, $\widehat{a}_1 \neq 0$ implies that $\omega \equiv 1$ modulo 2.

In this same fashion we may use several potentially aliased FFTs in parallel to discover ω modulo 5, and 7, since they both also divide N. Once we have collected these moduli we can reconstruct ω via the Chinese Remainder Theorem (CRT).

Theorem 1. CHINESE REMAINDER THEOREM: *Any integer x is uniquely specified modulo N by its remainders modulo m relatively prime integers $p_1, \ldots, p_m$ as long as $\prod_{l=1}^{m} p_l \geq N$.*

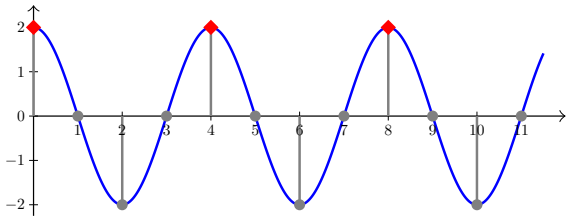
To finish our example, suppose that we have used four FFT's on sub-vectors of $\mathbf{f}$ of size 2, 5, and 7 to determine that $\omega \equiv 1 \bmod 2$, $\omega \equiv 4 \bmod 5$, and $\omega \equiv 3 \bmod 7$, respectively. Using that $\omega \equiv 1 \bmod 2$ we can see that $\omega = 2 \cdot a + 1$ for some integer a. Using this new expression for ω in our second modulus we get

$$(2 \cdot a + 1) \equiv 4 \bmod 5 \Rightarrow a \equiv 4 \bmod 5.$$

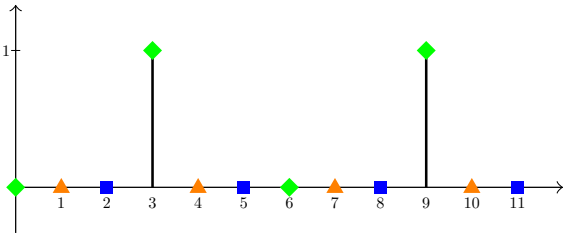
Therefore, $a = 5 \cdot b + 4$ for some integer b. Substituting for a we get that $\omega = 10 \cdot b + 9$. By similar work we can see that $b \equiv 5 \bmod 7$ after considering ω modulo 7. Hence, $\omega = 59$ by the CRT. As an added bonus we note that our three FFTs will have also provided us with three different estimates of $\hat{f}_\omega$.

The end result is that we have used significantly fewer than 70 entries to determine both ω and $\hat{f}_\omega$. Using the CRT we read only $2 + 5 + 7 = 14$ entries of $\mathbf{f}$. In contrast, a standard FFT would have had to process all 70 entries in order to compute $\hat{\mathbf{f}}$. This CRT-based single frequency method also reduces the required computational effort. Of course, a single frequency signal is incredibly simple. Vectors with more than 1 non-zero Fourier coefficient are much more

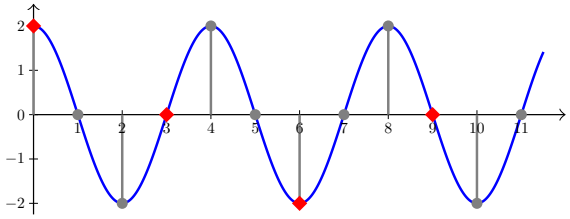
difficult to handle since frequency moduli may begin to collide modulo various numbers. In the next section we will discuss methods for removing this difficulty.



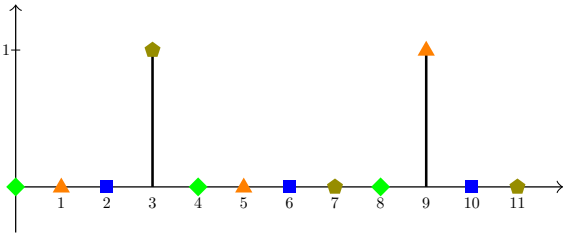
(a) Three equally spaced subsamples (the red diamonds)



(b) Three frequency bins, one for each residue modulo three



(c) Four equally spaced subsamples (the red diamonds)



(d) Four frequency bins, one for each residue modulo four

Figure 2: A demonstration of aliasing-based filtering. The vector under consideration has entries $f_j := \cos(2\pi \cdot 3 \cdot j/12)$ for $j = 0, \ldots, 11$. Note that $\hat{\mathbf{f}} \in \mathbf{C^{12}}$ contains only two nonzero entries: $\hat{f}_3 = 1$, and $\hat{f}_9 = 1$. Figure 2(a) marks the entries of a sub-vector, $\mathbf{a} \in \mathbf{R}^3$, of $\mathbf{f}$. Its DFT, $\hat{\mathbf{a}}$, also has three entries, each corresponding to a different subset of $\hat{\mathbf{f}}$. Each subset is labeled using a different symbol in figure 2(b). Note that both nonzero entries of $\hat{\mathbf{f}}$ fall into the same subset – both are labeled with a green diamond. Figure 2(c) marks the entries of another sub-vector of $\mathbf{f}$ consisting of four entries. Its DFT partitions the entries of $\hat{\mathbf{f}}$ into four subsets (see figure 2(d)). In this case the two nonzero entries of $\hat{\mathbf{f}}$ fall into different subsets: one is labeled with a pentagon, and the other with a triangle.

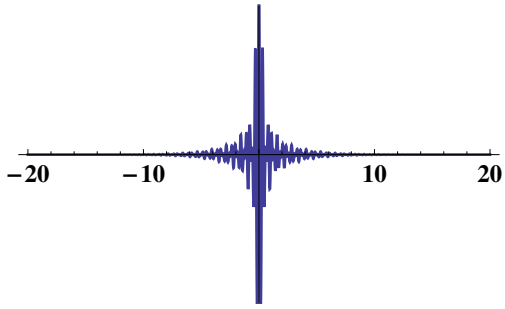
3.2 Filtering to Isolate Frequencies

We will begin our discussion of filtering by extending our aliasing-based frequency identification ideas from the last section. In this example we assume that our vector $\mathbf{f}$ has length twelve, with entries given by $f_j := \cos(2\pi \cdot 3 \cdot j/12)$ for $j = 0, \ldots, 11$. Note that this means $\hat{\mathbf{f}} \in \mathbf{C^{12}}$ has two nonzero entries: $\hat{f}_3 = 1$, and $\hat{f}_9 = 1$. Our objective is to learn the location and Fourier coefficient of each of them by reading fewer than 12 entries of $\mathbf{f}$.

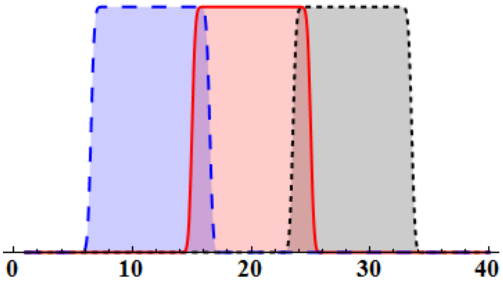
Preceding according to the last section, we might try to learn the two frequencies by looking at the three element sub-vector of $\mathbf{f}$, $\mathbf{a} \in \mathbf{R}^3$, given by $a_j := f_{4j}$ for $j = 0, 1, 2$ (see Figure 2(a)). Unfortunately, we will fail to learn anything about the individual entries of $\hat{\mathbf{f}}$ this way because

both of its nonzero DFT entries are congruent to 0 modulo 3 (see Figure 2(b)).[3] In particular, we will only see that $\hat{a}_0 = \hat{f}_0 + \hat{f}_3 + \hat{f}_6 + \hat{f}_9 = 2$, and that $\hat{a}_1 = \hat{a}_2 = 0$. If all we know is that $\hat{\mathbf{f}}$ contains at most two nonzero entries, we are unable to determine its nonzero entries using this information. The problem is that the two nonzero entries of $\hat{\mathbf{f}}$ have *collided* modulo 3 (i.e., they are both congruent to the same residue modulo 3).

Note, however, that the CRT guarantees that the two nonzero entries of $\hat{\mathbf{f}}$ can not also collide modulo $4 = 12/3$. If a new sub-array of $\mathbf{f}$ is created using four equally spaced entries (see Figure 2(c)), its DFT will separate the two nonzero entries of $\hat{\mathbf{f}}$ (see Figure 2(d)). The end result is that two sub-vectors will always reveal the locations of the nonzero entries of $\hat{\mathbf{f}}$, as long as $\hat{\mathbf{f}}$ has at most two nonzero entries. More generally, one can use similar ideas in order to learn the k largest magnitude entries of $\hat{\mathbf{f}}$ from the results of a small number of aliased DFTs of sub-vectors of $\mathbf{f}$.



(a) The filter function



(b) Fourier Space: Three modulations of the filter

Figure 3: A filter function with small (effective) support, and several frequency bins resulting from different modulations of the filter. The filter is a product of a sinc function with a Gaussian (see figure 3(a)). The Fourier transform of the filter is a characteristic (i.e., box) function convolved with a Gaussian. Three translates of the Fourier transform – each produced by a different modulation of the filter function – are graphed in figure 3(b). Note that modulations of the filter can be used to (effectively) regroup the Fourier spectrum into a small number of (essentially) disjoint frequency bins.

In the continuous setting, one can view the preceding discussion as a demonstration of how a relatively small set of spike-train filters can be used in order to separate important frequencies from one another in Fourier space. This turns out to be a fruitful interpretation. This perspective motivates the development of other types of filters which, when modulated (i.e., shifted in Fourier space) a few times, can be used to group different subsets of Fourier space together into a small number of shorter intervals, or "bins" (see Figure 3). If the most

[3]Recall that $\hat{a}_\omega$ is the sum of all Fourier coefficients whose indices are congruent to ω modulo 3.

important frequencies in a function, f, are uniformly spread over a given interval of Fourier space, one will be likely to isolate them from one another by convolving f with a few different modulations of such a filter. This effectively "bins" the Fourier coefficients of $\hat{f}$ into different frequency bins. Once an important frequency is isolated in a filtered version of f, the methods from the last subsection can then be used to recover it via, e.g., a modified binary search.

Note that a good continuous filter can be periodized and discretized for use as part of a discrete SFT. One generally does so in order to design a discrete filter that is highly sparse in time (i.e., that is "essentially zero" everywhere, except for a small number of time-domain entries). This allows fast convolution calculations to be performed with the filter during frequency binning. This is crucial since these convolutions are used to repeatedly compute time samples from filtered versions of $\hat{\mathbf{f}}$ during each modified binary search for an important frequency. Furthermore, the DFT of the discrete filter should also have a special structure in order to aid in the construction of good, low leakage[4], frequency filters. Suppose, for example, that one wants to isolate the k-largest entries of $\hat{\mathbf{f}}$ from one another. To accomplish this, one should use a filter whose DFT looks like a characteristic function on $\{0, \ldots, O(N/k)\} \subset [0, N) \cap \mathbf{Z}$ (i.e., on a $O(1/k)$-fraction of the "discrete Fourier spectrum" of $\mathbf{f}$). The filter can then be modulated $O(k)$ times in order to create a filter bank with $O(k)$ approximate "pass regions", each of size $O(N/k)$, that collectively tile all of $[0, N) \cap \mathbf{Z}$. These pass regions form the frequency bins discussed above (recall Figure 3(b)). To date several different types of filters have been utilized in sparse FFTs, including Gaussians [HIKP12b], indicator functions [GGI+02, GMS05], spike trains [Iwe10, Iwe13, GHI+13, PR13], and Dolph-Chebyshev filters [HIKP12a].

3.3 Randomly Binning Frequencies

As mentioned in the previous subsection, a filter function can be used to isolate the most important entries of $\hat{\mathbf{f}}$ from one another when they are sufficiently well separated. Unfortunately, an arbitrary vector $\mathbf{f} \in \mathbf{C^N}$ will not generally have a DFT with this property. The largest magnitude entries of $\hat{\mathbf{f}}$ can appear anywhere in principle. One can compensate for this problem, however, by pseudo-randomly permuting $\hat{\mathbf{f}}$ so that it "looks" uniformly distributed. As long as the permutation is reversible, any information gathered from the permuted vector can then be directly translated into information about the original vector's DFT, $\hat{\mathbf{f}}$.

Perhaps the easiest means of randomly permuting $\mathbf{f} \in \mathbf{C^N}$, and therefore $\hat{\mathbf{f}}$, is to use two basic properties of Fourier transform: the scaling property, stating that for $a_j = f_{cj}$ we have

[4]We say that a frequency filter leaks if it has non-zero values at frequencies other than our desired values.

$\hat{a}_j = \hat{f}_{c^{-1}j}$ (where c^{-1} is the inverse of c modulo N, assuming it exists); and the modulation property, stating that for $a_j = e^{2\pi i \cdot b \cdot j/N} \cdot f_j$ we have $\hat{a}_j = \hat{f}_{j-b}$. We proceed by choosing two random integers $b, c \in [0, N)$, and defining $\mathbf{a} \in \mathbf{C^N}$ as

$$a_j := e^{2\pi i \cdot b \cdot j/N} \cdot f_{c \cdot j} \tag{22}$$

for $j = 0, \ldots, N-1$. It can be seen that $\hat{\mathbf{a}}$ is a permuted version of $\hat{\mathbf{f}}$, as the entry $\hat{f}_\omega$ appears in $\hat{\mathbf{a}}$ as entry $(\omega \cdot c + b) \bmod N$. Note that permutations of this form are not fully random, even though b and c were selected randomly. Nevertheless, they are "random enough" for our purposes. In particular, the probability that any two non-zero coefficients land close to each other can be shown to be small.

4 The Prototypical SFT

In the simplest setting, a sparse FFT is a method which is designed to approximately compute the discrete Fourier transform (DFT) of a vector $\mathbf{f} \in \mathbf{C}^N$ as quickly as possible under the presumption that the result, $\hat{\mathbf{f}} \in \mathbf{C}^N$, will be sparse, or compressible. Here *compressible* means that $\hat{\mathbf{f}}$ will have a small number of indices (i.e., frequencies) whose entries (i.e., Fourier coefficients) have magnitudes that are large compared to the Euclidean norm of $\hat{\mathbf{f}}$ (i.e., the energy of $\hat{\mathbf{f}}$). Sparse FFTs improve on the runtime of traditional FFTs for such Fourier sparse signals by focussing exclusively on identifying energetic frequencies, and then estimating their Fourier coefficients. This allows sparse FFTs to avoid "wasting time" computing the Fourier coefficients of many insignificant frequencies.

Although several different sparse FFT variants exist, they generally share a common three stage approach to computing the sparse DFT of a vector: Briefly put, all sparse FFTs (repeatedly) perform some version of the three following steps:

1. identification of frequencies whose Fourier coefficients are large in magnitude (typically a randomized sub-routine),
2. accurate estimation of the Fourier coefficients of the frequencies identified in the first step, and
3. subtraction of the contribution of the partial Fourier representation computed by the first two steps from the entries of $\mathbf{f}$ before any subsequent repetitions.

Generally, each repetition of the three stages above is guaranteed to gather a substantial fraction

of the energy present in $\hat{\mathbf{f}}$ with high probability. Subtracting the located coefficients from the signal effectively improves the spectral sparsity of the given input vector, $\mathbf{f}$, from one repetition to the next. The end result is that a small number of repetitions will gather (almost) all of the signal energy with high probability, thereby accurately approximating the sparse Fourier transform, $\hat{\mathbf{f}}$, of the given vector $\mathbf{f}$.

Consider, for example, a vector $\mathbf{f}$ whose DFT has 100 nonzero entries (i.e., 100 nonzero Fourier coefficients). The first round of the three stages above will generally find and accurately estimate a large fraction of these entries (e.g., three fifths of them, or 60 terms in this case). The contributions of the discovered terms are then subtracted off of the remaining samples. This effectively reduces the number of nonzero entries in $\hat{\mathbf{f}}$, leaving about 40 terms in the current example. The next repetition of the three stages is now executed as before, but with the smaller effective sparsity of 40. Eventually all nonzero entries of $\hat{\mathbf{f}}$ will be found and estimated after a few repetitions with high probability. We will now consider each of the three stages mentioned above in greater detail.

4.1 Stage (i): Identifying Frequencies

Stage (i) of each repetition, which identifies frequencies whose Fourier coefficients are large in magnitude, is generally the most involved of the three repeated stages mentioned above. It usually consists of several ingredients, including: randomly sampling $\mathbf{f}$ in order to randomly permute its DFT, filtering to separate the permuted Fourier coefficients into different frequency bands, and estimating the energy in subsets of each of the aforementioned frequency bands. Many of these ingredients are illustrated with concrete examples in Section 3 above. Our objective now is to understand the general functionality of the identification stage as a whole.

Roughly speaking, stage (i) works by randomly binning the Fourier coefficients of $\mathbf{f}$ into a small number of "bins" (i.e., frequency bands) , and then performing a single frequency recovery procedure (as described in Section 3.1) within each bin in order to find any energetic frequencies that may have been isolated there. The randomness is introduced into the Fourier spectrum of $\mathbf{f} \in \mathbf{C}^N$ by randomly subsampling its entries (see Section 3.3). This has the effect of randomly permuting the entries of $\hat{\mathbf{f}}$. The resulting "randomized version of $\hat{\mathbf{f}}$" is then binned via a filter bank. (i.e., as discussed in Section 3.2). Because $\hat{\mathbf{f}}$ is approximately sparse, each "frequency bin" is likely to receive exactly one relatively large Fourier coefficient. Each such isolated Fourier coefficient is then identified by using one of the procedures described in Section 3.1. The collection of frequencies discovered in each different bin is then saved to be

analyzed further during the estimation stage (ii).

4.2 Stage (ii): Estimating Coefficients

Recall that stage (ii) involves estimating the Fourier coefficient, $\hat{f}_\omega$, of each frequency ω identified during stage (i) of the sparse FFT. In the simplest case, this can be done for each such ω by using $L \ll N$ independent and uniformly distributed random samples from the entries of $\mathbf{f}$, $f_{u_1}, \ldots, f_{u_L}$, in order to compute the estimator

$$\hat{f}'_\omega := \frac{1}{L} \cdot \sum_{l=1}^{L} f_{u_l} e^{-2\pi i \cdot \omega \cdot u_l / N}. \tag{23}$$

Note that $\hat{f}'_\omega$ is an unbiased estimator for $\hat{f}_\omega$ (i.e., $\mathbf{E}\left[\hat{f}'_\omega\right] = \hat{f}_\omega$) whose variance is $O(\|\hat{\mathbf{f}}\|_2^2/L)$. Thus, the estimator will approximate $\hat{f}_\omega$ to high (relative) precision with high probability whenever $|\hat{f}_\omega|^2$ is large compared to $\|\hat{\mathbf{f}}\|_2^2/L$. In slightly more complicated scenarios, the estimates might come "for free" as part of the identification stage (see Section 3.2 for an example).

4.3 Stage (iii): Repeating

A naive implementation of stage (iii) is even more straightforward than stage (ii). Suppose that

$$\left\{ \hat{f}'_{\omega_m} \;\middle|\; m = 1, \ldots, k \right\} \subset \mathbf{C}$$

is the approximate sparse DFT discovered for $\mathbf{f}$ during stages (i) and (ii) of the current repetition of our sparse FFT. Here, $\omega_1, \ldots, \omega_k$ are the frequencies identified during stage (i), while $\hat{f}'_{\omega_1}, \ldots, \hat{f}'_{\omega_m}$ are the estimates for their Fourier coefficients found during stage (ii) (e.g., via (23)). In future iterations of stages (i) through (iii), one can simply replace each sampled entry of $\mathbf{f}$, f_j, with

$$f_j - \sum_{m=1}^{k} \hat{f}'_{\omega_m} e^{2\pi i \cdot \omega_m \cdot j / N}. \tag{24}$$

If the entries of $\mathbf{f}$ to be used during each iteration of the three stages have been predetermined, which is often the case, (24) can be used to update them all at once. These "updated samples" are then used in the subsequent repetitions of the three stages.

One of the primary purposes of stage (iii) is to avoid mistakenly identifying insignificant frequencies as being energetic. Suppose, for example, that a small number of erroneous Fourier coefficients are identified during the j^{th} repetition of the first two stages. Then, subtracting their contribution from the original signal samples during the third stage will effectively add

them as new, albeit erroneous, energetic Fourier coefficients in $\hat{\mathbf{f}}$. This, in turn, allows them to be corrected in subsequent repetitions of the first two stages. Hence, stage three allows errors (assuming they are rare) to be corrected in later repetitions.

In contrast, some SFT methods [Iwe10, Iwe13, HIKP12b] perform only stages (i) and (ii) without any stage (iii). These methods identify all the energetic frequencies in stage (i) and then estimate their Fourier coefficients in stage (ii), completely in only one iteration. Of course, such methods can also mistakenly identify significant frequencies as being energetic during stage (i). Such mistaken frequencies are, however, generally discovered as being insignificant by these methods later, during stage (ii), when their Fourier coefficients are estimated.

5 Empirical Evaluation

In this section we compare several existing SFT implementations to FFTW, a fast implementation of the standard FFT, in order to demonstrate the computational gains that recent SFTs can provide over the standard FFT when dealing with Fourier-compressible signals. To this end, we consider the following algorithms and implementations.

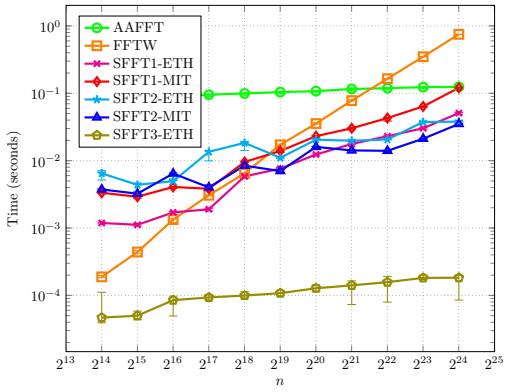
- FFTW: base line implementation of the standard FFT.
- AAAFT: An implementation of [GMS05].
- SFFT1-MIT, SFFT2-MIT: implementations of the algorithms in [HIKP12b] by the authors
- SFFT1-ETH, SFFT2-ETH: implementations of the algorithms in [HIKP12b] given in [Sch13]
- SFFT3-ETH: implementation of a variant of the algorithm in [HIKP12a] given in [Sch13]

All implementations are freely available.[5] The SFT variants considered herein are limited to those which compute the DFT of a vector (i.e., (2)). Additional experiments involving other existing SFT variants which sample continuous functions, as well as additional experiments demonstrating noise tolerance and sampling complexity, can be found here.[6]

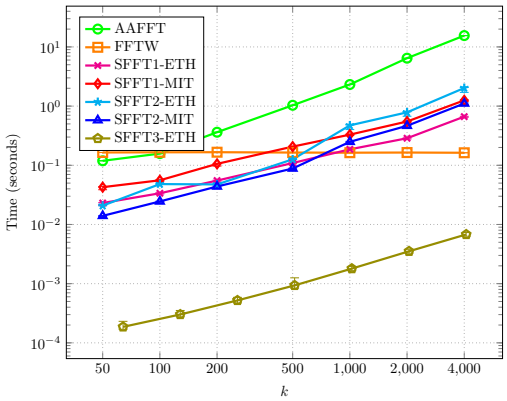
Figure 4 plots the runtimes of these SFTs against FFTW for various sparsity levels, k, and vector lengths, N. The algorithms were run on randomly generated vectors of length N whose DFTs were k-sparse (containing k ones in randomly chosen locations) for varying values of k and N. For each pair of values of k and N, the parameters of the (randomized) algorithms were

[5] FFTW is available at **http://www.fftw.org**. AAFFT, as well as several significantly faster sampling-based SFTs, are available at **http://sourceforge.net/projects/aafftannarborfa/**. The ETH implementations are available at **http://www.spiral.net/software/sfft.html**. All other SFT implementations are available at **http://groups.csail.mit.edu/netmit/sFFT/**.

[6] **https://github.com/ludwigschmidt/sft-experiments**



(a) Runtime as a function of the signal length N, for $k = 50$



(b) Runtime as a function of the signal length k, for $N = 2^{22}$

Figure 4: Running time plots for several algorithms and implementations of sparse FFT

optimized in order to minimize the running time while ensuring that the empirical probability of correct recovery was greater than 0.9.

6 Applications

In this section we give an overview of some of the data-intensive applications of sparse FFT algorithms and techniques that emerged over the last few years. These applications involve, for example, GPS receivers, cognitive radios, and, more generally, any analog signal that we wish to digitize. It is these applications that we focus on as they highlight the role of sparse FFT algorithms in the signal processing of large data.

GPS synchronization. In the (simplified)[7] GPS synchronization problem, we are given a (pseudorandom) code, corresponding to a particular satellite. The satellite repeatedly transmits the code. Furthermore, we are given a signal recorded by a GPS receiver, which consists of a window of the signal generated by the satellite, corrupted by noise. The goal is to align the code to the recorded signal, i.e., identify where the code starts and ends. To this end, the receiver computes the convolution of the code and the received signal and reports the shift that maximizes the correlation. This computation is typically done using the FFT: one applies the FFT to the code and the signal, computes the product of the outputs, and applies the inverse FFT to the product.

The paper [HAKI12] uses sparse FFT techniques to speed-up the process. The improvement is based on the following observation: since the output of the inverse FFT contains a single peak

[7]For simplicity, this description ignores certain issues such as the Doppler shift, etc. See [HAKI12] for details.

corresponding to the correct shift, the inverse step can be implemented using the sparse FFT. In fact, since $k = 1$, the algorithm is particularly simple, and relies on a simple aliasing filter. Furthermore, since the sparse inverse FFT algorithm uses only some of the samples of the product, it suffices to compute only those samples. This reduces the cost of the forward step as well. The experiments on real signals show that the new algorithm reduces the median number of multiplications by a factor of 2.2, or more if the value of the Doppler shift is known.

Spectrum sensing. The goal of a spectrum sensing algorithm is to scan the available spectrum and identify the "occupied" frequency slots. In many applications this task needs to be done quickly, since the spectrum changes dynamically. Unfortunately, scanning a GHz-wide spectrum is a highly power-consuming operation. To reduce the power and acquisition time, one can use a sparse Fourier transform to compute the frequency representation of a sparse signal without sampling it at full bandwidth. One such proposal was presented in [YG12] which uses a method of frequency identification similar to that described in Section 3.1.3. Another approach is presented in [HSA+14], which uses a sparse FFT procedure similar to that in [GHI+13, PR13]. It describes a prototype device using 3 software radios (USRPs), each sampling the spectrum at 50 MHz. The device captures 0.9 GHz, i.e., 6x larger bandwidth than the three USRPs combined.

Analog to digital converters (ADCs). The random binning procedure described in Section 3.3 forms the basis of the pulse-position-modulation (PPM) ADC presented in [YRR+12]. A prototype 9-bit random PPM ADC incorporating a pseudo-random sampling scheme is implemented as proof of concept. The approach leverages the energy efficiency of time-based processing.

Other applications include 2D Correlation Spectroscopy [SAH+13].

7 Conclusion

It is interesting to note that SFTs have a good deal in common with compressive sensing techniques. The latter generally aim to reduce sampling requirements as much as possible in order to recover accurate sparse approximations of frequency-compressable functions. SFTs, on the other hand, attempt to recover accurate sparse approximations of frequency-compressible functions as quickly as absolutely possible. By necessity, therefore, an SFT also cannot sample a function many times (i.e., since sampling takes time). As a result, SFTs also utilize a relatively small number of samples and, so, can be considered as compressive sensing algorithms. This puts SFTs into a broader spectrum of compressive sensing strategies that tradeoff additional

sampling for decreased computational complexity.

SFTs are also closely related to streaming (or sublinear) algorithms developed in the computer science community. Streaming algorithms aim to run in time considerably smaller than the time required to read the entire original data set, or signal. Hence, the streaming literature contains a rich set of tools for processing and approximating large data sets both quickly and accurately. Many of the techniques employed in existing SFTs are adapted to the Fourier setting from streaming techniques. For an overview of the links between these two topics, see [Ind13].

Acknowledgements

The authors would like to thank Haitham Hassanieh, Chinmay Hegde, David Lawlor, Eric Price and Jörn Schumacher for providing source code and very helpful discussions.

References

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

[AHH+14] O. Abari, E. Hamed, H. Hassanieh, A. Agarwal, D. Katabi, A. P. Chandrakasan, and V. Stojanovic. A 0.75-million-point Fourier-transform chip for frequency-sparse signals. *ISSCC*, 2014.

[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.

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

[GHI+13] B. Ghazi, H. Hassanieh, P. Indyk, D. Katabi, E. Price, and L. Shi. Sample-optimal average-case sparse Fourier transform in two dimensions. *Allerton*, 2013.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-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.

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

[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] S. Heider, S. Kunis, D. Potts, and M. Veit. A sparse Prony FFT. *SAMPTA*, 2013.

[HSA+14] H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi. Ghz-wide sensing and decoding using the sparse Fourier transform. *INFOCOM*, 2014.

[IKP14] P. Indyk, M. Kapralov, and E. Price. (Nearly) sample-optimal sparse Fourier transform. *SODA*, 2014.

[Ind13] P. Indyk. Sketching via hashing: from heavy hitters to compressed sensing to sparse Fourier transform. *PODS*, pages 87–90, 2013.

[Iwe10] M. A. Iwen. Combinatorial sublinear-time Fourier algorithms. *Foundations of Computational Mathematics*, 10:303–338, 2010.

[Iwe13] M.A. Iwen. Improved approximation guarantees for sublinear-time Fourier algorithms. *Applied And Computational Harmonic Analysis*, 34:57–82, 2013.

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

[Lev93] L.A. Levin. Randomness and non-determinism. *J. Symb. Logic*, 58(3):1102–1103, 1993.

[LWC13] D. Lawlor, Y. Wang, and A. Christlieb. Adaptive sub-linear time Fourier algorithms. *Advances in Adaptive Data Analysis*, 5(1), 2013.

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

[PR13] S. Pawar and K. Ramchandran. Computing a k-sparse n-length discrete Fourier transform using at most $4k$ samples and $O(k \log k)$ complexity. *ISIT*, 2013.

[SAH+13] L. Shi, O. Andronesi, H. Hassanieh, B. Ghazi, D. Katabi, and E. Adalsteinsson. Mrs sparse-fft: Reducing acquisition time and artifacts for in vivo 2d correlation spectroscopy. *ISMRM*, 2013.

[Sch13] J. Schumacher. High Performance sparse fast fourier transform. *Master thesis, Computer Science, ETH Zurich, Switzerland*, 2013.

[SHV13] R. Scheibler, S. Haghighatshoar, and M. Vetterli. A sparse sub-linear Hadamard transform. *Allerton*, 2013.

[WAPC+13] C. Wang, M. Araya-Polo, S. Chandrasekaran, A. St-Cyr, B. Chapman, and D. Hohl. Parallel sparse FFT. *SC Workshop on Irregular Applications: Architectures and Algorithms*, 2013.

[YG12] P. Yenduri and A. C. Gilbert. Compressive, collaborative spectrum sensing for wideband cognitive radios. *ISWCS*, pages 531–535, 2012.

[YRR+12] P. K. Yenduri, A. Z. Rocca, A. S. Rao, S. Naraghi, M. P. Flynn, and A. C. Gilbert. A low-power compressive sampling time-based analog-to-digital converter. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 2(3):502–515, 2012.