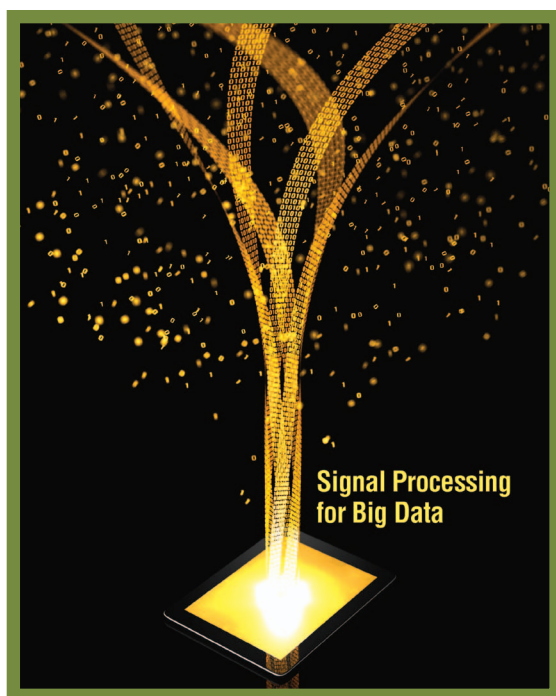


Recent Developments in the Sparse Fourier Transform



© ISTOCKPHOTO.COM/TAZYO4NORI

[A compressed Fourier transform for big data]

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 FFT is often no longer fast enough. In addition, in many big

data applications it is hard to acquire a sufficient amount of data 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 smaller than the data set size. The goal of this article is to survey these recent developments, explain the basic techniques with examples and applications in big data, demonstrate tradeoffs in empirical performance of the algorithms, and discuss the connection between the SFT and other techniques for massive data analysis such as streaming algorithms and compressive sensing.

INTRODUCTION

The DFT is one of the main mathematical workhorses of signal processing. The most popular approach for computing the DFT is the 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 FFT is not 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 a magnetic resonance imaging machine. This motivates the need for algorithms that can compute the Fourier transform in sublinear 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 SFT provides precisely this functionality.

Developed over the last decade, SFT 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 nonzero 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., nuclear magnetic resonance), global positioning system (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 [19], [20] (cf. [7]). Shortly thereafter, algorithms for the complex Fourier transform were discovered as well [1], [5], [8], [22]. The most efficient of those algorithms [8] computed the DFT in time $k \log^{(1)} N$, 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 [9].

Over the last few years, the topic has been the subject of extensive research, from the algorithmic [3], [4], [6], [10]–[13], [15], [17], [18], [21], [23], [26], implementation [25], [27], and hardware [2], [29] perspectives. These developments include the first deterministic algorithms that make no errors [3], [17], [18], as well as algorithms that, given a signal with k -sparse spectrum, compute the nonzero coefficients in time $O(k \log N)$ [11] or even $O(k \log k)$ [6], [21], [23], [26]. 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.

DEFINITIONS

Let $F \in \mathbb{C}^{N \times N}$ be the DFT matrix of size N , defined entrywise by

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

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

$$\hat{\mathbf{f}} := F\mathbf{f}. \quad (2)$$

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

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

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

$$F_{j,\omega}^{-1} := e^{2\pi i \omega j / N} \quad (4)$$

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

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

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

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

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 nonzero frequency. In this case we show that the position and the value of the nonzero 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 the section “Phase Encoding.” Second, we address the general case by reducing it to several subproblems involving a single nonzero 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 the section “Filtering to Isolate Frequencies.” Finally, in the section “Randomly Binning Frequencies,” we show how to deal with spectra in which two nonzero 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 pseudorandom fashion, by pseudorandomly permuting the time domain signal. Since the positions of the nonzero 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 nonzero 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 the section “The Prototypical SFT.” 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.

SINGLE-FREQUENCY RECOVERY

In the simplest possible case, a vector $\mathbf{f} \in \mathbb{C}^N$ 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 to produce the type of single-frequency vectors considered here. For now, however, we will assume that our vector is of the following form:

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

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

PHASE ENCODING

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

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

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 $\mathbf{f} \in \mathbb{C}^N$ below. 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 effectively finds the DFT of the vector from (7) in $O(1)$ -time by inspecting only two entries of $\mathbf{f}$. The procedure, referred to as the *OFDM trick* in [11], was also used in [21]. It can also be seen as a very special case of the Prony method [13].

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).

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 vector

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

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 three rounds of testing based on at most six 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 $j \in \{0, \dots, 7\}$ and then test whether both

$$\begin{aligned} |f_j| \cdot |i - e^{2\pi i \omega / 8}| &= |i \cdot f_j - f_{j+1}| < |i \cdot f_j + f_{j+1}| \\ &= |f_j| \cdot |i + e^{2\pi i \omega / 8}|, \end{aligned} \quad (10)$$

and

$$\begin{aligned} |f_j| \cdot |1 - e^{2\pi i \omega / 8}| &= |f_j - f_{j+1}| < |f_j + f_{j+1}| \\ &= |f_j| \cdot |1 + e^{2\pi i \omega / 8}| \end{aligned} \quad (11)$$

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

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

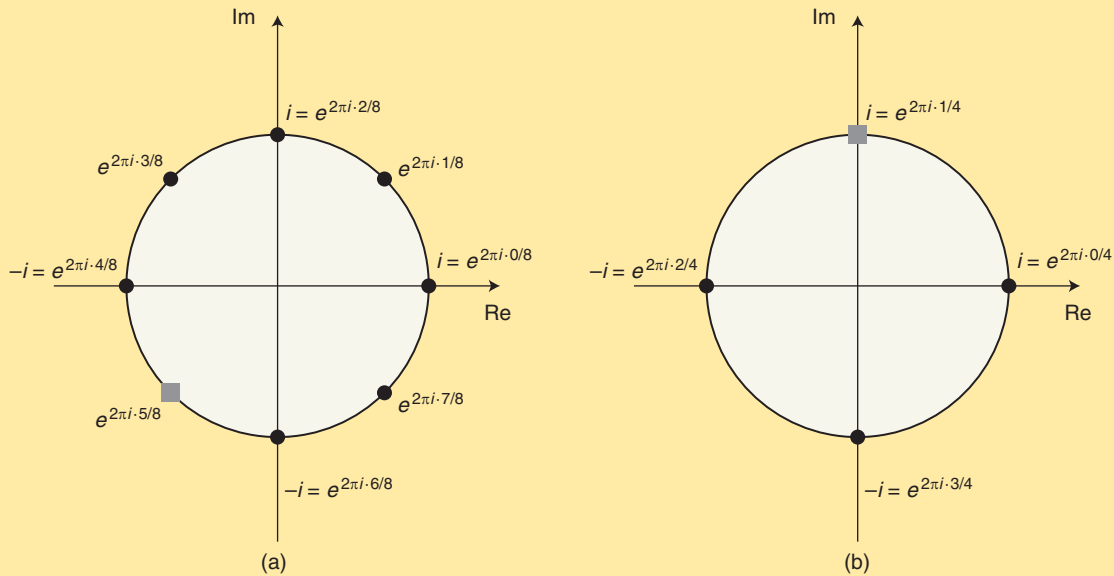
and

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

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 $\omega \notin \{5, 6, 7\}$ [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 $e^{2\pi i \omega / 8}$ 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



[FIG1] Recovering a single frequency via a binary search (see the section “A Binary Search Technique”). Both parts 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 at $e^{2\pi i \omega/8}$. (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 associated with f' , which is the second stage of the binary search. (b) The simplified search problem associated with 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.

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

for all $0 \leq j < 4$. Note that $f' \in \mathbb{C}^4$ was formed by 1) shifting the possible range for ω into the first quadrant (by multiplying f by $e^{-2\pi i \cdot 4j/8}$), and then 2) discarding the odd entries. This effectively halves our initial problem: $f' \in \mathbb{C}^4$ 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 two entries of f' (i.e., two additional entries of f).

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

$$|i \cdot f'_j - f'_{j+1}| < |i \cdot f'_j + f'_{j+1}|, \quad (15)$$

and

$$|f'_j - f'_{j+1}| < |f'_j + f'_{j+1}|. \quad (16)$$

As previously shown, 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 $f'' \in \mathbb{C}^2$ by

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

for $j = 0, 1$. Note that we have once again formed our new vector by 1) shifting the possible values of $\omega' = (\omega - 4)$ to the first quadrant of the complex plane, and then 2) discarding all odd entries of f' . We now know that $\omega'' = (\omega - 5) \in \{0, 1\}$, and may decide which it is by testing f'' .

In particular, suppose that

$$|f''_j - f''_{j+1}| < |f''_j + f''_{j+1}| \quad (18)$$

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 nonzero 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 [9].

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., [17]

and [18]). 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, e.g., that

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

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

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

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

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

and

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

Note that since ω is an integer, exactly one element of $\hat{\mathbf{a}}$ will be nonzero. If $\hat{a}_0 \neq 0$ then we know that $\omega \equiv 0$ modulo 2. On the other hand, $\hat{a}_1 \neq 0$ 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) (see “Theorem 1: Chinese Remainder Theorem”).

THEOREM 1: CHINESE REMAINDER THEOREM

Any integer x is uniquely specified modulo N by its remainders modulo m relatively prime integers $p_1, \dots, p_m$ as long as $\prod_{i=1}^m p_i \geq N$.

To finish our example, suppose that we have used four FFTs on subvectors of $\mathbf{f}$ of size 2, 5, and 7 to determine that $\omega \equiv 1 \pmod{2}$, $\omega \equiv 4 \pmod{5}$, and $\omega \equiv 3 \pmod{7}$, respectively. Using that $\omega \equiv 1 \pmod{2}$ 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

$$(2 \cdot a + 1) \equiv 4 \pmod{5} \Rightarrow a \equiv 4 \pmod{5}.$$

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 $b \equiv 5 \pmod{7}$ 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 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 one nonzero 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.

FILTERING TO ISOLATE FREQUENCIES

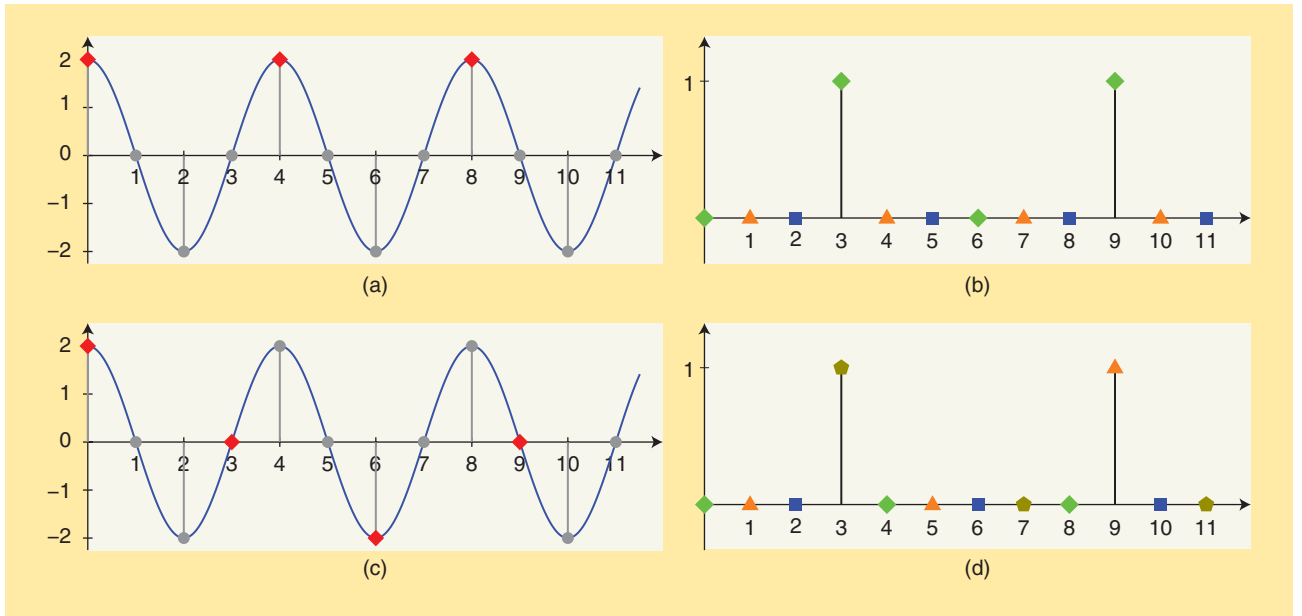
We will begin our discussion of filtering by extending our aliasing-based frequency identification ideas from the last section. In this example we assume that our vector $\mathbf{f}$ has length 12, with entries given by $f_j := \cos(2\pi \cdot 3 \cdot j/12)$ for $j = 0, \dots, 11$. Note that this means $\hat{\mathbf{f}} \in \mathbb{C}^{12}$ 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 $\mathbf{f}$.

Proceeding according to the last section, we might try to learn the two frequencies by looking at the three-element subvector of $\mathbf{f}$, $\mathbf{a} \in \mathbb{R}^3$, 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)]. Recall that $\hat{\mathbf{a}}_\omega$ is the sum of all Fourier coefficients whose indices are congruent to ω modulo 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 subarray 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 subvectors 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 to learn the k largest magnitude entries of $\hat{\mathbf{f}}$ from the results of a small number of aliased DFTs of subvectors of $\mathbf{f}$.

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 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 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 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



[FIG2] A demonstration of aliasing-based filtering. The vector under consideration has entries $f_j := \cos(2\pi \cdot 3 \cdot j/12)$ for $j = 0, \dots, 11$. Note that $\mathbf{f} \in \mathbb{C}^{12}$ contains only two nonzero entries: $f_3 = 1$, and $f_9 = 1$. (a) marks the entries of a subvector, $\mathbf{a} \in \mathbb{R}^3$, of $\mathbf{f}$. Its DFT, $\hat{\mathbf{a}}$, also has three entries, each corresponding to a different subset of $\mathbf{f}$. Each subset is labeled using a different symbol in (b). Note that both nonzero entries of $\mathbf{f}$ fall into the same subset—both are labeled with a green diamond. (c) marks the entries of another subvector of $\mathbf{f}$ consisting of four entries. Its DFT partitions the entries of $\mathbf{f}$ into four subsets [see (d)]. In this case the two nonzero entries of $\mathbf{f}$ fall into different subsets: one is labeled with a pentagon and the other with a triangle. (a) Three equally spaced subsamples (the red diamonds). (b) Three frequency bins, one for each residue modulo 3. (c) Four equally spaced subsamples (the red diamonds). (d) Four frequency bins, one for each residue modulo 4.

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 to aid in the construction of good, low-leakage, frequency filters. (We say that a frequency filter leaks if it has nonzero values at frequencies other than our desired values.) Suppose, e.g., 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 $\{0, \dots, O(N/k)\} \subset [0, N) \cap \mathbb{Z}$ (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 to create a filter bank with $O(k)$ approximate “pass regions,” each of size $O(N/k)$, that collectively tile all of $[0, N) \cap \mathbb{Z}$. These pass regions form the frequency bins discussed above [Figure 3(b)]. To date several different types of filters have been utilized in sparse FFTs, including Gaussians [12], indicator functions [5], [8], spike trains [6], [17], [18], [23], and Dolph–Chebyshev filters [11].

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 $\mathbf{f} \in \mathbb{C}^N$ 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 pseudorandomly 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 $\mathbf{f} \in \mathbb{C}^N$, and therefore $\hat{\mathbf{f}}$, is to use two basic properties of the Fourier transform: the scaling property, stating that for $a_j = f_{cj}$ we have $\hat{a}_j = \hat{f}_{c^{-1}j}$ (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 $\mathbf{a} \in \mathbb{C}^N$ as

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

for $j = 0, \dots, N-1$. 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 nonzero coefficients land close to each other can be shown to be small.

THE PROTOTYPICAL SFT

In the simplest setting, a sparse FFT is a method that is designed to approximately compute the DFT of a vector $\mathbf{f} \in \mathbb{C}^N$ as quickly as possible under the presumption that the result, $\hat{\mathbf{f}} \in \mathbb{C}^N$, 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 $\hat{f}$ (i.e., the energy of $\hat{f}$). Sparse FFTs improve on the runtime of traditional FFTs for such Fourier sparse signals by focusing 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 subroutine)
- 2) accurate estimation of the Fourier coefficients of the frequencies identified in the first step
- 3) subtraction of the contribution of the partial Fourier representation computed by the first two steps from the entries of f before any subsequent repetitions.

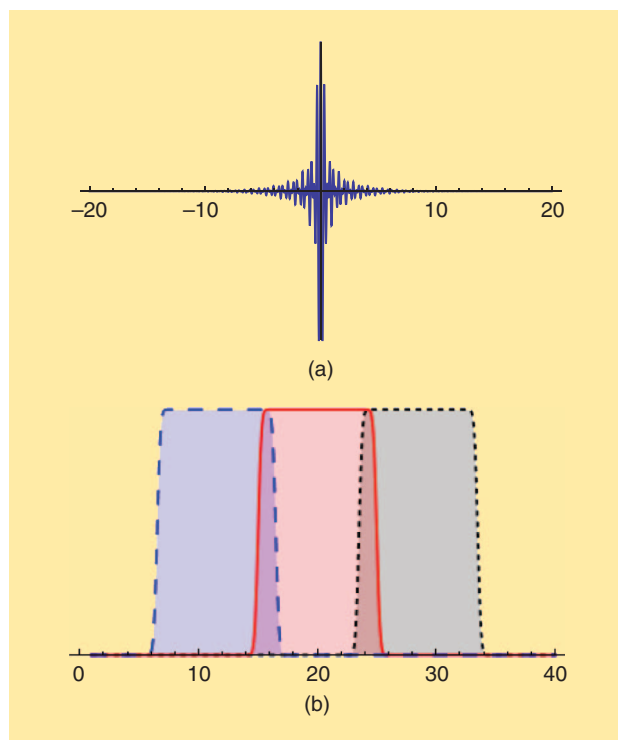
Generally, each repetition of the three stages above is guaranteed to gather a substantial fraction of the energy present in $\hat{f}$ with high probability. Subtracting the located coefficients from the signal effectively improves the spectral sparsity of the given input vector, f , from one repetition 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 SFT, $\hat{f}$, of the given vector f .

Consider, e.g., a vector 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 $\hat{f}$, 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 $\hat{f}$ 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.

STAGE 1: IDENTIFYING FREQUENCIES

Stage 1 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 f 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 the section “Techniques.” Our objective now is to understand the general functionality of the identification stage as a whole.

Roughly speaking, stage 1 works by randomly binning the Fourier coefficients of f into a small number of “bins” (i.e., frequency bands), and then performing a single frequency recovery procedure (as described in the section “Phase Encoding”) within each bin to find any energetic frequencies that may have been



[FIG3] 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 (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 (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. (a) The filter function. (b) The Fourier space: three modulations of the filter.

isolated there. The randomness is introduced into the Fourier spectrum of $f \in \mathbb{C}^N$ by randomly subsampling its entries (see the section “Randomly Binning Frequencies”). This has the effect of randomly permuting the entries of $\hat{f}$. The resulting “randomized version of $\hat{f}$ ” is then binned via a filter bank. (i.e., as discussed in the section “Filtering to Isolate Frequencies”). Because $\hat{f}$ 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 the section “Single Frequency Recovery.” The collection of frequencies discovered in each different bin is then saved to be analyzed further during the estimation stage 2.

STAGE 2: ESTIMATING COEFFICIENTS

Recall that stage 2 involves estimating the Fourier coefficient, $\hat{f}_\omega$, of each frequency ω identified during stage 1 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 f , $f_{u_1}, \dots, f_{u_L}$, to compute the estimator

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

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

STAGE 3: REPEATING

A naive implementation of stage 3 is even more straightforward than stage 2. Suppose that

$$\{\hat{f}'_{\omega_m} \mid m = 1, \dots, k\} \subset C$$

is the approximate sparse DFT discovered for f during stages 1 and 2 of the current repetition of our sparse FFT. Here, $\omega_1, \dots, \omega_k$ are the frequencies identified during stage 1, while $\hat{f}'_{\omega_1}, \dots, \hat{f}'_{\omega_m}$ are the estimates for their Fourier coefficients found during stage 2 [e.g., via (23)]. In future iterations of stages 1–3, one can simply replace each sampled entry of f , f_j , with

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

If the entries of 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 3 is to avoid mistakenly identifying insignificant frequencies as being energetic. Suppose, e.g., 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 $\hat{f}$. 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.

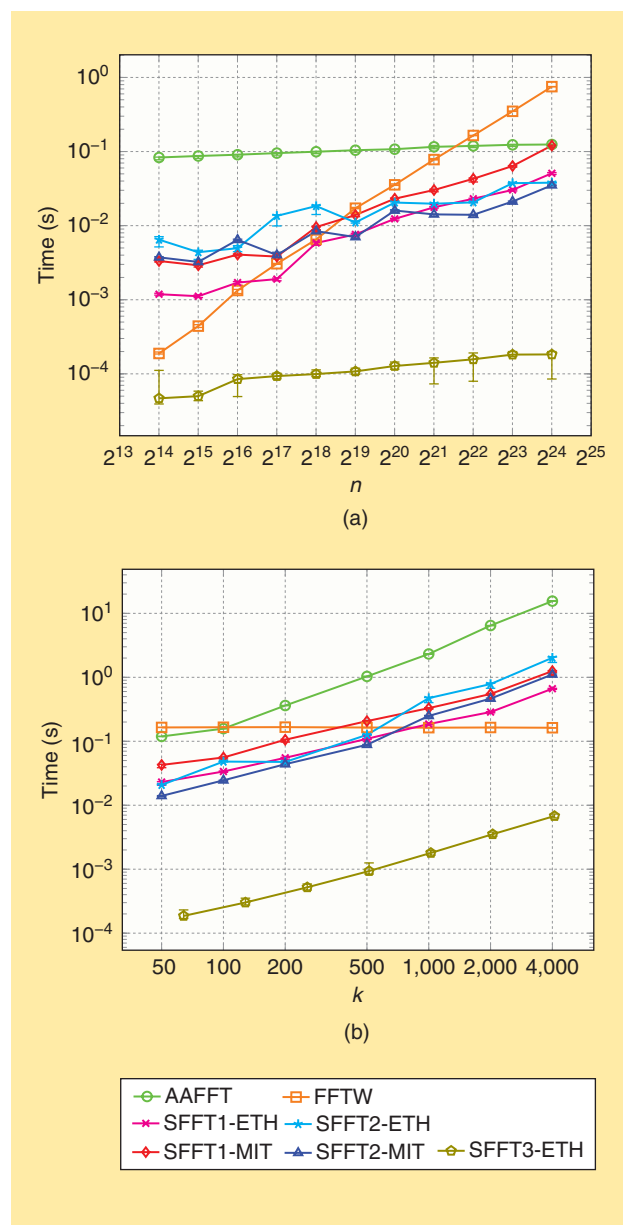
In contrast, some SFT methods [12], [17], [18] perform only stages 1 and 2 without any stage 3. These methods identify all the energetic frequencies in stage 1 and then estimate their Fourier coefficients in stage 2, completely in only one iteration. Of course, such methods can also mistakenly identify significant frequencies as being energetic during stage 1. Such mistaken frequencies are, however, generally discovered as being insignificant by these methods later, during stage 2, when their Fourier coefficients are estimated.

EMPIRICAL EVALUATION

In this section, we compare several existing SFT implementations to the fastest Fourier transform in the West (FFTW), a fast implementation of the standard FFT, 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
- AAFFT: an implementation of [8]
- SFFT1-MIT, SFFT2-MIT: implementations of the algorithms in [12] by the authors
- SFFT1-ETH, SFFT2-ETH: implementations of the algorithms in [12] given in [25]
- SFFT3-ETH: implementation of a variant of the algorithm in [11] given in [25].

All implementations are freely available. 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



[FIG4] Running time plots for several algorithms and implementations of sparse FFT. (a) Runtime as a function of the signal length N , for $k = 50$. (b) Runtime as a function of the signal length k , for $N = 2^{22}$.

available at <http://www.spiral.net/software/sfft.html>. All other SFT implementations are available at <http://groups.csail.mit.edu/netmit/sFFT/>. 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 that sample continuous functions, as well as additional experiments demonstrating noise tolerance and sampling complexity, can be found at <https://github.com/ludwigschmidt/sft-experiments>.

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

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, e.g., 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) GPS synchronization problem, we are given a (pseudorandom) code, corresponding to a particular satellite. (For simplicity, this description ignores certain issues such as the Doppler shift, etc. See [10] for details.) 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 [10] 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 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 an SFT to compute the frequency representation of a sparse signal without sampling it at full bandwidth. One such proposal was presented in [28], which uses a method of frequency identification similar to that described in the section “An Aliasing-Based Search.” Another approach is presented in [14], which uses a sparse FFT procedure similar to that in [6], [23]. It describes a prototype device using three software radios called Universal Software Radio Peripherals (USRPs), each sampling the spectrum at 50 MHz. The device captures 0.9 GHz, i.e., six times larger bandwidth than the three USRPs combined.

ANALOG TO DIGITAL CONVERTERS

The random binning procedure described in the section “Randomly Binning Frequencies” forms the basis of the pulse-position modulation (PPM) analog to digital converter presented in [29]. A prototype 9-bit random PPM analog to digital converter incorporating a pseudorandom sampling scheme is implemented as proof of concept. The approach leverages the energy efficiency of time-based processing.

OTHER APPLICATIONS

Other applications include 2-D correlation spectroscopy [24].

CONCLUSIONS

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 to recover accurate sparse approximations of frequency-compressible 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 [16].

ACKNOWLEDGMENTS

We would like to thank Haitham Hassanieh, Chinmay Hegde, David Lawlor, Eric Price, and Jörn Schumacher for providing source code and very helpful discussions. Anna C. Gilbert was supported in part by National Science Foundation (NSF) CIF 0910765 and NSF CCF 1161233. Piotr Indyk was supported in part by the Simons Foundation and NSF CCF 1065125. Mark Iwen was supported in part by NSA grant H98230-13-1-0275.

AUTHORS

Anna C. Gilbert (annacg@umich.edu) is a professor of mathematics at the University of Michigan. She was a postdoctoral fellow at AT&T Labs and Yale University in 1997. From 1998 to 2004, she was a member of technical staff at AT&T Labs. Her research interests include randomized algorithms with applications to harmonic analysis, signal and image processing, networking, and massive data sets. She has received several awards for her research on streaming algorithms and sparse Fourier transforms, including a Sloan Research Fellowship (2006), a National Science Foundation CAREER Award (2006), the National Academy of Sciences Award for Initiatives in Research (2008), and the SIAM Ralph E. Kleinman Prize (2013).

Piotr Indyk (indyk@theory.lcs.mit.edu) is a professor of electrical engineering and computer science at the Massachusetts Institute of Technology (MIT). He joined MIT in 2000 after earning a Ph.D. degree from Stanford University. His research interests lie in the design and analysis of efficient algorithms. Specific interests include high-dimensional computational geometry, sketching and streaming algorithms, sparse recovery, and compressive sensing. He received the Sloan Fellowship (2003), the Packard Fellowship (2003), and the Simons Investigator Award (2013). His work on sparse Fourier sampling was named to *MIT Technology Review* TR10 in 2012, while his work on locality-sensitive hashing has received the 2012 ACM Kanellakis Theory and Practice Award.

Mark Iwen (markiwen@math.msu.edu) earned a Ph.D. degree in applied and interdisciplinary mathematics from the University of Michigan in 2008. From 2008 to 2010, he was a postdoctoral fellow at the Institute for Mathematics and its Applications, and then moved to Duke University as a visiting assistant professor from 2010 until 2013. He has been an assistant professor at Michigan State University since the fall of 2013.

Ludwig Schmidt (ludwigs@mit.edu) is a Ph.D. student in the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology (MIT), where he works with Prof. Piotr Indyk in the Theory of Computation group. His interest lies in algorithms, with a focus on applications outside theoretical computer science. Specific interests include signal processing, especially compressive sensing and similarity search. Before coming to MIT, he received his bachelor's degree in computer science from the University of Cambridge in 2011.

REFERENCES

[1] A. Akavia, S. Goldwasser, and S. Safra, "Proving hard-core predicates using list decoding," in *Proc. Annu. IEEE Symp. Foundations of Computer Science*, vol. 44, pp. 146–159, 2003.

- [2] 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," in *IEEE Int. Solid-State Circuits Conf. Dig. Technical Papers (ISSCC)*, 2014.
- [3] A. Akavia, "Deterministic sparse Fourier approximation via fooling arithmetic progressions," in *Proc. 23rd Conf. Learning Theory (COLT)*, 2010, pp. 381–393.
- [4] P. Boufounos, V. Cevher, A. C. Gilbert, Y. Li, and M. J. Strauss, "What's the frequency, Kenneth? Sublinear Fourier sampling off the grid," in *Proc. RANDOM/APPROX*, 2012.
- [5] A. Gilbert, S. Guha, P. Indyk, M. Muthukrishnan, and M. Strauss, "Near-optimal sparse Fourier representations via sampling," in *Proc. 34th Annu. Symp. Theory of Computing (STOC)*, 2002.
- [6] B. Ghazi, H. Hassanieh, P. Indyk, D. Katabi, E. Price, and L. Shi, "Sample-optimal average-case sparse Fourier transform in two dimensions," in *Proc. Allerton*, 2013.
- [7] O. Goldreich and L. Levin, "A hard-core predicate for all one-way functions," in *Proc. 21st Annu. Symp. Theory of Computing (STOC)*, 1989, pp. 25–32.
- [8] A. Gilbert, M. Muthukrishnan, and M. Strauss, "Improved time bounds for near-optimal space Fourier representations," in *Proc. SPIE Conf. Wavelets*, 2005.
- [9] A. C. Gilbert, M. J. Strauss, and J. A. Tropp, "A tutorial on fast Fourier sampling," *IEEE Signal Processing Mag.*, 2008.
- [10] H. Hassanieh, F. Adib, D. Katabi, and P. Indyk, "Faster GPS via the sparse Fourier transform," in *Proc. 18th Annu. Int. Conf. Mobile Networking and Computing (MOBICOM)*, 2012.
- [11] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Near-optimal algorithm for sparse Fourier transform," in *Proc. 44th Annu. ACM Symp. Theory of Computing (STOC)*, 2012.
- [12] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Simple and practical algorithm for sparse Fourier transform," in *Proc. 23rd Annu. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2012.
- [13] S. Heider, S. Kunis, D. Potts, and M. Veit, "A sparse Prony FFT," in *Proc. 10th Int. Conf. Sampling Theory and Applications (SAMPAT)*, 2013.
- [14] H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi, "Ghz-wide sensing and decoding using the sparse Fourier transform," in *Proc. IEEE Conf. Computer Communications (INFOCOM)*, 2014.
- [15] P. Indyk, M. Kapralov, and E. Price, "(Nearly) sample-optimal sparse Fourier transform," in *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2014.
- [16] P. Indyk, "Sketching via hashing: From heavy hitters to compressed sensing to sparse Fourier transform," in *Proc. 32nd Symp. Principles of Database Systems (PODS)*, 2013, pp. 87–90.
- [17] M. A. Iwen, "Combinatorial sublinear-time Fourier algorithms," *Found. Computat. Math.*, vol. 10, pp. 303–338, 2010.
- [18] M. A. Iwen, "Improved approximation guarantees for sublinear-time Fourier algorithms," *Appl. Computat. Harm. Anal.*, vol. 34, pp. 57–82, 2013.
- [19] E. Kushilevitz and Y. Mansour, "Learning decision trees using the Fourier spectrum," in *Proc. 23rd Annu. Symp. Theory of Computing (STOC)*, 1991.
- [20] L. A. Levin, "Randomness and non-determinism," *J. Symb. Logic*, vol. 58, no. 3, pp. 1102–1103, 1993.
- [21] D. Lawlor, Y. Wang, and A. Christlieb, "Adaptive sub-linear time Fourier algorithms," *Adv. Adapt. Data Anal.*, vol. 5, no. 1, 2013.
- [22] Y. Mansour, "Randomized interpolation and approximation of sparse polynomials," in *Proc. 19th Int. Colloq. Automata, Languages and Programming (ICALP)*, 1992.
- [23] 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," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2013.
- [24] 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," in *Proc. Int. Society for Magnetic Resonance in Medicine Annu. Meeting & Exhibition (ISMRM)*, 2013.
- [25] J. Schumacher, "High performance sparse fast Fourier transform," Master's thesis, Computer Science, ETH Zurich, Switzerland, 2013.
- [26] R. Scheibler, S. Haghshatshoar, and M. Vetterli, "A sparse sub-linear Hadamard transform," in *Proc. Allerton*, 2013.
- [27] C. Wang, M. Araya-Polo, S. Chandrasekaran, A. St-Cyr, B. Chapman, and D. Hohl, "Parallel sparse FFT," in *Proc. SC Workshop on Irregular Applications: Architectures and Algorithms*, 2013.
- [28] P. Yenduri and A. C. Gilbert, "Compressive, collaborative spectrum sensing for wideband cognitive radios," in *Proc. Int. Symp. Wireless Communication Systems (ISWCS)*, 2012, pp. 531–535.
- [29] 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," *IEEE J. Emerg. Select. Topics Circuits Syst.*, vol. 2, no. 3, pp. 502–515, 2012.