SIGNAL APPROXIMATION VIA THE GOPHER FAST FOURIER TRANSFORM

By

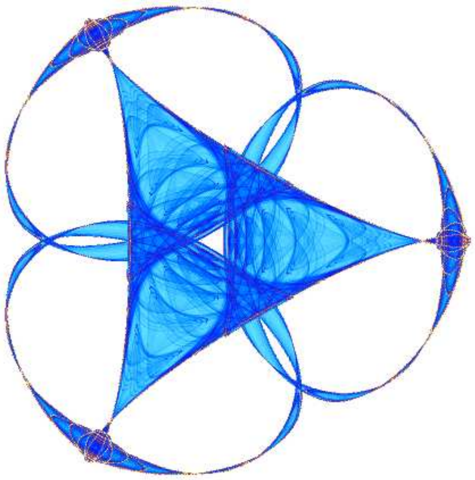
I. Ben Segal

and

M.A. Iwen

IMA Preprint Series # 2326

(July 2010)



Signal Approximation via the Gopher Fast Fourier Transform

I. Ben Segal[*] and M. A. Iwen[†]

[*]*School of Mathematics, University of Minnesota*
[†]*Institute for Mathematics and its Applications (IMA), University of Minnesota*

Abstract. We consider the problem of quickly estimating the best k-term Fourier representation for a given frequency-sparse band-limited signal (i.e., function) $f : [0,2\pi] \to \mathbb{C}$. In essence, this requires the identification of k of the largest magnitude frequencies of $\hat{f} \in \mathbb{C}^N$, and the estimation their Fourier coefficients. Randomized sublinear-time Monte Carlo algorithms, which have a small probability of failing to output accurate answers for each input signal, have been developed for solving this problem [1, 2]. These methods were implemented as the Ann Arbor Fast Fourier Transform (AAFFT) and empirically evaluated in [3]. In this paper we present and evaluate the first implementation, called the Gopher Fast Fourier Transform (GFFT), of the more recently developed sparse Fourier transform techniques from [4]. Our experiments indicate that different variants of GFFT generally outperform AAFFT with respect to runtime and sample usage.

Keywords: Computational Methods, Fourier Transform, Sparsity, Nonlinear Approximation
PACS: 02.70.-c, 89.20.Ff, 02.30.Nw

1. INTRODUCTION

In many applications only a few of the largest magnitude terms in the Fourier Transform (FT) of an N-bandwidth input signal, $f : [0,2\pi] \to \mathbb{C}$, are of interest. In such applications the Fast Fourier Transform (FFT) [5], which computes all FT terms in $O(N \log N)$-time, can be computationally wasteful when N is large. This is especially true in application areas such as MR imaging [6, 7] and analog-to-digital conversion [8, 9] where even acquiring enough signal samples on which to run a full FFT can sometimes be daunting.

Let f be a band-limited function with bandwidth N. Denote the FT of f by $\hat{f}$. Given samples from f we are interested in locating (and then approximating) $k \ll N$ of the largest magnitude entries of $\hat{f}$ as quickly as possible. Hence, we seek Fourier algorithms with runtime and sampling complexities that scale sublinearly in N, the bandwidth of our periodic input function f. Any Fourier algorithm of this type will output a k-element list of $\hat{f}$'s largest magnitude values. Hereafter we will refer to such lists as a *sparse Fourier representations*.

Over the past several years compressed sensing methods [10, 11, 12, 13, 14] have made astonishing progress with respect to reducing the number of linear measurements required in order to approximate input signals with respect to a given sparsifying basis. However, when the signal of interest is sparse in the Fourier basis all the aforementioned compressed sensing methods either run in superlinear-time, or require as many signal samples as a standard FFT. Existing randomized sublinear-time Fourier algorithms [1, 2] are capable of both decreasing sampling costs and recovering Fourier sparse signals faster than standard FFT methods [3]. This makes them promising for use in numerical applications where runtime is the dominant concern (e.g., see [15]). However, these algorithms can produce inaccurate results with some small probability on each input signal. Thus, they are inappropriate for failure intolerant applications.

More recently, deterministic sparse Fourier algorithms have been developed [4] which provide guaranteed error bounds. In this paper we perform the first empirical evaluation of several variants of these more recent sparse Fourier algorithms against the earlier methods developed in [1]. The remainder of this paper is organized as follows: In Section 2 we set terminology and state relevant theorems. In Section 3 we introduce the Gopher Fast Fourier Transform (GFFT), the first implementation of the sparse Fourier approximation techniques from [4]. To demonstrate its capabilities we empirically evaluate GFFT against both the Ann Arbor Fast Fourier Transform (AAFFT) [3], and a highly optimized standard Fast Fourier Transform implementation, FFTW3 [16]. Finally, we conclude with a brief summary of our results in Section 4.

2. BACKGROUND

We are interested in band-limited periodic functions, $f : [0,2\pi] \rightarrow \mathbb{C}$, which are approximately Fourier sparse. Throughout the remainder of this paper we will denote the bandwidth of f by N. Hence, we are justified in considering the Fourier Transform of f, $\hat{f}$, to be a finite length vector in $\mathbb{C}^N$. Because f is approximately Fourier sparse we assume only $k < N$ entries of $\hat{f} \in \mathbb{C}^N$ contain values that are significant (or large) in magnitude. Given such a signal, fast Fourier approximation algorithms [1, 2, 4] produce output consisting of frequency and Fourier coefficient pairs of the form $(\omega_1, C_{\omega_1}), \ldots, (\omega_k, C_{\omega_k})$, where each $(\omega_m, C_{\omega_m}) \in (\mathbb{Z} \cap (-N/2, N/2]) \times \mathbb{C}$. We will refer to any such set of $k < N$ tuples, $\mathbf{R}^{\mathrm{s}} = \{(\omega_m, C_m) \text{ s.t. } m \in [1,k] \cap \mathbb{N}\}$, as a *k-sparse representation*. Note that if we are given a sparse representation, $\mathbf{R}^{\mathrm{s}}$, we may consider it to be a length-N signal $\mathbf{R} \in \mathbb{C}^N$ by

$$\mathbf{R}_\omega = \begin{cases} C_\omega & \text{if } (\omega, C_\omega) \in \mathbf{R}^{\mathrm{s}} \\ 0 & \text{otherwise} \end{cases}$$

for all $\omega \in \left(-\frac{N}{2}, \frac{N}{2}\right] \cap \mathbb{Z}$.

Denote the discrete l^q-norm of a vector $\vec{a} \in \mathbb{C}^N$ by $\|\vec{a}\|_q = \left(\sum_{j=1-N/2}^{N/2} |a_j|^q\right)^{\frac{1}{q}}$, $q \in \mathbb{N}$. Furthermore, let $\vec{\omega} \in \mathbb{Z}^N$ be the first vector in lexicographical order whose entries sort $\hat{f} \in \mathbb{C}^N$ by magnitude so that $|\hat{f}_{\omega_1}| \geq |\hat{f}_{\omega_2}| \geq \ldots \geq |\hat{f}_{\omega_N}|$. We define the optimal k-sparse representation, $\mathbf{R}^{\mathrm{s}}_{\mathrm{opt}(k)}$, to be $(\omega_1, \hat{f}_{\omega_1}), \ldots, (\omega_k, \hat{f}_{\omega_k})$. All of the aforementioned sparse Fourier approximation algorithms attempt to recover k of the largest magnitude Fourier coefficients of $\hat{f}$ using as few samples from f as possible. Hence, the quality of their output sparse representation's measured with respect to the best possible k-sparse approximation errors, $\|\hat{f} - \mathbf{R}_{\mathrm{opt}(k)}\|_2$ and $\|\hat{f} - \mathbf{R}_{\mathrm{opt}(k)}\|_1$.

A variant of the following sparse Fourier approximation theorem is proven in [2].

Theorem 1 ***(Gilbert, Muthukrishnan, Strauss).*** *Fix precision parameters $\eta, \tau \in \mathbb{R}^+$ and probability parameter $\lambda \in (0,1)$. There exists a randomized sampling algorithm which, when given sampling access to a band-limited input signal $f : [0,2\pi] \rightarrow \mathbb{C}$ with* *bandwidth N, will output a k-sparse representation* $\mathbf{R}^s$ *for* $\hat{f}$ *satisfying*

$$\|\hat{f}-\mathbf{R}\|_2 \leq \sqrt{1+\tau}\cdot\max\left\{\eta, \|\hat{f}-\mathbf{R}_{\mathrm{opt}(k)}\|_2\right\}$$

with probability at least $1-\lambda$. *Both the runtime and sampling complexities are*

$$k\cdot\text{polynomial}\left(\log\frac{1}{\lambda}, \log\frac{1}{\eta}, \log\|\hat{f}\|_2, \log N, \frac{1}{\tau}\right).$$

Although the randomized Fourier sampling algorithm referred to in Theorem 1 will fail to output accurate results with some probability for each signal, both the probability of failure and the approximation precision can be made arbitrarily small in theory. Of course, there is no free lunch: increasing either the precision or the probability of successful approximation will increase both the runtime and sampling requirements of the algorithm by logarithmic factors. The algorithm referred to by Theorem 1 has been implemented. The implementation, called AAFFT, was empirically evaluated in [3].

The following approximation result was recently proven (see [17]) for a sparse Fourier approximation algorithm proposed in [4].

Theorem 2 *Fix precision parameter* $\varepsilon \in (0,1)$. *There exists a deterministic sampling algorithm which, when given sampling access to a band-limited input signal* $f : [0,2\pi] \to \mathbb{C}$ *with bandwidth N, will output a* $2k$*-sparse representation* $\mathbf{R}^s$ *for* $\hat{f}$ *satisfying*

$$\left\|\hat{f}-\mathbf{R}\right\|_2 \leq \left\|\hat{f}-\mathbf{R}_{\mathrm{opt}(k)}\right\|_2 + \frac{\varepsilon\cdot\left\|\hat{f}-\mathbf{R}_{\mathrm{opt}(k/\varepsilon)}\right\|_1}{\sqrt{k}}. \tag{1}$$

Both the runtime and sampling complexities are

$$O\left(\frac{k^2\cdot\log^4 N}{\varepsilon^2}\right).$$

Unlike Theorem 1, Theorem 2 provides a deterministic approximation guarantee. However, the sampling and runtime complexities of Theorem 2 scale quadratically in the sparsity parameter, k, instead of linearly as achieved by Theorem 1. Randomized versions of Theorem 2, which achieve the approximation error bound stated in Equation (1) with arbitrarily high probability for each input signal along the lines of Theorem 1, also exist. See [4, 17] for details. We have implemented variants of both the aforementioned randomized and deterministic algorithms from [4, 17]. For the remainder of this paper they will be collectively referred to as GFFT. See Table 1 for a summary of all the algorithms and implementations considered in this paper.

The first column of Table 1 lists the Fourier approximation results considered in this paper. The second column denotes whether each related Fourier result exhibits Deterministic (D) approximation guarantees, or Randomized (R) approximate recovery with high probability. The third and fourth columns of Table 1 list the runtime and sampling complexities of the Fourier results, respectively. The listed runtime and sampling complexities assume that all precision parameters (e.g., τ, η, ε in Theorems 1 and 2) are

TABLE 1. Fourier Algorithms and Implementations

Fourier Result	**D/R**	**Runtime Complexity**	**Function Samples**	**Implementation**
FFT [5]	D	$O(N \cdot \log N)$	N	FFTW3 [16]
Theorem 3 in [17]	D	$O(N \cdot k \cdot \log^2 N)$	$O(k^2 \cdot \log^2 N)$	GFFT-det-slow
Corollary 3 in [17]	R	$O(N \cdot \log N)$	$O\left(k \cdot \log^2 N\right)$	GFFT-rand-slow
Theorem 1	R	$O\left(k \cdot \log^{O(1)} N\right)$	$O\left(k \cdot \log^{O(1)} N\right)$	AAFFT [3]
Theorem 2	D	$O(k^2 \cdot \log^4 N)$	$O(k^2 \cdot \log^4 N)$	GFFT-det-fast
Corollary 4 in [17]	R	$O\left(k \cdot \log^5 N\right)$	$O\left(k \cdot \log^4 N\right)$	GFFT-rand-fast

fixed constants. For the randomized methods it is also assumed that the probability of successful approximation is at least $1 - 1/N^{O(1)}$ (e.g., that λ in Theorem 1 is $1/N^{O(1)}$). In the third row of Table 1 the stated $O(N \log N)$ runtime for Corollary 3 holds only when the number of samples it utilizes is $O(N)$ (i.e., only if k is $O(N/\log^2 N)$). Finally, the fifth column of Table 1 lists the implementation of each Fourier algorithm tested in Section 3.

3. EMPIRICAL EVALUATION

All experiments were run on a system with a AMD Phenom II X4 965 3.4 GHz processor and 8GB of DDR2-800 RAM, running Ubuntu 10.04 with Linux Kernel 2.6.32-22 for x86_64, GCC 4.4.3, and the FFTW 3.2.2 library. FFTW3 was always run using an FFTW_PATIENT plan with wisdom enabled. All GFFT variants were implemented in C++ as per Algorithms 1, 2, and 3 in [4]. All run times are reported in tick counts using the "cycle.h" file included with the source code to the FFTW 3.2.2 library. Tick counts correspond closely to the number of CPU cycles used during the execution of a program and, therefore, accurately reflect each implementation's comparative computational complexity. They are used by FFTW3 in order to measure the run times of various DFT algorithms so as to select the fastest for execution on problems of a given size (see [16] for details).

Every data point in every figure below has an associated (i) sparsity $k \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60\}$, (ii) bandwidth $N \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, 2^{17}, 2^{18}, 2^{19}, 2^{20}, 2^{21}\}$, and (iii) set of $1,000$ trial signals. Every trial signal, $f : [0, 2\pi] \to \mathbb{C}$, was constructed independently as follows: First, k frequency values, $\{\omega_1, \dots, \omega_k\}$, were independently selected uniformly at random without replacement from $(0, N] \cap \mathbb{Z}$. Next, k complex values, $\{C_1, \dots, C_k\}$, were independently selected uniformly at random from the unit circle in the complex plane (i.e., each C_j has magnitude 1 and a random phase angle in $[0, 2\pi)$). Finally, f was defined to be

$$f(x) = \sum_{j=1}^{k} C_j \cdot \mathbb{e}^{\mathbb{i} \cdot \omega_j \cdot x}. \qquad (2)$$

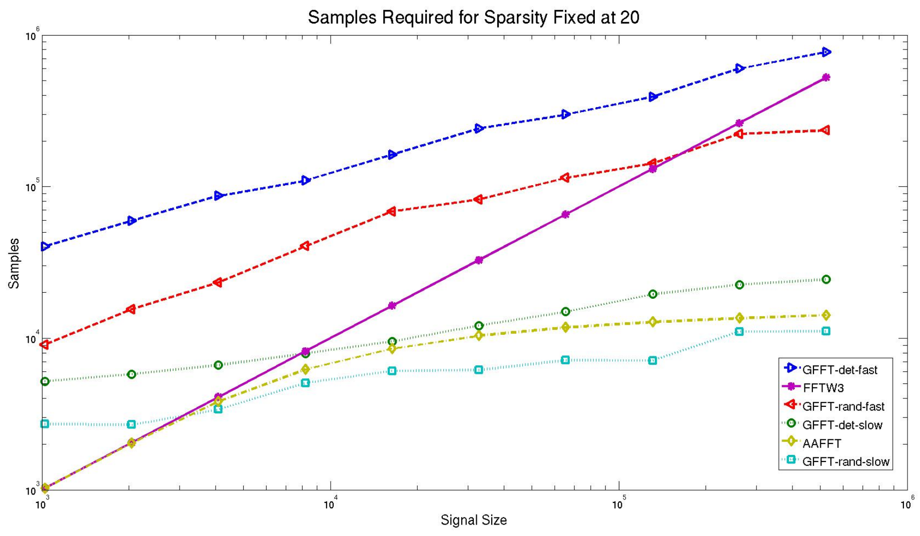


FIGURE 1. Number of Function Evaluations Required at Fixed Sparsity, $k = 20$

Every data point plotted below in Figures 3, 4, and 6 corresponds to the average runtime required by each Table 1 implementation to recover one of the data point's $1,000$ associated trial signals. Likewise, every data point plotted below in a Figures 1, 2, and 5 corresponds to the average number of function evaluations required by each Table 1 implementation to recover one of the data point's trial signals.

Note that when k is known in advance we can expect all four GFFT variants, when they succeed, to exactly recover each trial signal in Equation (2) (i.e., see Equation (1) in Theorem 2). On the other hand, AAFFT can only recover each signal up to a user defined precision (i.e., η in Theorem 1). Furthermore, some of the implementations in Table 1 guarantee recovery while others have some probability of failing to accurately recover each trail signal. In order to compare all six implementations despite these algorithmic differences, we chose parameters for each Monte-Carlo implementation (i.e., GFFT-rand-slow, GFFT-rand-fast, and AAFFT) which allowed it to recover at least 95% of every plotted data point's trial signals to within 10^{-6} l^1-error below.

3.1. Sampling Complexity

Figure 1 compares the average number of unique function evaluations, or function samples, taken by each implementation in Table 1 during the process of signal recovery when sparsity, k, is fixed at 20. The bandwidth of the signal, or signal size N, varies between 2^{10} and 2^{19} by powers of two. Using powers of two for N allows for the best possible performance by AAFFT (and FFTW3) against all four GFFT variants. Looking at Figure 1 we can see that GFFT-rand-slow has the lowest sampling complexity for all signal sizes $N \geq 2^{12} = 4,096$. However, AAFFT utilizes many fewer unique function evaluations than both fast variants of GFFT on average.

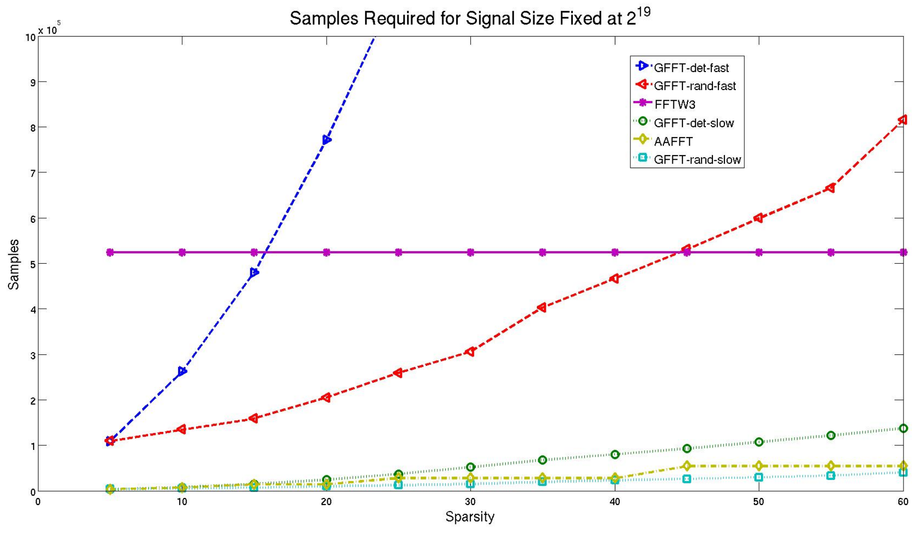


FIGURE 2. Number of Function Evaluations Required at Fixed Signal Size, $N = 2^{19}$

Figure 2 measures average sample usage with the signal bandwidths fixed to $N = 2^{19}$. The sparsity varies from $k = 5$ up to $k = 60$ by increments of 5. The results again verify that GFFT-rand-slow generally has the lowest overall average sampling complexity. AAFFT does, however, utilize significantly fewer samples than both GFFT-det-fast and GFFT-rand-fast. Most importantly, we can see that all of GFFT-det-slow, AAFFT, and GFFT-rand-slow utilize significantly fewer samples than required by a traditional FFT in order to accurately recover the sparse signals considered in this section.

3.2. Runtime Complexity

Figure 3 compares the average runtime of the implementations in Table 1 when the sparsity, k, is fixed at 20. The signal sizes vary as in Section 3.1. FFTW3 is the fastest method for all signal sizes $N \leq 2^{18}$. However, for the largest signal size both AAFFT and GFFT-rand-fast become faster. More importantly, the sublinear scaling of the fast methods with bandwidth, or signal size N, is clearly apparent.

Figure 4 compares the average runtime of all six implementations when the bandwidth, N, is fixed at 2^{19}. The sparsity varies as in Section 3.1. Looking at Figure 4 we can see that GFFT-rand-fast is generally the fastest sublinear-time method (e.g., it outperforms AAFFT at the majority of data points). At this bandwidth size GFFT-rand-fast is also faster than FFTW3 at recovering signals containing fewer than 30 nonzero Fourier coefficients. On the other hand, both slow GFFT variants require nearly 100 times as long to finish recovery as FFTW3 does despite the fact that they require significantly fewer function samples.

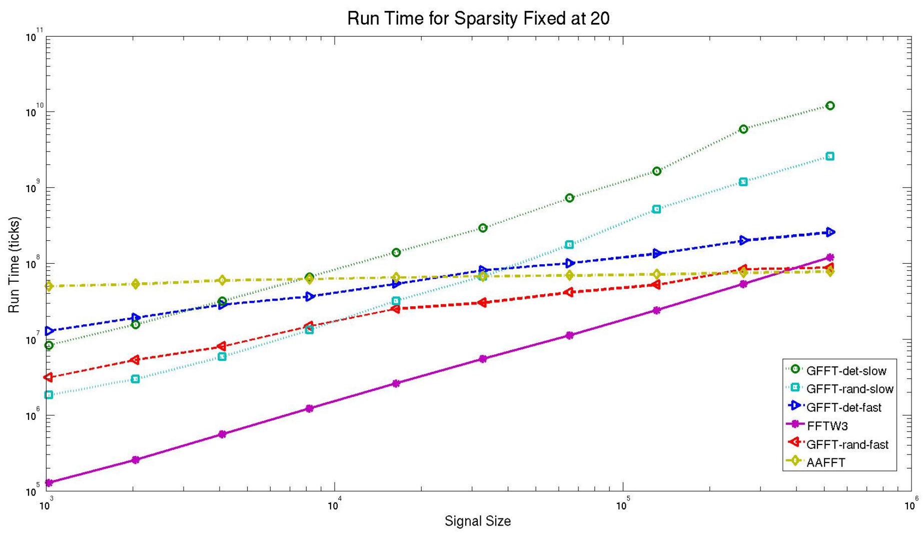


FIGURE 3. Runtime of Implementations at Fixed Sparsity, $k = 20$

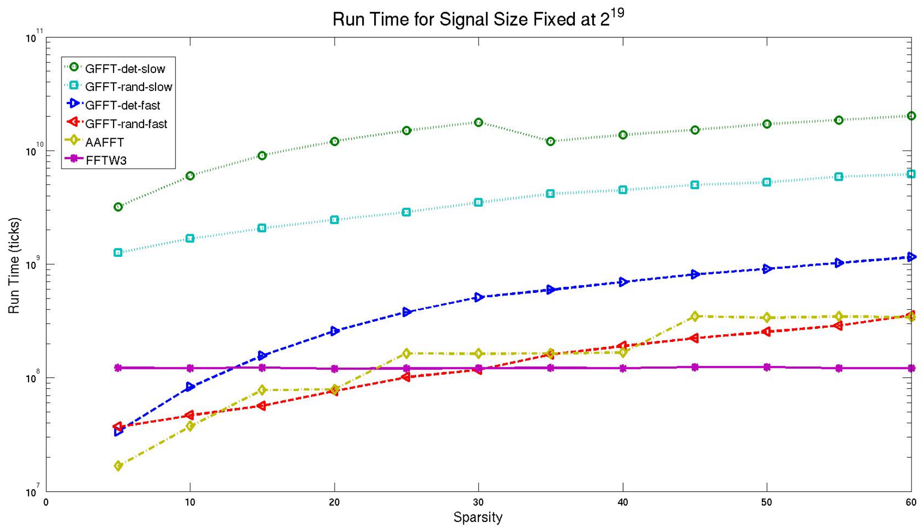


FIGURE 4. Runtime of Implementations at Fixed Signal Size, $N = 2^{19}$

3.3. Additional Experiments: Nonadaptive Sampling

In applications it is often valuable to have knowledge of the required sampling positions for a function in advance. This is particularly critical for hardware design when sampling positions must be hard coded. In Sections 3.2 and 3.1 the randomized GFFT variants had new sample positions independently chosen at random for each individual trial signal as per [4]. In this section we investigate the behavior of these variants when the sampling positions are instead randomly chosen once for each data point and then subsequently fixed for all $1,000$ trial signals.

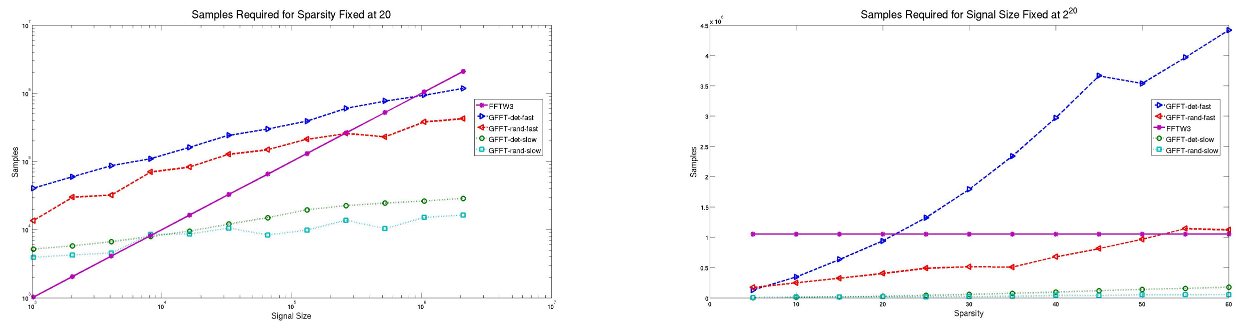


FIGURE 5. Function Evaluations Used by GFFT Variants for Fixed Sparsity and Signal Size

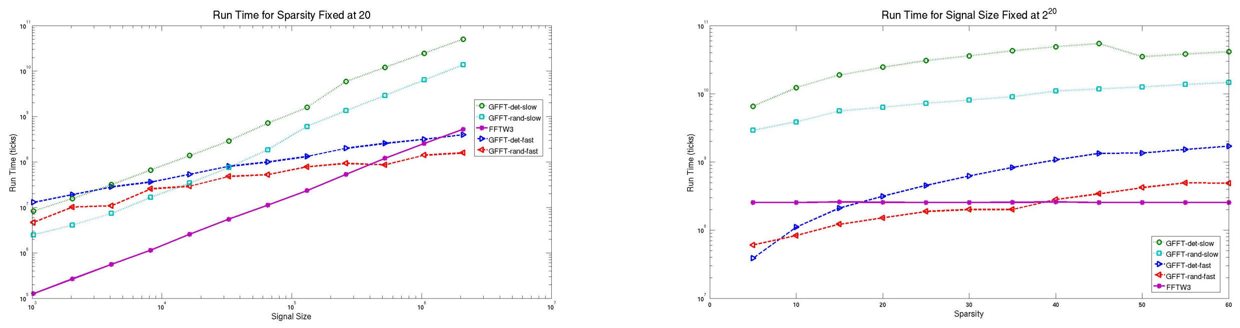


FIGURE 6. Runtime for GFFT Variants for Fixed Sparsity (left) and Fixed Signal Size (right)

Figures 5 and 6 were created as in Section 3 above, except that both GFFT-rand-slow and GFFT-rand-fast had their randomly selected point evaluation sets (i.e., the values in $[0,2\pi]$ at which each trial signal in Equation (2) was tested) fixed for all $1,000$ trial signals at each data point. As above, the l^1-error at every data point was held below 10^{-6} for at least 95% of the data point's $1,000$ trial signals. The qualitative results are essentially identical to those observed before. Nonadaptive sampling does not appear to significantly alter the behavior of the randomized GFFT variants. AAFFT was excluded from these experiments because it utilizes samples which depend on trial signal being reconstructed (i.e., it employs adaptive sampling in order to reduce sampling complexity as much as possible).

4. CONCLUSION

In this paper we implemented and empirically evaluated the sparse Fourier transform algorithms proposed in [4]. Our implementation, the Gopher Fast Fourier Transform (GFFT), was bench marked against both AAFFT, an earlier sparse Fourier transform algorithm, and FFTW3, a highly optimized standard FFT implementation. As indicated by Section 3.1, the GFFT-rand-slow variant of GFFT generally has the lowest sampling complexity of all the tested implementations. Similarly, the GFFT-rand-fast variant of GFFT generally has the lowest runtime complexity of all the tested implementations.

REFERENCES

1. A. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss, *ACM STOC* pp. 152–161 (2002).
2. A. Gilbert, S. Muthukrishnan, and M. Strauss, *Proceedings of SPIE Wavelets XI* (2005).
3. M. A. Iwen, A. C. Gilbert, and M. J. Strauss, *Communications in Mathematical Sciences* **5** (2007).
4. M. A. Iwen, *Foundations of Computational Mathematics* **10**, 303 – 338 (2010).
5. J. Cooley, and J. Tukey, *Math. Comput.* **19**, 297–301 (1965).
6. M. Lustig, D. Donoho, and J. Pauly, *Magnetic Resonance in Medicine* **58**, 1182–1195 (2007).
7. R. Maleh, A. C. Gilbert, and M. J. Strauss, *IEEE Int. Conf. on Image Processing* (2007).
8. J. Laska, S. Kirolos, Y. Massoud, R. Baraniuk, A. Gilbert, M. Iwen, and M. Strauss, *Proc. IEEE Dallas Circuits and Systems Conference* (2006).
9. S. Kirolos, J. Laska, M. Wakin, M. Duarte, D. Baron, T. Ragheb, Y. Massoud, and R. Baraniuk, *Proc. IEEE Dallas Circuits and Systems Conference* (2006).
10. D. Donoho, *IEEE Trans. on Information Theory* **52**, 1289–1306 (2006).
11. E. Candes, J. Romberg, and T. Tao, *IEEE Trans. Inform. Theory* **52**, 489–509 (2006).
12. E. Candes, and T. Tao, *IEEE Trans. on Information Theory* (2006).
13. A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, *preprint* (2006).
14. G. Cormode, and S. Muthukrishnan, *Conference on Information Sciences and Systems* (2006).
15. I. Daubechies, O. Runborg, and J. Zou, *Multiscale Model. Sim.* (2007).
16. M. Frigo, and S. Johnson, *Proceedings of IEEE 93 (2)* pp. 216–231 (2005).
17. M. A. Iwen, *Improved Approximation Guarantees for Sublinear-Time Fourier Algorithms, Preprint* (2010).