Received April 6, 2020, accepted April 17, 2020, date of publication April 21, 2020, date of current version May 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2989327

On Performance of Sparse Fast Fourier Transform Algorithms Using the Flat Window Filter

BIN LI, ZHIKANG JIANG, AND JIE CHEN
School of Mechanical and Electrical Engineering and Automation, Shanghai University, Shanghai 200072, China

Corresponding author: Jie Chen (jane.chen@shu.edu.cn)

This work was supported by Youth Program of National Natural Science Foundation of China under Grant 61703263.

ABSTRACT The problem of computing the Sparse Fast Fourier Transform(sFFT) of a K-sparse signal of size N has received significant attention for a long time. The first stage of sFFT is hashing the frequency coefficients into $B(\approx K)$ buckets named frequency bucketization. The process of frequency bucketization is achieved through the use of filters: Dirichlet kernel filter, aliasing filter, flat filter, etc. The frequency bucketization through these filters can decrease runtime and sampling complexity in low dimensions. It is a hot topic about sFFT algorithms using the flat filter because of its convenience and efficiency since its emergence and wide application. The next stage of sFFT is the spectrum reconstruction by identifying frequencies that are isolated in their buckets. Up to now, there are more than thirty different sFFT algorithms using the sFFT idea as mentioned above by their unique methods. An important question now is how to analyze and evaluate the performance of these sFFT algorithms in theory and practice. In this paper, it is mainly discussed about sFFT algorithms using the flat filter. In the first part, the paper introduces the techniques in detail, including two types of frameworks, five different methods to reconstruct spectrum and corresponding algorithms. We get the conclusion of the performance of these five algorithms, including runtime complexity, sampling complexity and robustness in theory. In the second part, we make three categories of experiments for computing the signals of different SNR, different N, and different K by a standard testing platform and record the run time, percentage of the signal sampled, and L_0, L_1, L_2 error both in the exactly sparse case and general sparse case. The result of experiments is consistent with the inferences obtained in theory. It can help us to optimize these algorithms and use them correctly in the right areas.

INDEX TERMS Sparse fast Fourier transform (sFFT), flat window filter, sub-linear algorithms, computational complexity.

I. INTRODUCTION

The Discrete Fourier Transform(DFT) is one of the most important and widely used techniques in signal processing and mathematical computing. The most popular algorithm to compute the DFT is the fast Fourier Transform(FFT) invented by Cooley and Tukey. The algorithm can compute the DFT of a signal of size N in $O(N\log N)$ time and use $O(N)$ samples. FFT dramatically simplifies the operation process; however, with the emergence of big data problems, the FFT is no longer fast enough. Furthermore, sometimes it is hard to acquire a sufficient amount of data to compute the DFT. These two problems become the major computational bottleneck in many applications. It motivates the need for new algorithms that can compute the Fourier Transform in sub-linear time and that use only a subset of the input data. People thought of many ideas to realize such an algorithm. Later, they focused on the study of the characteristics of the signal itself. The research found that a large number of signals are sparse in the frequency domain; only K frequencies are non-zeros or are significantly large. This feature is universal and inherent in signals that cover many fields(e.g., audio, video data, medical image, etc.). In this case, when $K << N$, one can retrieve the information with high accuracy using only the coefficients of the K most significant frequencies. So the sFFT has been proposed and achieved excellent results. The research of sFFT has been a hot topic in signal processing research since its birth; it was named one of the 10 Breakthrough Technologies in MIT Technology Review in 2012.

The firsts stage of the sFFT algorithm is bucketization such that the value of the bucket is the sum of the values of the frequency coefficients that hash into the bucket.

The associate editor coordinating the review of this manuscript and approving it for publication was Wenming Cao.

The number of buckets is denoted by B, and the size of one bucket is denoted by L. The process of bucketization is achieved through the use of filters. The effect of the Dirichlet kernel filter is to make the signal convoluted a rectangular window in the time domain; it can be equivalent to the signal multiply a Dirichlet kernel window of size $L(L << N)$ in the frequency domain. The typical application using the Dirichlet kernel filter is the AAFFT algorithm. The effect of the aliasing filter is to make the signal multiply a comb window in the time domain; it can be equivalent to the signal convoluted a comb window of size $B(\approx K)$ in the frequency domain. The typical application using the aliasing filter is the FFAST algorithm. The effect of the flat filter is to make the signal multiply a mix window in the time domain; it can be equivalent to the signal convoluted a flat window of size $L(L << N)$ in the frequency domain. The typical application using the flat filter is the sFFT1.0 algorithm. After bucketization, the algorithm then focuses on the non-empty buckets and computes the positions and values of the significant frequency coefficients in those buckets in what we call the spectrum reconstruction or identifying frequencies. As we can see as follows, more than thirty algorithms are using the sFFT idea, and more than ten sFFT algorithms are using the flat filter. A central question now is how to analyze and evaluate the performance of these algorithms for computing signals by the compare of themselves or other types of algorithms. It should be proved whether the runtime complexity, sampling complexity, and robustness performance are consistent with the theory or not. Are there any better ways to improve these algorithms when using it in practice? The results of these performance analyses are the guide for us to optimize these algorithms and use them correctly in different areas.

The first sFFT algorithm [1] with sub-linear runtime and sub-sampling property is a randomized algorithm with runtime and sampling complexity $O(K^2 poly(\log N))$. It was later improved to $O(K poly(\log N))$ [2], [3] through the use of binary search technique for spectrum reconstruction and the use of unequally-spaced FFTs. The algorithm is the so-called Ann Arbor fast Fourier Transform (AAFFT); the versions of them are AAFFT0.5 and AAFFT0.9.

The sFFT algorithm so-called Fast Fourier Aliasing-based Sparse Transform(FFAST) [4], [5], which focuses on exactly K-sparse signals, is an efficient algorithm. Its approach is based on the downsampling of the input signal using a constant number of co-prime downsampling factors guided by the Chinese Remainder Theorem(CRT). These aliasing patterns of different downsampled signals are formulated as parity-check constraints of useful erasure-correcting sparse-graph codes. The FFAST algorithm costs $O(K\log K)$ to compute the exact signals and only use $O(K)$ samples. The researcher adopted the FFAST framework to the case that is corrupted by white Gaussian noise. The author showed that the extended noise-robust algorithm R-FFAST [6], [7] computes the DFT using $O(K\log K)$ samples in $O(K\log^4 N)$ runtime. These two algorithms perform well when N is a product of some smaller prime numbers.

The new algorithm so-called sFFT by downsampling in the time domain(sFFT-DT) [8] is proposed in the advantage of the aliasing filter. The idea behind sFFT-DT is to downsample the original input signal first, and then all subsequent operations are conducted on the downsampled signals. To overcome the aliasing problem; the author considers the locations and values of K non-zero entries as variables and the aliasing problem is found to be equivalent to the moment-preserving problem(MPP), which can be solved via orthogonal polynomials or syndrome decoding with compressive sensing(CS) based solver.

The deterministic algorithm so-called Gopher Fast Fourier Transform(GFFT) [9], which based on the CRT, is an aliasing-based search algorithm. The approximation error bounds in [9] are further improved in [10]. Later, an algorithm so-called Christlieb Lawlor Wang Sparse Fourier Transform(CLW-SFT), which used the phase encoding method, was given in [11], [12]. The noiseless version of this algorithm is an adaptive algorithm [12], which has runtime $O(K\log K)$. The author developed this algorithm [11] by using the multiscale error-correcting method to cope with high-level noise with runtime $O(K^2\log K)$. The author evaluated the performance [13] of DMSFT (generated from GFFT) and CLW-DSFT (generated from CLW-SFT) and compared their runtime and robustness characteristics with other algorithms. These four algorithms all have a hypothesis that the algorithms can sample anywhere they want.

The sFFT algorithms using the flat window filter so-called sFFT1.0-sFFT4.0 [14], [15] can compute the exactly K-sparse signals in time $O(K\log N)$ and the general K-sparse signals in time $O(K\log N\log(N/K))$. These algorithms leverage characteristic of the flat filter. The sFFT1.0 and sFFT2.0 algorithms can identify and estimate the K largest coefficients in one shot. The sFFT3.0 algorithm can estimate the position by using only two samples of the filtered signal inspired by the frequency offset estimation in the exactly sparse case. Later, a new robust algorithm so-called Matrix Pencil FFT(MPFFT) [16] was proposed on the basis of the sFFT3.0 algorithm. The major new ingredient is a mode collision detector based on the matrix pencil method. The method enables the algorithm to use fewer samples of the input signal.

The paper [17] proposes an overview of sFFT technology and summarizes a three-step approach in the stage of spectrum reconstruction and provides a standard testing platform that can be used to evaluate different sFFT algorithms. There are also some researches try to conquer the sFFT problem from a lot of aspects: computational complexity [18], [19], performance of the algorithm [20], [21], software [22], [23], higher dimensions [24], [25], implementation [26], hardware [27] and special setting [28], [29] perspectives.

The identification of different sFFT algorithms can be known through a brief analysis as above. The Dirichlet kernel filter is not efficient because it only bins some frequency coefficients into one bucket one time. As to the aliasing filter, it is difficult to solve the worst case because there may be many frequency coefficients in the same bucket accidentally if B only can be supposed as a power of two because the scaling operation is of no use. In comparison to them, using the flat filter is very convenient and efficient.

This paper is structured as follows. Section II and Section III provide a brief overview of the sFFT technique. Section IV introduces and analyzes two frameworks and five spectrum reconstruction methods of five algorithms. In the one-shot framework, the sFFT1.0 and sFFT2.0 algorithm use the voting method with the help of the stochastic characteristics. In the iterative framework, the sFFT3.0 and sFFT4.0 algorithm use the phase encoding method with the help of the time shift characteristics, and the MPSFT algorithm uses the matrix pencil method with the help of the Prony model. In section V, we do three categories of comparison experiments. The first kind of experiment is to compare them with each other. The second is to compare them with other sFFT algorithms. The third is to compare them with optimization to them without optimization. The analysis of the experiments satisfies theoretical inference.

II. NOTATION

In this section, we initially present some notation and basic definitions of sFFT. We use $\omega_N = e^{-2\pi \mathbf{i}/N}$ as the N-th root of unify. Let $\mathbf{F}_N \in \mathbb{C}^{N\times N}$ be the DFT matrix of size N defined as follows:

$$\mathbf{F}_N[j,k] = \frac{1}{N}\omega_N^{jk} \tag{1}$$

The DFT of a vector $x \in \mathbb{C}^N$ (consider a signal of size N, where N is a power of two) is a vector $\hat{x} \in \mathbb{C}^N$ defined as follows:

$$\hat{x} = \mathbf{F}_N x \tag{2}$$

$$\hat{x}_i = \frac{1}{N}\sum_{j=0}^{N-1} x_j \omega_N^{ij} \tag{3}$$

It is necessary to consider the inverse of the DFT matrix above. $\mathbf{F}_N^{-1} \in \mathbb{C}^{N\times N}$ defined as follows:

$$\mathbf{F}_N^{-1}[j,k] = \omega_N^{-jk} \tag{4}$$

The inverse DFT of $\hat{x}$ is a vector x defined as follows:

$$x = \mathbf{F}_N^{-1}\hat{x} = \mathbf{F}_N^{-1}(\mathbf{F}_N x) \tag{5}$$

$$x_i = \sum_{j=0}^{N-1} \hat{x}_j \omega_N^{-ij} \tag{6}$$

For $x_{-i} = x_{N-i}$, we may define convolution as follows:

$$(x * y)_i = \sum_{j=0}^{N-1} x_j y_{i-j} \tag{7}$$

For coordinate-wise product $(xy)_i = x_i y_i$ and the DFT of xy is performed as described in Equation 8:

$$\widehat{xy} = \hat{x} * \hat{y} \tag{8}$$

For exact signals, $\hat{x}$ is exactly K-sparse if it has exactly K non-zero frequency coefficients while the remaining $N-K$ coefficients are zero. For general signals, $\hat{x}$ is general K-sparse if the largest K frequency coefficients $\gg$ remaining $N-K$ coefficients. The goal of the sFFT is to recover a K-sparse approximation $\hat{x}$ by finding frequency positions f and estimating values $\hat{x}_f$ of the K largest coefficients.

III. TECHNIQUES

In this section, we start with an overview of the techniques that we will use in the sFFT.

A. RANDOM SPECTRUM PERMUTATION

The random permutation includes two operations; one is shift operation, another is scaling operation. Let $\tau \in \mathbb{R}$ be the offset parameter. Let matrix $\mathbf{S}_\tau \in \mathbb{R}^{N\times N}$ representing the shift operation, is defined as follows:

$$\mathbf{S}_\tau[j,k] = \begin{cases} 1, & j-\tau \equiv k(\mathrm{mod}N) \\ 0, & \text{o.w.} \end{cases} \tag{9}$$

Let $\sigma \in \mathbb{R}$ be the scaling parameter. Let matrix $\mathbf{P}_\sigma \in \mathbb{R}^{N\times N}$ representing the scaling operation, is defined as follows:

$$\mathbf{P}_\sigma[j,k] = \begin{cases} 1, & \sigma j \equiv k(\mathrm{mod}N) \\ 0, & \text{o.w.} \end{cases} \tag{10}$$

Suppose $\sigma^{-1} \in \mathbb{R}$ exists mod N, σ^{-1} satisfies $\sigma^{-1}\sigma \equiv 1(\mathrm{mod}N)$. If a vector $x' \in \mathbb{C}^N$, $x' = \mathbf{S}_\tau \mathbf{P}_\sigma x$, such that:

$$\begin{aligned} x'_i &= x_{\sigma(i-\tau)} \\ x'_{\sigma^{-1}i+\tau} &= x_i \end{aligned} \tag{11}$$

The random permutation isolates spectral components from each other, and it is performed as follows: if $x' = \mathbf{S}_\tau \mathbf{P}_\sigma x$, such that:

$$\begin{aligned} \hat{x}'_{\sigma i} &= \hat{x}_i \omega^{\sigma\tau i} \\ \hat{x}'_i &= \hat{x}_{\sigma^{-1}i}\omega^{\tau i} \end{aligned} \tag{12}$$

B. WINDOW FUNCTION

The window function is a mathematical tool and can be seen as a matrix multiply the original signal. We introduce three filters used in the sFFT algorithm mentioned in this paper.

The first filter is the frequency aliasing filter. Through the filter, the signal in the time domain is subsampled such that the corresponding signal in the frequency domain is aliased. Let $L \in \mathbb{Z}^+$ be the subsampling factor. Let $B \in \mathbb{Z}^+$ be the subsampling number. Let matrix $\mathbf{D}_L \in \mathbb{R}^{B\times N}$ representing the subsampling operation, is defined as follows:

$$\mathbf{D}_L[j,k] = \begin{cases} 1, & k = jL \\ 0, & \text{o.w.} \end{cases} \tag{13}$$

Let vector $y_{L,\tau}, \hat{y}_{L,\tau} \in \mathbb{C}^B$ be the filtered signal obtained by shift operation and aliasing filter. If $y_{L,\tau} = \mathbf{D}_L \mathbf{S}_\tau x$,

$\hat{y}_{L,\tau} = \mathbf{F}_B\mathbf{D}_L\mathbf{S}_\tau x$, we get formula (14). If $\tau = 0$ we get formula (15).

$$\hat{y}_{L,\tau}[i] = \hat{x}[i]\omega^{-\tau i} + \hat{x}[i+B]\omega^{-\tau(i+B)} + \cdots \hat{x}[i+(L-1)B]\omega^{-\tau(i+(L-1)B)} \quad (14)$$

$$\hat{y}_{L,0}[i] = \hat{x}[i] + \hat{x}[i+B] + \cdots + \hat{x}[i+(L-1)B] \quad (15)$$

The second filter is the frequency flat filter. We use a filter vector G that is concentrated both in time and frequency domain, G is zero except at a small number of time coordinates with $\text{supp}(G) \subseteq [-w/2, w/2]$ and its Fourier Transform $\hat{G}$ is negligible except at a small fraction L ($\approx \varepsilon N$) of the frequency coordinates (the pass region). The paper [14] claim there exists a standard window function $G(\varepsilon, \varepsilon', \delta, w)$ satisfies the formula (16). The filter can be obtained by convoluted a Gaussian function with a boxcar window function and $\text{supp}(G) = w = O(1/\varepsilon \log(1/\delta)))$. One can potentially use a Dolph-Chebyshev window function with minimal big-Oh constant. In this paper, we use filter $G \in \mathbb{C}^N$ be an $(L/N, L/2N, \delta, w)$ flat window. The width of the filter in the time domain is denoted by w, the width of the passband region in the frequency domain is denoted by L, the number of buckets is denoted by B and $B = N/L$.

$$\left|\hat{G}_i\right| \in [1-\delta, 1+\delta]\, for\ i \in [-\epsilon' N, \epsilon' N]$$
$$\left|\hat{G}_i\right| \in [0, \delta]\, for\ i \notin [-\epsilon N, \epsilon N]$$
$$\left|\hat{G}_i\right| \in [0, 1]\, for\ |i| \in [\epsilon' N, \epsilon N] \quad (16)$$

Let matrix $\mathbf{Q}_L \in \mathbb{C}^{N\times N}$ be a diagonal matrix whose diagonal entries represent filter coefficients in the time domain, is defined as follows:

$$\mathbf{Q}_L[j,k] = \begin{cases} G_j, & j = k \\ 0, & \text{o.w.} \end{cases} \quad (17)$$

The third filter is the frequency subsampled filter. Through the filter, the signal in the time domain is aliased such that the corresponding signal in the frequency domain is subsampled. Let matrix $\mathbf{U}_L \in \mathbb{R}^{B\times N}$ represents the aliasing operator as follows:

$$\mathbf{U}_L[j,k] = \begin{cases} 1, & j-k \equiv 0(\text{mod}B) \\ 0, & \text{o.w.} \end{cases} \quad (18)$$

Let vector $y_L,\ \hat{y}_L \in \mathbb{C}^B$, be the filtered signal obtained by the subsampled filter. If $y_L = \mathbf{U}_L x,\ \hat{y}_L = \mathbf{F}_B\mathbf{U}_L x$, we get formula(19).

$$\hat{y}_L[i] = \hat{x}[iL] \quad (19)$$

C. FREQUENCY BUCKETIZATION

The process of bucketization in this paper is achieved through the use of the flat filter, the subsampled filter, shift operation and scaling operation. It can be equivalent to the signal multiply $\mathbf{F}_B\mathbf{U}_L\mathbf{Q}_L\mathbf{S}_\tau\mathbf{P}_\sigma$. The filtered signal is performed as follows:

Lemma 1: If $y_{L,\tau,\sigma} = \mathbf{U}_L\mathbf{Q}_L\mathbf{S}_\tau\mathbf{P}_\sigma x$, and $\hat{y}_{L,\tau,\sigma} = \mathbf{F}_B\mathbf{U}_L\mathbf{Q}_L\mathbf{S}_\tau\mathbf{P}_\sigma x$, such that:

$$\hat{y}_{L,\tau,\sigma}[0] \approx \hat{G}_{\frac{L}{2}}\hat{x}_{\sigma^{-1}\left(-\frac{L}{2}\right)}\omega_N^{\tau\left(-\frac{L}{2}\right)} + \ldots \hat{G}_{-\frac{L}{2}+1}\hat{x}_{\sigma^{-1}(\frac{L}{2}-1)}\omega_N^{\tau(\frac{L}{2}-1)}$$
$$\hat{y}_{L,\tau,\sigma}[1] \approx \hat{G}_{\frac{L}{2}}\hat{x}_{\sigma^{-1}\left(\frac{L}{2}\right)}\omega_N^{\tau\left(\frac{L}{2}\right)} + \ldots \hat{G}_{-\frac{L}{2}+1}\hat{x}_{\sigma^{-1}(\frac{3L}{2}-1)}\omega_N^{\tau(\frac{3L}{2}-1)}$$
$$\hat{y}_{L,\tau,\sigma}[i] \approx \hat{G}_{\frac{L}{2}}\hat{x}_{\sigma^{-1}\left(\frac{(2i-1)L)}{2}\right)}\omega_N^{\tau\left(\frac{(2i-1)L}{2}\right)} + \ldots \hat{G}_{-\frac{L}{2}+1}\hat{x}_{\sigma^{-1}(\frac{(2i+1)L}{2}-1)}\omega_N^{\tau(\frac{(2i+1)L}{2}-1)} \quad (20)$$

Proof:

$$(\text{P0})x' = \mathbf{S}_\tau\mathbf{P}_\sigma x \Rightarrow \hat{x'}_i = \hat{x}_{\sigma^{-1}i}\omega^{\tau i}$$

$$(\text{P1})x'' = \mathbf{Q}_L x' \Rightarrow \begin{bmatrix} \hat{x''}[0] \\ \cdots \\ \hat{x''}[L] \\ \cdots \\ \hat{x''}[iL] \\ \cdots \end{bmatrix} = \begin{bmatrix} \hat{x'}[0] \\ \cdots \\ \hat{x'}[L] \\ \cdots \\ \hat{x'}[iL] \\ \cdots \end{bmatrix} * \begin{bmatrix} \hat{G}[0] \\ \cdots \\ \hat{G}[L] \\ \cdots \\ \hat{G}[iL] \\ \cdots \end{bmatrix}$$

$$(\text{P2})\,|G[i]| = \begin{cases} \approx 1, & i \in [-\frac{L}{2}+1, \frac{L}{2}] \\ \approx 0, & \text{o.w.} \end{cases}$$

$$(\text{P3})\hat{y}_{L,\tau,\sigma} = \mathbf{F}_B\mathbf{U}_L x'' \Rightarrow \hat{y}_{L,\tau,\sigma}[i] = \hat{x''}[iL]$$

$$(\text{P4})\hat{y}_{L,\tau,\sigma} = \mathbf{F}_B\mathbf{U}_L\mathbf{Q}_L\mathbf{S}_\tau\mathbf{P}_\sigma x$$

Based on the above-mentioned properties we get formula(20)

If the set I is a set of coordinates position, the position $f = (\sigma^{-1}u)\text{mod}N \in I$, suppose there is no hash collision in the bucket i, $i = \text{round}(u/L)$, round() means to make decimals rounded. Through formula(20), we can get the formula(21)

$$\hat{y}_{L,\tau,\sigma}[i] \approx \hat{G}_{iL-u}\hat{x}_{\sigma^{-1}u}\omega_N^{\tau u} \text{ for } u \in [\frac{(2i-1)L}{2}, \frac{(2i+1)L}{2} - 1]$$
$$\hat{x}_f \approx \hat{y}_{L,\tau,\sigma}[i]\omega_N^{-\tau u}/\hat{G}_{iL-u} \text{ for } u = \sigma f \text{mod}N,\ i = \text{round}(u/L) \quad (21)$$

As we see above, frequency bucketization includes three steps: random spectrum permutation($x' = \mathbf{S}_\tau\mathbf{P}_\sigma x$, it cost 0 runtime), flat window filter($x'' = \mathbf{Q}_L x'$, it cost w runtime and w samples), Fourier Transform of the aliasing signal($\hat{y}_{L,\tau,\sigma} = \mathbf{F}_B\mathbf{U}_L x''$, it cost $B\log B$ runtime and 0 samples). So totally frequency bucketization one round cost $w + B\log B$ runtime and w samples.

IV. ALGORITHMS ANALYSIS

As mentioned above the goal of frequency bucketization is to decrease runtime and sampling complexity in the advantage of low dimensions; after bucketization the filtered signal $\hat{y}_{L,\tau,\sigma}$ can be obtained by original signal x. In this section, we introduce two frameworks, five methods and corresponding

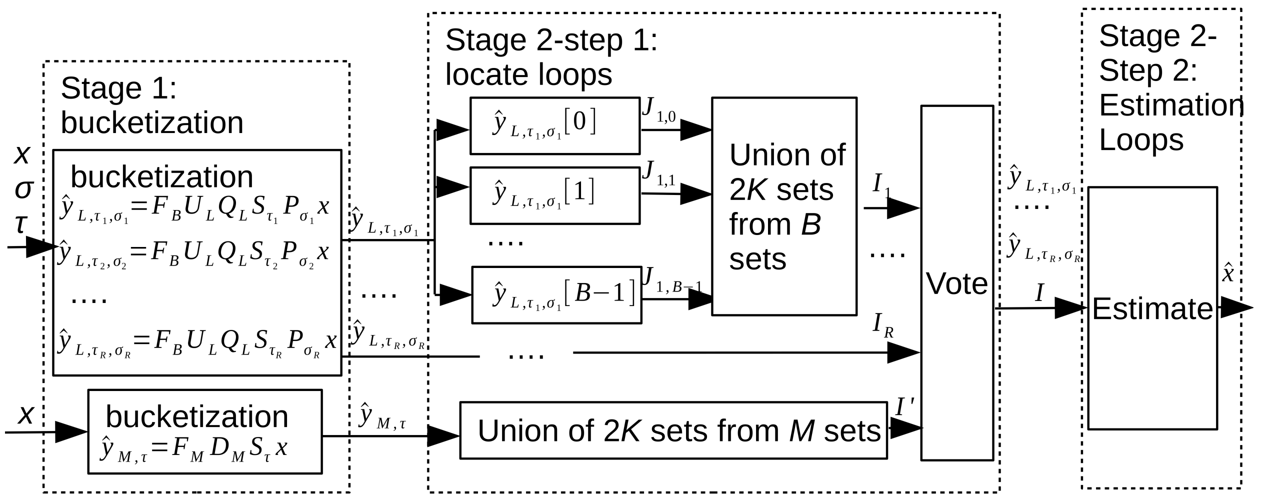

Stage 1: bucketization
bucketization
$\hat{y}_{L,\tau_1,\sigma_1}=F_B U_L Q_L S_{\tau_1} P_{\sigma_1} x$
$\hat{y}_{L,\tau_2,\sigma_2}=F_B U_L Q_L S_{\tau_2} P_{\sigma_2} x$
$\hat{y}_{L,\tau_R,\sigma_R}=F_B U_L Q_L S_{\tau_R} P_{\sigma_R} x$
x σ τ
bucketization
$\hat{y}_{M,\tau}=F_M D_M S_\tau x$
Stage 2-step 1: locate loops
$\hat{y}_{L,\tau_1,\sigma_1}[0]$
$\hat{y}_{L,\tau_1,\sigma_1}[1]$
$\hat{y}_{L,\tau_1,\sigma_1}[B-1]$
$J_{1,0}$ $J_{1,1}$ $J_{1,B-1}$
Union of 2K sets from B sets
I_1 I_R I'
Vote
Union of 2K sets from M sets
Stage 2-Step 2: Estimation Loops
Estimate
I
$\hat{x}$

FIGURE 1. A system block diagram of the one-shot framework.

algorithms to recover the spectrum $\hat{x}$ of the filtered signal $\hat{y}_{L,\tau,\sigma}$ in their own way.

A. THE sFFT1.0 ALGORITHM BY THE ONE-SHOT FRAMEWORK

The first framework can directly reconstruct the spectrum by one-shot does not need iteration. The process to reconstruct the spectrum of the sFFT1.0 algorithm includes two kinds of rounds, the first is location round and another is estimation round. Every location round one time generates a list of candidate coordinates I_r. Candidate coordinates $i \in I_r$ have a certain probability of being indices of one of the K significant coefficients in spectrum. By running multiple rounds, this probability can be increased, so it is certain to vote the candidate coordinates with a high probability after $R(\approx \log N)$ times' rounds. The next step is to do estimation rounds used to exactly determine the value of identified frequency $\hat{x}_f$ isolated in the bucket in the reason of the value of the bucket is approximate the frequency that identified in the bucket if there is no hash collision. The block diagram of the sFFT algorithms system of the one-shot framework is shown in Figure 1. We explain the details as follows.

Stage1 Bucketization: Run R times' round for set $\tau = \{\tau_1, \tau_2, \cdots \tau_R\}$ and set $\sigma = \{\sigma_1, \sigma_2, \cdots \sigma_R\}$, Calculate $\hat{y}_{L,\tau,\sigma} = \mathbf{F}_B\mathbf{U}_L\mathbf{Q}_L\mathbf{S}_\tau\mathbf{P}_\sigma x$ representing the filtered spectrum.

Stage2-Step1 Location rounds: After R times' round, return R sets of coordinates $I_1, \cdots I_R$(set I_r representing a union of $2K$ sets J from B sets J, $J \in \left\{J_{r,0}, J_{r,1}, \cdots J_{r,B-1}\right\}$ in the No.r' round, set $J_{r,i} = \left\{\sigma_r^{-1}\frac{(2i-1)L}{2}, \sigma_r^{-1}(\frac{(2i-1)L}{2}+1), \cdots \sigma_r^{-1}(\frac{(2i+1)L}{2}-1)\right\}$). Then do the vote, count the number s_i of occurrences of each found coordinate i, that is: $s_i = \| \{r|i \in I_r\} \|_0$ ($\| \|_0$ representing ℓ_0-norm). Only keep the coordinates occurred in at least fifty percentage proportion($I = \{i \in I_1 \cup \cdots \cup I_R | s_i > R/2\}$).

Stage2-Step2 Estimation rounds: After location rounds, the set I can be obtained then estimate R sets of frequency coefficients $\hat{x}^1, \cdots \hat{x}^R$. The method is if position $f \in I$, we can get the value of position f through formula(21). For identified position f, R different $\hat{x}_f^r$ can be obtained in R times' round, finally use the median value of the sets as the final estimator.

Finally, we analyse the performance of the sFFT1.0 algorithm. In stage1 it cost $R(w + B\log B)$ runtime, in stage2-step1 it cost $2RK(N/B)$ runtime, in stage2-step 2 it cost $2RK$ runtime, totally it cost $O(R(w + B\log B + KN/B))$ runtime. And the runtime satisfies Lemma 2.

Lemma 2: Suppose $R = O(\log N)$, $w = B\log(N/\delta)$, $\delta = 1/(N^c)$, it cost $O(\log N\sqrt{NK\log N})$ runtime in the sFFT1.0 algorithm

Proof:

$$\begin{aligned} &O(R(w + B\log B + KN/B)) \\ &\quad = O\left(\log N \log(N/\delta)B + \log NKNB^{-1}\right) \\ &\quad \geq O(\log N\sqrt{NK\log N})(\text{ for } B = \sqrt{NK\log^{-1}(N/\delta)}) \end{aligned}$$

In stage 1 it needs w samples one time. In the first round, the signal not chosen is in the probability of $(N-w)/N$; suppose the probability does not change; on average the samples chosen after R times' round is in the number of $N\left(1-(\frac{N-w}{N})^R\right) = N\left(1-\left(\frac{N-\sqrt{KN}\log(N/\delta)}{N}\right)^{(\log N)}\right)$

B. THE sFFT2.0 ALGORITHM BY THE ONE-SHOT FRAMEWORK

As is shown in Figure 1, the sFFT2.0 algorithm is very similar to the sFFT1.0 algorithm. Additionally, another bucketization and location round are used with the frequency aliasing filter to restrict the locations of the large coefficient. Let M be the size of the aliasing filter and M divides N, it does a pre-processing stage as follows. Firstly obtain $\hat{y}_{M,0} = \mathbf{F}_M\mathbf{D}_M\mathbf{S}_0 x$, then get a union of $2K$ sets from M sets: $\{0, M, \cdots (L-1)M\}, \cdots \{M-1, 2M-1, \cdots N-1\}$ by selecting the $2K$ largest coefficients of magnitude $\hat{y}_{M,0}[i]$ for $i \in [0, M-1]$, the union of $2K$ sets is set I', assuming that all large coefficients j have j mod M in I'. That is, we restrict out sets I_r talked above to contain only coordinates i with i mod $M \in I'$, we expect that $|I_r| \approx 2K/M(2KN/B)$ rather than the previous $|I_r| \approx 2KN/B$.

Finally, we analyse the performance of the sFFT2.0 algorithm. In stage1 it is the same as the sFFT1.0 algorithm, in stage2-step1 it cost $R(2K/M \cdot 2K(N/B) + M) + M\log M$ runtime, in stage2-step 2 it is the same, totally it cost $O(R(w+BlogB + K^2N/(BM) + M) + M\log M)$ runtime. And the runtime satisfies Lemma 3.

Lemma 3: Suppose $R = O(\log N)$, $w = B\log(N/\delta)$, $\delta = 1/(N^c)$, it cost $O\left(\log N \left(K^2 N \log(N)\right)^{1/3}\right)$ runtime in the sFFT2.0 algorithm

Proof:

$$\begin{aligned}
&O(R(w + BlogB + K^2N/(BM) + M) + M\log M)\\
&= O\left(\log N \log(N/\delta)B + \log NK^2NM^{-1}B^{-1} + M\log M\right)\\
&\geq O(\log NK\sqrt{\log(N/\delta)NM^{-1}} + M\log M)\\
&\approx O(\log NK\sqrt{\log(N/\delta)NM^{-1}} + M\log N)\\
&\geq O\left(\log N \left(K^2 N \log(N)\right)^{1/3}\right)
\end{aligned}$$

Compared to the sFFT1.0 algorithm, the runtime the sFFT2.0 algorithm is a factor $(N\log N)^{1/6}$ smaller. On average the samples of the sFFT2.0 algorithm chosen is in the number of $N\left(1 - \left(\frac{N-w}{N}\right)^R\right) + M$, compared to the sFFT1.0 algorithm, B decreases so w decreases so that the samples decreases as well.

C. THE sFFT3.0 ALGORITHM BY THE ITERATIVE FRAMEWORK

Compared to the one-shot framework, the iterative framework has two improvements. The first advantage of the iterative framework is that once a frequency coefficient of the signal was found and estimated, it can be subtracted from the signal. This fact can be used to reduce the amount of work to be done in subsequent steps. It is not necessary to update the whole input signal. Instead, it is sufficient to update the B-dimensional buckets. This way, the removal of the effects of already found coefficients can be done in $O(B)$ time. The second important improvement in the iterative framework is an improved method for finding the signal's significant frequency coordinates rather than the voting method by R times' rounds. In the one-shot framework, $R(\approx \log N)$ rounds are run and their results combined in order to get correct locations at a high probability. In the iterative algorithms, two or $\log_2 L$ rounds is enough in their own ways.

In the No.m' iteration, let K_m be the expected sparsity, R_m be how many rounds in the No.m' iteration, B_m be the number of buckets, L_m be the size of one bucket, w_m be the support of filter G, $\hat{y}_{L,\tau,\sigma}$ be filtered spectrum, $\hat{y}_{update}$ be the spectrum have already gained, $\hat{x}^{m-1}$ be the last result, $\hat{y'}_{L,\tau,\sigma}$ be the spectrum need to recover, $\hat{x'}^m$ be the recovered spectrum, $\hat{x}^m$ be the new result, set $\tau = \{\tau_1, \tau_2, \cdots \tau_R\}$ and set $\sigma = \{\sigma_1, \sigma_2, \cdots \sigma_R\}$ be the parameter. It can be seen that $R_m = 2$ in the sFFT3.0 algorithm, $R_m = \log_l L_m$ in the sFFT4.0 algorithm, $Rm = \log_2 L_m$ in the MPSFT algorithm. The detailed course in No.m' iteration is shown in Figure 2 and explained as follows.

Step1: Run R_m bucketization rounds for K_m, B_m, L_m, set σ and set τ to calculate $\hat{y}_{L,\tau,\sigma} = \mathbf{F}_B\mathbf{U}_L\mathbf{Q}_L\mathbf{S}_\tau\mathbf{P}_\sigma x$ representing the filtered spectrum.

Step2: Run R_m times' rounds for $\hat{x}^{m-1}$, set τ, set σ and formula(21) to obtain $\hat{y}_{update}$ representing spectrum have already gained.

Step3: $\hat{y'}_{L,\tau,\sigma} = \hat{y}_{L,\tau,\sigma}$ - $\hat{y}_{update}$ representing the spectrum need to recover.

Step4: recover the spectrum $\hat{x'}^m$ of $\hat{y'}_{L,\tau,\sigma}$ by different methods.

Step5: $\hat{x}^m = \hat{x}^{m-1} + \hat{x'}^m$ representing the result of this iteration.

Step6: If it is the last iteration, the final result is $\hat{x}^m$, otherwise $\hat{x}^m$ will be the input to make $\hat{y}_{update}$ in the next iteration.

It is sufficient to locate the position only using $R(=2)$ rounds instead of $R(\approx \log N)$ rounds by the phase encoding method in the sFFT3.0 algorithm in the exactly sparse case. The process is in the first round we set $\tau_1 = 0$, and the second round we set $\tau_2 = 1$, then suppose in the bucket i, it contains only one large frequency, so we get $\hat{y}_{L,0,\sigma}[i] \approx \hat{G}_{iL-u}\hat{X}_{\sigma^{-1}(u)}\omega_N^{0\cdot(u)}$ and $\hat{y}_{L,1,\sigma}[i] \approx \hat{G}_{iL-u}\hat{X}_{\sigma^{-1}(u)}\omega_N^{1\cdot(u)}$, then $\omega_N^{1\cdot(u)} \approx \frac{\hat{y}_{L,0,\sigma}[i]}{\hat{y}_{L,1,\sigma}[i]}$ so we can locate the position $f = (\sigma^{-1}u)\text{mod}N$. As to estimate the value identified, it is similar to the process of the sFFT1.0 algorithm. Finally, we analyze the performance of the sFFT3.0 algorithm. And it satisfies Lemma 4.

Lemma 4: In the sFFT3.0 algorithm, it cost $O(K\log N)$ runtime and $O(K\log N)$ samples.

Proof: In the first iteration, suppose $w_1 = B_1\log(N/\delta)$, $K_1 = K$, it cost $2(w_1{+}B_1\log B_1{+}K_1) = O(B_1\log N)$ runtime and find at least $K/2$ true frequency, In the second iteration, suppose $B_2 = B_1/2$, $w_2 = B_2\log(N/\delta)$, $K_2 = K/2$, it cost $2(w_2 + B_2\log B_2)$ runtime in the step1, it cost $2K_1$ runtime in the step2, it cost $2B_2$ runtime in the step3, it cost $2K_2$ runtime in the step4, it cost K_2 runtime in the step5, it total cost $O(w_2 + B_2\log B_2 + K_1) < O(B_1\log N)/2$ runtime in the second iteration, so the total runtime is $O(B_1\log N) + O(B_1\log N/2) + \cdots = O(B_1\log N) = O(K\log N)$. As to the sampling complexity, it is clear that the samples chosen after two rounds in the first iteration is in the number of $2w_1 = 2B_1\log(N/\delta) = O(K\log N)$, as to the samples of the following iterations, the samples can be ignored because of the decrease of B.

D. THE sFFT4.0 ALGORITHM BY THE ITERATIVE FRAMEWORK

Compared with the sFFT3.0 algorithm, only step4 of the sFFT4.0 algorithm is different. In the sFFT3.0 algorithm, it is sufficient to locate the position only running two rounds in the noiseless case in advance of it satisfies Lemma 5. In the

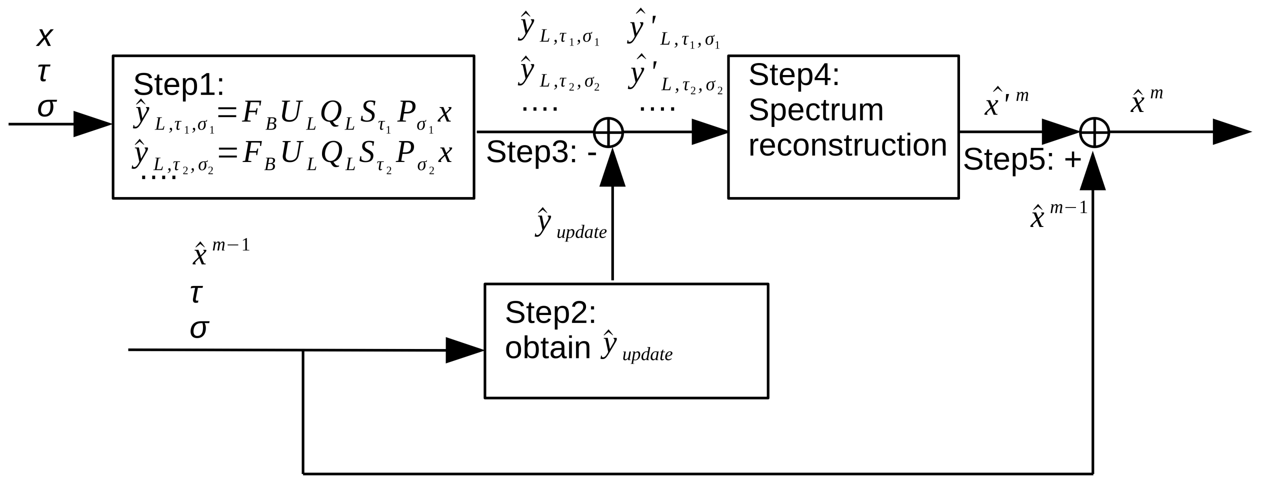

x
τ
σ
Step1:
$\hat{y}_{L,\tau_1,\sigma_1}=F_B U_L Q_L S_{\tau_1} P_{\sigma_1} x$
$\hat{y}_{L,\tau_2,\sigma_2}=F_B U_L Q_L S_{\tau_2} P_{\sigma_2} x$
$\hat{y}_{L,\tau_1,\sigma_1}$ $\hat{y}'_{L,\tau_1,\sigma_1}$
$\hat{y}_{L,\tau_2,\sigma_2}$ $\hat{y}'_{L,\tau_2,\sigma_2}$
Step3: -
Step4: Spectrum reconstruction
$\hat{x}'^m$
$\hat{x}^m$
Step5: +
$\hat{x}^{m-1}$
$\hat{y}_{update}$
$\hat{x}^{m-1}$
τ
σ
Step2: obtain $\hat{y}_{update}$

FIGURE 2. **A system block diagram of the iterative framework.**

sFFT4.0 algorithm, it is sufficient to locate the position only running $R(=\log_l L)$ times' round by the multiscale phase encoding method in advance of it satisfies Lemma 6.(let l be the multiscale parameter).

Lemma 5: In the bucket i, suppose the located position is denoted by u', the real position is denoted by u, the noise is denoted by $S_\tau[i]$, it satisfied formula(22), so $S_0[i]$ is the noise in the bucket i for $\tau = 0$, $S_1[i]$ is the noise in the bucket i for $\tau = 1$, function $\Phi(\theta)$ satisfies $e^{\Phi(\theta)\mathbf{i}} = \theta$, $\Phi(\theta) \in [0, 2\pi)$. In the sFFT3.0 algorithm, the algorithm guarantee must be required as formula(23).

$$S_\tau[i] = \hat{y}_{L,\tau,\sigma}[i] - \hat{G}_{iL-u}\hat{x}_{\sigma^{-1}u}\omega_N^{\tau u} \tag{22}$$

$$\left|\Phi\left(\frac{\hat{y}_{L,0,\sigma}[i]}{\hat{y}_{L,1,\sigma}[i]}\right) - \Phi\left(\frac{\hat{y}_{L,0,\sigma}[i]-S_0[i]}{\hat{y}_{L,1,\sigma}[i]-S_1[i]}\right)\right| \le \frac{\pi}{N} \tag{23}$$

Proof:

$$\omega_N^{1\cdot(u')} = \frac{\hat{y}_{L,0,\sigma}[i]}{\hat{y}_{L,1,\sigma}[i]} \Rightarrow u' = \text{round}(\Phi(\frac{\hat{y}_0[i]}{\hat{y}_1[i]})\frac{N}{2\pi})$$

$$\omega_N^{1\cdot(u)} = \frac{\hat{y}_{L,0,\sigma}[i]-S_0[i]}{\hat{y}_{L,1,\sigma}[i]-S_1[i]} \Rightarrow u = \Phi(\frac{\hat{y}_0[i]-S_0[i]}{\hat{y}_1[i]-S_1[i]})\frac{N}{2\pi}$$

in order to $u' = u \Rightarrow$ absolute value ≤ 0.5

$$\Rightarrow \left|\Phi\left(\frac{\hat{y}_{L,0,\sigma}[i]}{\hat{y}_{L,1,\sigma}[i]}\right) - \Phi\left(\frac{\hat{y}_{L,0,\sigma}[i]-S_0[i]}{\hat{y}_{L,1,\sigma}[i]-S_1[i]}\right)\right| \le \frac{\pi}{N}$$

Lemma 6: In the bucket i, suppose L be the range in this location, l be the multiscale parameter, r be the size of one scale($r = L/l$), u_0 be the initial position, u'_l be the located value from located position $u'(u'_l = (u' - u_0)/r,\ u'_l \in [0, l])$, u_l be the real value from real position $u(u_l = (u - u_0)/r,\ u_l \in [0, l])$, $\tau_1 = 0$, $\tau_2 \approx N/L$, In the sFFT4.0 algorithm, the algorithm guarantee must be required as formula(24),

$$\left|\Phi\left(\frac{\hat{y}_{L,\tau_1,\sigma}[i]}{\hat{y}_{L,\tau_2,\sigma}[i]}\right) - \Phi\left(\frac{\hat{y}_{L,\tau_1,\sigma}[i]-S_{\tau_1}[i]}{\hat{y}_{L,\tau_2,\sigma}[i]-S_{\tau_2}[i]}\right)\right| \le \frac{\pi}{l} \tag{24}$$

Proof:

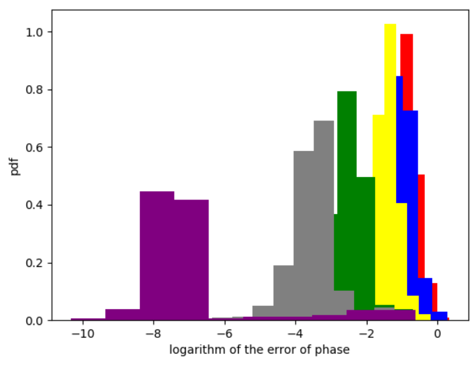
$$\omega_N^{\tau_2 u'} = \frac{\hat{y}_{L,\tau_1,\sigma}[i]}{\hat{y}_{L,\tau_2,\sigma}[i]} \Rightarrow (u_0 + u'_l r)\tau_2 \text{mod} N\frac{2\pi}{N} = \Phi(\frac{\hat{y}_{\tau_1}[i]}{\hat{y}_{\tau_2}[i]})$$

$$\omega_N^{\tau_2 u} = \frac{\hat{y}_{\tau_1}[i]-S_{\tau_1}[i]}{\hat{y}_{\tau_2}[i]-S_{\tau_2}[i]} \Rightarrow (u_0 + u_l r)\tau_2 \text{mod} N\frac{2\pi}{N} = \Phi(\frac{\hat{y}_{\tau_1}[i]-S_{\tau_1}[i]}{\hat{y}_{\tau_2}[i]-S_{\tau_2}[i]})$$

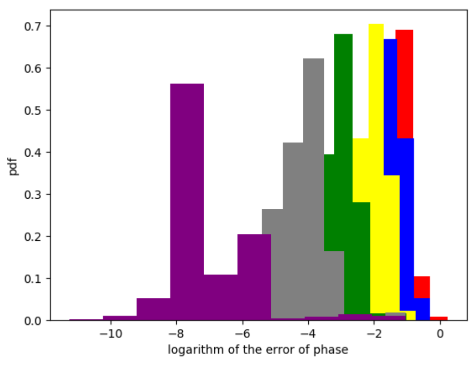
in order to $u'_l = u_l \Rightarrow$ absolute value ≤ 0.5

$$\Rightarrow \left|\Phi\left(\frac{\hat{y}_{L,\tau_1,\sigma}[i]}{\hat{y}_{L,\tau_2,\sigma}[i]}\right) - \Phi\left(\frac{\hat{y}_{L,\tau_1,\sigma}[i]-S_{\tau_1}[i]}{\hat{y}_{L,\tau_2,\sigma}[i]-S_{\tau_2}[i]}\right)\right| \le \frac{\pi}{l}$$

It is clear that the restrictive conditions of formula(23) are very harsh in the sFFT3.0 algorithm when N is large, so the sFFT3.0 algorithm is not robustness. From the lemma6, it is most robust when we use the binary search ($l = 2$), because the confidence upper limit($\pi/2$) is very big, but it is not effective. We want to know how to set the confidence upper limit of multiscale parameter l, we do it by Monte Carlo experiment, $\left|\Phi\left(\frac{\hat{y}_{L,\tau_1,\sigma}[i]}{\hat{y}_{L,\tau_2,\sigma}[i]}\right) - \Phi\left(\frac{\hat{y}_{L,\tau_1,\sigma}[i]-S_{\tau_1}[i]}{\hat{y}_{L,\tau_2,\sigma}[i]-S_{\tau_2}[i]}\right)\right|$ is denoted by $\Delta\Phi(\theta)$, we do two categories experiments to calculate the logarithm of the error of phase $\log_{10}(\Delta\Phi(\theta))$ and computing probability distribution function(PDF) of the value $\log_{10}(\Delta\Phi(\theta))$ in all valuable buckets and all rounds by the input signals of different N under different signal noise ratio(SNR) circumstances only if the bucket is not aliasing, then we get Figure 3. From Figure 3, we can see with the development of SNR(from the red space to purple space), the probability of small error increases. Compare of the two cases: small N and big N under the condition of the same K and same SNR, if N is big, it means in one bucket the noisy has less energy compared with the effective signal, it can be concluded both in theory and in experiments that the variance of the error becomes smaller when N is big. From Figure 3, if we want to keep the probability greater than 0.99 under the condition of SNR $= -20$(red space), the threshold should be more than $10^{0.5} \approx 3.2$, it seems impossible. Under the condition of SNR $= -10$(blue space), the threshold should be more than $10^0 \approx 1$; it can be solved by the binary search method because the confidence upper



(a)



(b)

FIGURE 3. PDF of the logarithm of the error of phase $\log_{10}(\Delta\Phi(\theta))$ vs SNR. (a)$K = 50$ and $N = 8192$, (b)$K = 50$ and $N = 1048576$(red space, blue space, yellow space, green space, black space, purple space individually representing SNR = −20, −10, 0, 20, 40, 120).

limit is $\pi/2 \approx 1.7$. Under the condition of SNR=0(yellow space), the threshold should be more than $10^{-0.5} \approx 0.3$, and under the condition of SNR>0, it is certain to keep the high probability if the threshold is more than 0.3. Under the condition of SNR=120(purple space), if the threshold is $\pi/N = \pi/8192 \approx 0.004 > 10^{-3}$, or the threshold is $\pi/N = \pi/1048576 \approx 0.000003 > 10^{-6}$, it can also keep the high probability to satisfy the formula(23)(Remarks: It is easy to know the PDF of the error of phase will not change much with different τ and different σ).

In the real sFFT4.0 algorithm, we use l =8; the confidence upper limit is $\pi/8 \approx 0.4$, it can keep the high probability to satisfy formula(24) under the condition of SNR$\geq$0. As to the runtime complexity, we can easy to know it should run $\log_l L(= \log_8(N/B))$ times' rounds instead of two times' rounds in every iteration, so the runtime and sampling complexity is $O(K \log N \log_8(N/K))$.

E. THE MPSFT ALGORITHM BY THE ITERATIVE FRAMEWORK

Compared with the sFFT4.0 algorithm, only the discriminant equation to the location of the MPSFT algorithm is different. The matrix pencil method, like the Prony method, is a standard technique in signal processing for mode frequency identification. In this section, we use the matrix pencil method into the MPSFT algorithm to achieve two effects. Firstly, it identifies modes much more accurately. Secondly, it helps detect errors in our mode identification step and greatly reduces the number of spurious modes being found. Rely on these; we can use only a little cost to solve the collision problem.

Suppose the number of significant frequencies in the bucket i is denoted by a. In most buckets $a = 0$, in a part of buckets $a = 1$, only in a small part of buckets $a >= 2$. Then the formula(21) can be translated to the formula(25), the problem to reconstruct spectrum is translated to how to calculate $2a$ variables as follows: a amplitudes($\hat{G}_{\text{poly}(f_0)}\hat{x}_{f_0} \cdots \hat{G}_{\text{poly}(f_{a-1})}\hat{x}_{f_{a-1}}$) and a positions($\omega^{f_0\sigma}, \cdots \omega^{f_{a-1}\sigma}$). It needs $2a$ equations, where $\hat{y}_{L,\tau,\sigma}[i]$ is known and denoted by $m_\tau = \hat{y}_{L,\tau,\sigma}[i]$ using fixed L and σ, p_j representing unknown $\hat{G}_{iL-\sigma f_j}\hat{x}_{f_j}$, z_j representing unknown $\omega_N^{\sigma f_j}$. By taking the above into consideration, the problem can be formulated by BCH codes as formula(26) by using $\tau = 0, 1, \cdots, 2a-1$.

$$\hat{y}_{L,\tau,\sigma}[i] \approx \hat{G}_{iL-\sigma f_0}\hat{x}_{f_0}\omega_N^{\sigma\tau f_0} + \cdots + \hat{G}_{iL-\sigma f_{a-1}}\hat{x}_{f_{a-1}}\omega_N^{\sigma\tau f_{a-1}} \tag{25}$$

$$\begin{bmatrix} z_0^0 & z_1^0 & \cdots & z_{a-1}^0 \\ z_0^1 & z_1^1 & \cdots & z_{a-1}^1 \\ \cdots & \cdots & \cdots & \cdots \\ z_0^{2a-1} & z_1^{2a-1} & \cdots & z_{a-1}^{2a-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \cdots \\ p_{a-1} \end{bmatrix} = \begin{bmatrix} m_0 \\ m_1 \\ \cdots \\ m_{2a-1} \end{bmatrix} \tag{26}$$

$$\mathbf{M}_{a_m} = \begin{bmatrix} m_0 & m_1 & \cdots & m_{a_m-1} \\ m_1 & m_2 & \cdots & m_{a_m} \\ \cdots & \cdots & \cdots & \cdots \\ m_{a_m-1} & m_{a_m} & \cdots & m_{2a_m-1} \end{bmatrix}_{a_m \times a_m} \tag{27}$$

Suppose there are at most a_m significant frequencies in the bucket. By singular value decomposition(SVD) of the matrix $\mathbf{M}_{a_m}$ defined as for formula(27) in bucket i, we obtain a_m singular values for each frequency. For example, we can set a_m equal to two, so we obtain two singular values by SVD. If two singular values are both small, it means there is no significant frequency. If there is only one big singular value, it means there is one significant frequency. If both of them are big, it means there are more than one significant frequencies. The way to solve the collision problem is as above, as to distinguish the position of the frequency, the method is very similar to the sFFT4.0 algorithm using binary search and the discriminant is inspired by the matrix pencil method. The detail can see [16]. It is sufficient to locate the position and estimate the value of frequencies only using $R(= 2\log_2 L)$ times' rounds in the MPSFT algorithm, the runtime and sampling complexity of the MPSFT algorithm is approximately equal to $O(K \log N \log_2(N/K))$.

After analyzing two types of frameworks, five different methods to reconstruct spectrum and corresponding

TABLE 1. The performance of sFFT algorithms and fftw algorithm in theory.

algorithm	runtime complexity	sampling complexity	robustness
sFFT1.0	$O(K^{\frac{1}{2}} N^{\frac{1}{2}} \log^{\frac{3}{2}} N)$	$N\left(1-\left(\frac{N-w}{N}\right)^{\log N}\right)$	medium
sFFT2.0	$O(K^{\frac{2}{3}} N^{\frac{1}{3}} \log^{\frac{4}{3}} N)$	$N\left(1-\left(\frac{N-w}{N}\right)^{\log N}\right)$	medium
sFFT3.0	$O(K \log N)$	$O(K \log N)$	none
sFFT4.0	$O(K \log N \log_8(N/K))$	$O(K \log N \log_8(N/K))$	bad
MPFFT	$O(K \log N \log_2(N/K))$	$O(K \log N \log_2(N/K))$	good
sFFT-DT	$O(K \log K + N)$	$O(K \log K + N)$	medium
FFAST	$O(K \log K)$	$O(K \log K)$	none
R-FFAST	$O(K \log^4 N)$	$O(K \log K)$	medium
AAFFT	$O(K \text{poly}(\log N))$	$O(K \text{poly}(\log N))$	medium
fftw	$O(N \log N)$	N	good

algorithms, Table 1[1] can be concluded with the additional information of other sFFT algorithms and fftw algorithm.

From Table 1, we can see the sFFT3.0 algorithm has the lowest runtime and sampling complexity, but it is non-robustness. Other algorithms using the flat window are good robustness but compare them with other sFFT algorithms it is no advantage in the sampling complexity except the sFFT4.0 algorithm.

V. EXPERIMENTAL EVALUATION

In this section we evaluate the performance of five sFFT algorithms using the flat window filter: sFFT1.0, sFFT2.0, sFFT3.0, sFFT4.0 and MPSFT algorithm. All of them are implemented in C or C++ language to empirically evaluate their runtime characteristics. We firstly compare these algorithms' runtime, percentage of the signal sampled and robustness characteristics with each other. Then we compare these algorithms' characteristics with other algorithms: fftw, sFFT-DT, FFAST and AAFFT algorithm. Finally, we compare these algorithms' runtime characteristics with themselves optimized. All experiments are run on a CentOS7.6 computer with 4 Intel(R) Core(TM) i5-4570 3.20GHz CPU, a cache size of 6144 KB and 8 GB of RAM.

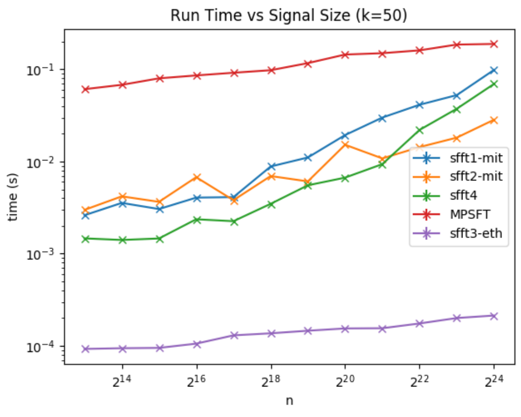
A. EXPERIMENTAL SETUP

In the experiment, the test signals are gained in a manner that K frequencies are selected from N frequencies uniformly at random and assigned a magnitude of 1 and a uniformly random phase and the rest frequencies are set to zero in the exact case. When in the general sparse case, the test signals are gained similarly but they are combined with additive white Gaussian noise, whose variance varies depending on the SNR required. Each point in the figure is the average result over 5 runs with 5 different instances as desired. The parameters of these algorithms are chosen so that can make a balance between time efficiency and robustness.

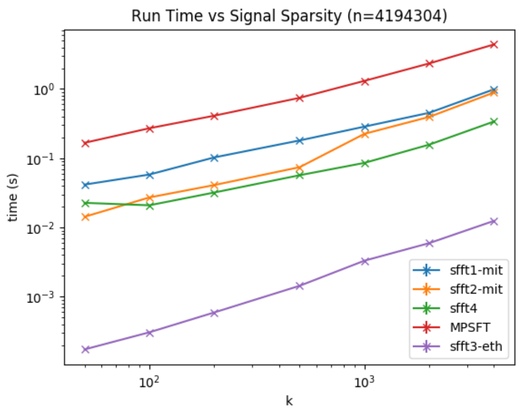
[1]The performance of algorithms using the flat window is got as above. The performance of other algorithms is got from [2], [3], [5], [6], [8]. The analysis of robustness will be explained in the next section.

B. COMPARISON EXPERIMENT ABOUT DIFFERENT ALGORITHMS USING THE FLAT FILTER OF THEMSELVES

We plot Figure 4 representing runtime vs signal size and vs signal sparsity for sFFT1.0, sFFT2.0, sFFT3.0, sFFT4.0 and MPSFT algorithm in the exactly sparse case.[2] As mentioned above, the runtime is determined by two factors. One is how many rounds(it manly depend on R) and how much time cost in one round(it manly depend on w). So from Figure 4 we can see 1)The runtime of these five algorithms are approximately linear in the log scale as a function of N and in the standard scale as a function of K. The reason is R and w is with the growth of $\log N$ and K. 2)Results of ranking the runtime complexity of five algorithms is sFFT3.0 > sFFT4.0 > sFFT2.0 > sFFT1.0 > MPSFT. The reason is their individual's R is about 2, $2\log_8 L$, approximate $\log N$, $\log N$ and $2\log_2 L$.



(a)

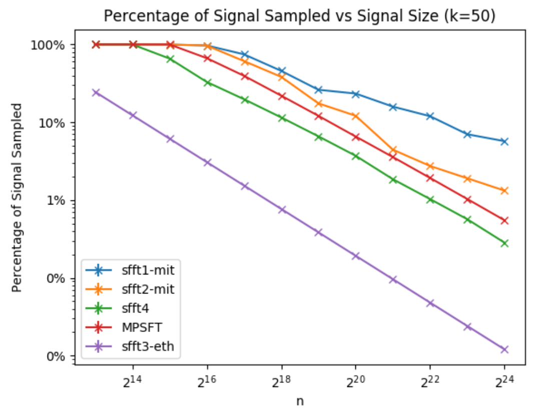


(b)

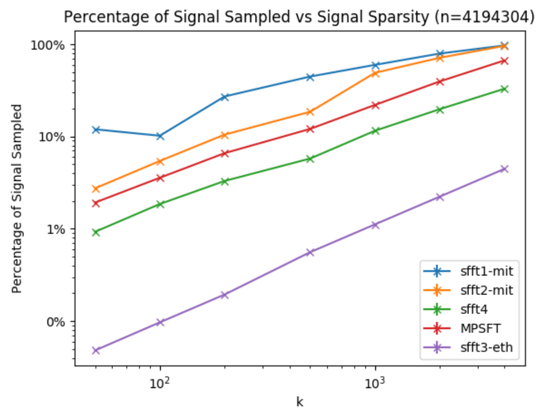
FIGURE 4. Runtime of five algorithms using the flat filter in the exactly sparse case. (a) vs signal size, (b) vs signal sparsity.

We plot Figure 5 representing the percentage of the signal sampled vs signal size and vs signal sparsity for sFFT1.0,

[2]The general sparse case means SNR=20db is very similar to the exactly sparse case except the sFFT3.0 algorithm. The detail of code, data, report can be provided in https://github.com/zkjiang/-/tree/master/docs/sfft project

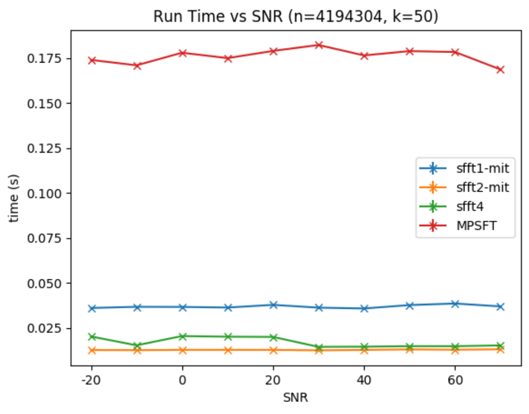


(a)

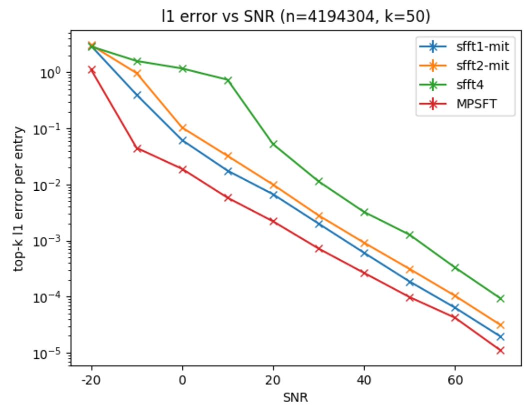


(b)

FIGURE 5. **Percentage of the signal sampled of five algorithms using the flat filter in the exactly sparse case. (a) vs signal size, (b) vs signal sparsity.**



(a)



(b)

FIGURE 6. **(a) Runtime of four algorithms using the flat filter in the general sparse case vs SNR, (b) $L1$-error of four algorithms using the flat filter in the general sparse case vs SNR.**

sFFT2.0, sFFT3.0, sFFT4.0, and MPSFT algorithm in the exactly sparse case.[3] As mentioned above, the percentage of the signal sampled is also determined by two factors: how many rounds and how many samples sampled in one round. So from Figure 5 we can see 1)The percentage of the signal sampled of these five algorithms are approximately linear in the log scale as a function of N and in the standard scale as a function of K. 2)Results of ranking the sampling complexity of five algorithms is sFFT3.0 > sFFT4.0 > MPSFT > sFFT2.0 > sFFT1.0 because of the different R.

We plot Figure 6 representing the runtime and $L1$-error vs SNR for sFFT1.0, sFFT2.0, sFFT3.0, sFFT4.0, and MPSFT algorithm.[4] From Figure 6 we can see 1)The runtime is approximately equal vs SNR. 2) To a certain extent, these four algorithms are all robustness, but when SNR is low, only MPSFT satisfies the ensure of robustness. When SNR is medium, sFFT1.0 and sFFT2.0 can also meet the ensure of robustness. And only when SNR is bigger than 20db, sFFT4.0 can deal with noise interference. The reason is that the way of binary search is better than voting method under the large noisy situation. And the way of multiscale search is not good when it use in noisy situation according to the formula(26) and Figure 3.

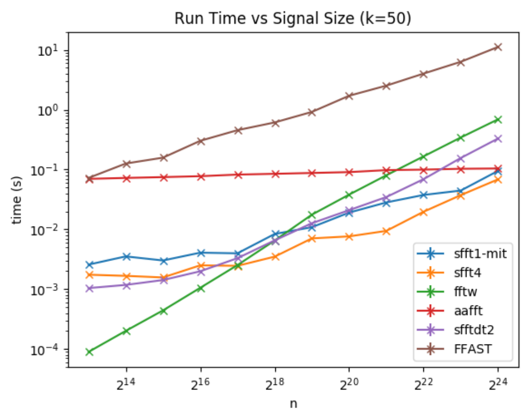
C. COMPARISON EXPERIMENT ABOUT ALGORITHMS USING THE FLAT FILTER AND OTHER ALGORITHMS

We plot Figure 7 representing run times vs signal size and vs signal sparsity for sFFT1.0, sFFT4.0, AAFFT, R-FFAST, SFFT-DT and fftw algorithm in the general sparse case.[5] From Figure 7, we can see 1)These algorithms are approximately linear in the log scale as a function of N except the fftw algorithm. These algorithms are approximately linear in the standard scale as a function of K except the fftw and SFFT-DT algorithm. 2) Results of ranking the runtime complexity of these six algorithms is sFFT4.0 > sFFT1.0 > AAFFT > SFFT-DT > fftw > R-FFAST when N is large.

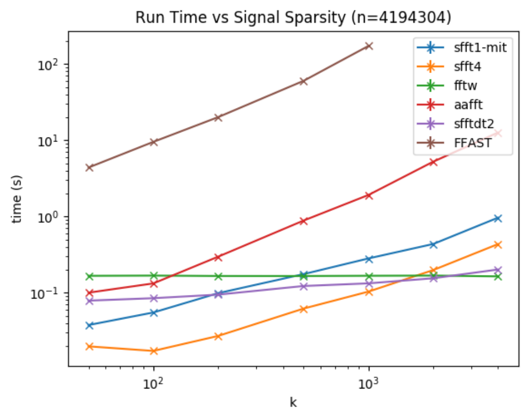
[3]The general sparse case is very similar to the exactly sparse case except the sFFT3.0 algorithm.

[4]the $L0$-error, $L1$-error $L2$-error of all experiments can be provided in https://github.com/zkjiang/-/tree/master/docs/sfft project /experiment data

[5]The exactly sparse case is very similar including the sFFT3.0 algorithm and FFAST algorithm. The FFAST and R-FFAST algorithm are not available when K is very large

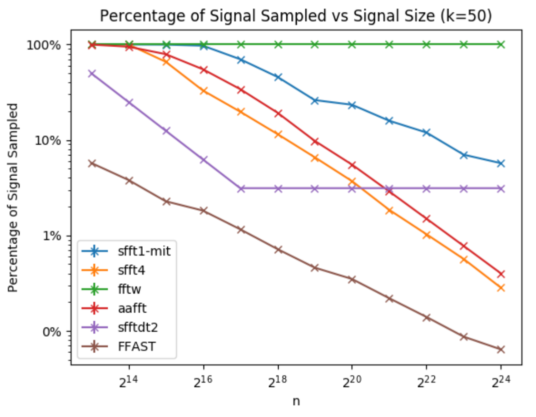


(a)

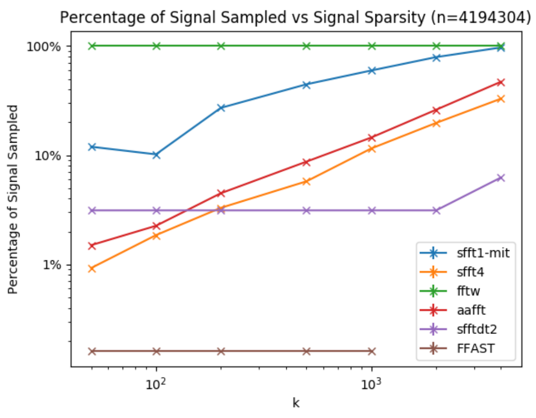


(b)

FIGURE 7. Runtime of two typical algorithms using the flat filter and other four algorithms in the general sparse case. (a) vs signal size, (b) vs signal sparsity.



(a)



(b)

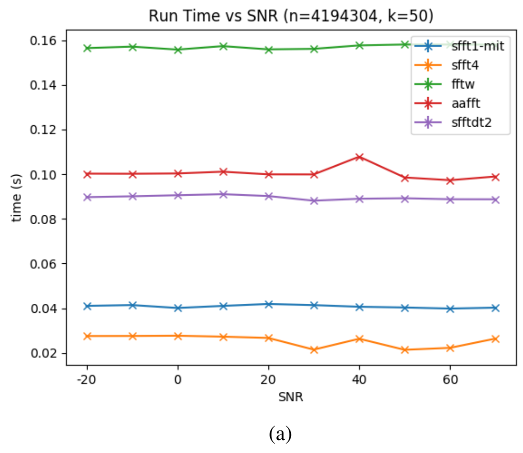
FIGURE 8. Percentage of the signal sampled of two typical algorithms using the flat filter and other four algorithms in the general sparse case. (a) vs signal size, (b) vs signal sparsity.

The reason is the Least Absolute Shrinkage and Selection Operator(LASSO) method used in the R-FFAST algorithm costs a lot of time. And the SVD and CS method used in the SFFT-DT also cost a lot of time. 2)Results of ranking the runtime complexity of these six algorithms is fftw > SFFT-DT > sFFT4.0 > sFFT1.0 > AAFFT > R-FFAST when K is large. The reason is algorithms using the aliasing filter saving the time by using a small number of buckets in the first stage compared to algorithms using the flat filter when K is large.

We plot Figure 8 representing the percentage of the signal sampled vs signal size and vs signal sparsity for sFFT1.0, sFFT4.0, AAFFT, R-FFAST, SFFT-DT and fftw algorithm in the general sparse case.[6] From Figure 8, we can see 1)These algorithms are approximately linear in the log scale as a function of N except the fftw and SFFT-DT algorithm. The reason is sampling in low-dimension in sFFT algorithms can decrease sampling complexity, and it is a limit to the size of the bucket in the SFFT-DT algorithm by using the CS and SVD method. These algorithms are approximately linear in the standard scale as a function of K except the R-FFAST and SFFT-DT algorithm. The reason is algorithms using the aliasing filter saving the time by using less number of buckets. 2)Results of ranking the sampling complexity of these six algorithms is R-FFAST > sFFT4.0 > AAFFT > SFFT-DT > sFFT4.0 > fftw when N is large. 2)Results of ranking the sampling complexity is SFFT-DT > sFFT4.0 > AAFFT > sFFT1.0 > fftw when K is large. The reason is that algorithms using the aliasing filter need less buckets than other algorithms, the number of buckets they use in the R-FFAST algorithm is only connected to the prime numbers gained by N and the number of buckets they use in the SFFT-DT algorithm is only connected to limit to the size of the bucket.

We plot Figure 9 representing runtime and $L1$-error vs SNR for sFFT1.0, sFFT4.0, AAFFT, SFFT-DT and fftw algorithm. From Figure 9 we can see 1)the runtime is approximately equal vs SNR. 2)To a certain extent, these five algorithms are

[6]The exactly sparse case is very similar including the sFFT3.0 algorithm. The FFAST and R-FFAST algorithms are not available when K is very large. It is a limit to the size of bucket in the SFFT-DT algorithm(L is not allowed more than 1024).



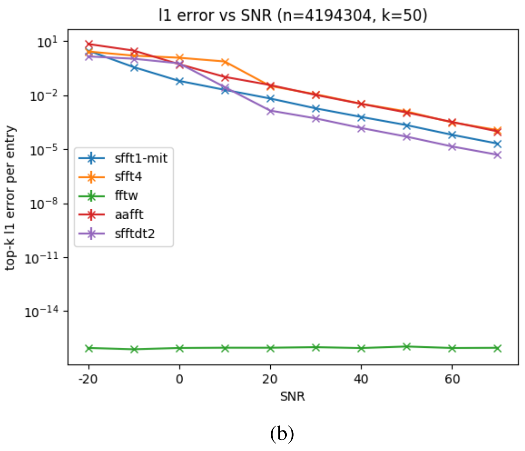
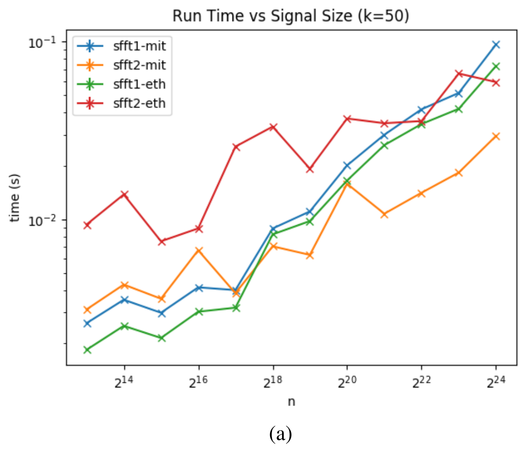


FIGURE 9. (a) Runtime of two typical algorithms using the flat filter and other four algorithms in the general sparse case vs SNR, (b) $L1$-error of two typical algorithms using the flat filter and other four algorithms in the general sparse case vs SNR.



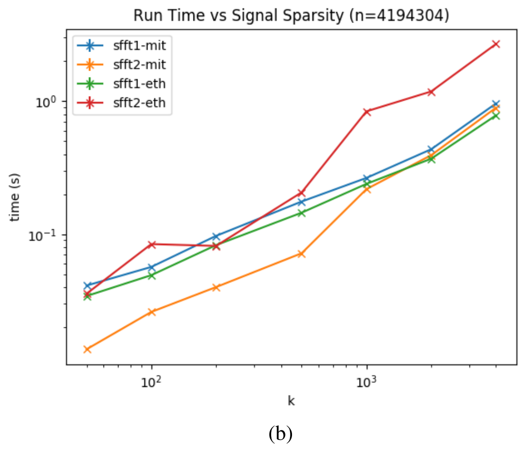


FIGURE 10. Runtime of sFFT1.0-mit, sFFT2.0-mit, sFFT1.0-eth and sFFT2.0-eth algorithm. (a) vs signal size, (b) vs signal sparsity.

all robustness, but when SNR is low, only the fftw algorithm satisfies the ensure of robustness. When SNR is medium, the sFFT1.0, AAFFT and SFFT-DT algorithm can also meet the ensure of robustness. And only when SNR is bigger than 20db, the sFFT4.0 algorithm can deal with the noise interference.

D. COMPARISON EXPERIMENT ABOUT THE SAME ALGORITHM WITH OPTIMIZATION AND WITHOUT OPTIMIZATION

We plot Figure 10[7] representing runtime vs signal size and vs signal sparsity for sFFT1.0-mit, sFFT2.0-mit, sFFT1.0-eth and sFFT2.0-eth algorithm in the general sparse case. From Figure 10, we can see the runtime of the same algorithm is accelerated a lot by the use of software optimization.

[7]sFFT1.0-mit, sFFT2.0-mit, sFFT1.0-eth and sFFT2.0-eth represent the sFFT1.0 and sFFT2.0 algorithm designed by MIT University and optimized by ETH University

VI. CONCLUSION

In the first part, the paper introduces the techniques used in sFFT algorithms including random spectrum permutation, window function and frequency bucketization. In the second part, we analyze five typical algorithms using the flat filter in detail including the sFFT1.0 algorithm using the voting method, the sFFT2.0 algorithm using the heuristic voting method by one-shot framework and the sFFT3.0 algorithm using the phase encoding method, the sFFT4.0 algorithm using the multiscale phase encoding method, the MPSFT algorithm using the matrix pencil method by the iterative framework. We get the conclusion of the performance of these five algorithms including runtime complexity, sampling complexity and robustness in theory in Table 1. In the third part, we make three categories of experiments for computing the signals of different SNR, different N, and different K by a standard testing platform through nine different sFFT algorithms and record the runtime, the percentage of the signal sampled and $L0, L1, L2$ error in every different situation both in the exactly sparse case and general sparse case. The analyse of the experiments satisfies theoretical inference.

The main contribution of this paper is 1)develop a standard testing platform which can test a lot of typical sFFT algorithms in different situations on the basis of the old platform. 2)get a conclusion of the character and performance of five typical sFFT algorithms using the flat window filter: the sFFT1.0 algorithm, the sFFT2.0 algorithm, the sFFT3.0 algorithm, the sFFT4.0 algorithm, and the MPSFT algorithm in theory and practice.

REFERENCES

[1] A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss, "Near-optimal sparse Fourier representations via sampling," in *Proc. 34th Annu. ACM Symp. Theory Comput. (STOC)*, vol. 2, 2002, pp. 152–161.

[2] A. Gilbert, M. A. Iwen, and M. Strauss, "Empirical evaluation of a sub-linear time sparse DFT algorithm," *Commun. Math. Sci.*, vol. 5, no. 4, pp. 981–998, 2007.

[3] A. C. Gilbert, M. J. Strauss, and J. A. Tropp, "A tutorial on fast Fourier sampling [How to apply it to problems]," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 57–66, Mar. 2008.

[4] 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. Inf. Theory*, Jul. 2013, pp. 464–468.

[5] S. Pawar and K. Ramchandran, "FFAST: An algorithm for computing an exactly k-sparse DFT in $O(k \log k)$ time," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 429–450, Jan. 2018.

[6] S. Pawar and K. Ramchandran, "A robust sub-linear time R-FFAST algorithm for computing a sparse DFT," 2015, *arXiv:1501.00320*. [Online]. Available: http://arxiv.org/abs/1501.00320

[7] F. Ong, R. Heckel, and K. Ramchandran, "A fast and robust paradigm for Fourier compressed sensing based on coded sampling," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 5117–5121.

[8] S.-H. Hsieh, C.-S. Lu, and S.-C. Pei, "Sparse fast Fourier transform by downsampling," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 5637–5641.

[9] M. A. Iwen, "Combinatorial sublinear-time Fourier algorithms," *Found. Comput. Math.*, vol. 10, no. 3, pp. 303–338, Jun. 2010.

[10] M. A. Iwen, "Improved approximation guarantees for sublinear-time Fourier algorithms," *Appl. Comput. Harmon. Anal.*, vol. 34, no. 1, pp. 57–82, Jan. 2013, doi: 10.1016/j.acha.2012.03.007.

[11] A. Christlieb, D. Lawlor, and Y. Wang, "A multiscale sub-linear time Fourier algorithm for noisy data," *Appl. Comput. Harmon. Anal.*, vol. 40, no. 3, pp. 553–574, May 2016, doi: 10.1016/j.acha.2015.04.002.

[12] D. Lawlor, Y. Wang, and A. Christlieb, "Adaptive sub-linear time Fourier algorithms," *Adv. Adapt. Data Anal.*, vol. 5, no. 1, Jan. 2013, Art. no. 1350003.

[13] S. Merhi, R. Zhang, M. A. Iwen, and A. Christlieb, "A new class of fully discrete sparse Fourier transforms: Faster stable implementations with guarantees," *J. Fourier Anal. Appl.*, vol. 25, no. 3, pp. 751–784, Jun. 2019.

[14] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Simple and practical algorithm for sparse Fourier transform," in *Proc. 23rd Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2012, pp. 1183–1194.

[15] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Nearly optimal sparse Fourier transform," in *Proc. 44th Symp. Theory Comput. (STOC)*, 2012, pp. 563–577.

[16] J. Chiu, "Matrix probing, skeleton decompositions, and sparse Fourier transform," Ph.D. dissertation, Dept. Mathematics, Massachusetts Inst. Technol., Cambridge, MA, USA, 2013.

[17] A. C. Gilbert, P. Indyk, M. Iwen, and L. Schmidt, "Recent developments in the sparse Fourier transform: A compressed Fourier transform for big data," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 91–100, Sep. 2014.

[18] M. Kapralov, "Sample efficient estimation and recovery in sparse FFT via isolation on average," in *Proc. IEEE 58th Annu. Symp. Found. Comput. Sci. (FOCS)*, no. 1, Oct. 2017, pp. 651–662.

[19] P. Indyk, M. Kapralov, and E. Price, "(Nearly) sample-optimal sparse Fourier transform," in *Proc. 25th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2014, pp. 480–499.

[20] A. López-Parrado and J. Velasco Medina, "Efficient software implementation of the nearly optimal sparse fast Fourier transform for the noisy case," *Ingeniería y Ciencia*, vol. 11, no. 22, pp. 73–94, Aug. 2015.

[21] G.-L. Chen, S.-H. Tsai, and K.-J. Yang, "On performance of sparse fast Fourier transform and enhancement algorithm," *IEEE Trans. Signal Process.*, vol. 65, no. 21, pp. 5716–5729, Nov. 2017.

[22] J. Schumacher and M. Puschel, "High-performance sparse fast Fourier transforms," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2014, pp. 1–6.

[23] C. Wang, S. Chandrasekaran, and B. Chapman, "CusFFT: A high-performance sparse fast Fourier transform algorithm on GPUs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2016, pp. 963–972.

[24] S. Wang, V. M. Patel, and A. Petropulu, "Multidimensional sparse Fourier transform based on the Fourier projection-slice theorem," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 54–69, Jan. 2019.

[25] M. Kapralov, A. Velingker, and A. Zandieh, "Dimension-independent sparse Fourier transform," in *Proc. Annu. ACM-SIAM Symp. Discrete Algorithms*, 2019, pp. 2709–2728.

[26] C. Pang, S. Liu, and Y. Han, "High-speed target detection algorithm based on sparse Fourier transform," *IEEE Access*, vol. 6, pp. 37828–37836, 2018.

[27] O. Abari, E. Hamed, H. Hassanieh, A. Agarwal, D. Katabi, A. P. Chandrakasan, and V. Stojanovic, "27.4 a 0.75-million-point Fourier-transform chip for frequency-sparse signals," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2014, pp. 458–459.

[28] G. Plonka and K. Wannenwetsch, "A deterministic sparse FFT algorithm for vectors with small support," *Numer. Algorithms*, vol. 71, no. 4, pp. 889–905, Apr. 2016.

[29] G. Plonka and K. Wannenwetsch, "A sparse fast Fourier algorithm for real non-negative vectors," *J. Comput. Appl. Math.*, vol. 321, pp. 532–539, Sep. 2017, doi: 10.1016/j.cam.2017.03.019.

BIN LI received the B.S. degree in computer science and the M.S. degree in automation from the Shanghai University of Science and Technology, Shanghai, China, in 1982 and 1988, respectively.

He is currently a Professor with the School of Mechanical and Electrical Engineering and Automation, Shanghai University, Shanghai. He has authored or coauthored over 17 refereed articles. He holds 19 patents. His current research interests include electromagnetic flowmeter, vortex flowmeter, ultrasonic flowmeter, sparse signal processing, and calibration instrument.

ZHIKANG JIANG received the B.S. degree in electrical engineering and automation from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2005, and the M.S. degree in automation from Guangxi University, Nanning, China in 2008. He is currently pursuing the Ph.D. degree in control theory and control engineering with Shanghai University, Shanghai, China.

His current research interests include sparse signal processing and electromagnetic flowmeter.

JIE CHEN received the B.S. degree in automatic control, the M.S. degree in detection technology and automatic equipment, and the Ph.D. degree in control theory and control engineering from Shanghai University, Shanghai, China, in 2002, 2005, and 2010, respectively.

She is currently a Lecturer with the School of Mechanical and Electrical Engineering and Automation, Shanghai University. She has authored or coauthored over 31 articles in journals and conferences. Her current research interests include electromagnetic flowmeter and sparse signal processing.