S^{3}CA: A Sparse Strip Spectral Correlation Analyzer

Carol Jingyi Li, Richard Rademacher, David Boland, Craig T. Jin, Chad M. Spooner, Philip H.W. Leong

***Abstract*—The spectral correlation density (SCD) is widely used to characterize cyclostationary signals and the strip spectral correlation analyzer (SSCA) is commonly used to estimate the SCD. Although the SSCA utilizes the fast Fourier transform (FFT) for computational efficiency, its real-time implementation still poses challenges as large input sizes are often involved. In this work, we present a sparse strip spectral correlation analyzer (S^{3}CA) based on the sparse fast Fourier transform (SFFT). The S^{3}CA approach involves computing a sparse, downsampled channel-data product (CDP) which is then passed to a modified SFFT implementation to obtain the spectral density. For an input of length 2 million samples, the S^{3}CA is 30× faster than the conventional SSCA.**

***Index Terms*—Spectral Correlation Density, Cyclostationarity, fast Fourier Transform.**

I. Introduction

If the probability distribution of a time series exhibits periodic variations, it is considered as *cyclostationary* [1], [2]. Cyclostationary time series analysis applies to a wide range of phenomena and is widely used in the analysis of digital modulation types: noise in periodic time-variant linear systems, synchronization problems, parameter and waveform estimation, channel identification and equalization, signal detection and classification, autoregressive modeling and prediction, and source separation [1], [3].

The spectral correlation density (SCD) is the idealized temporal cross correlation between all pairs of narrowband spectral-component time-series, and reflects the correlation distribution of the signal in terms of both spectral frequency and cycle frequency. Since the 1990s, computationally efficient algorithms for estimating the SCD have been studied [4], [5], [6]. Building upon the fast Fourier transform (FFT), Roberts et al. introduced the FFT accumulation method (FAM) and the strip spectral correlation analyzer (SSCA) [5]. The FAM method can suffer from degraded statistical performance due to non-uniform cycle frequency resolution and variance, leading to significant estimation errors and application limitations [5], [7]. The SSCA method was considered to be limited to smaller-size signals due to its larger memory requirements [5], [8]. References [7], [9] introduced the Fast Spectral Correlation technique using the short-time Fourier transform and [10] developed a fast average cyclic periodogram method, which overcomes memory limitations under certain conditions. The SSCA is widely used due to its computational efficiency and uniform frequency resolution.

Manuscript received xx xx, xxxx; accepted xx xx, xxxx. Data of publication xx xx, xxxx; data of current version xx xx, xxxx.

Carol Jingyi Li, Richard Rademacher, David Boland, Craig T. Jin, and Philip H.W. Leong are with The University of Sydney, Faculty of Engineering, School of Electrical and Computer Engineering, Australia (e-mail: jingyi.li@sydney.edu.au, philip.leong@sydney.edu.au).

Chad M. Spooner is with NorthWest Research Associates in Monterey, CA 93940, USA.

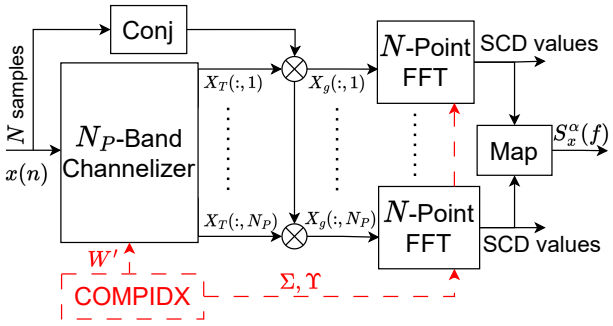

Fig. 1. The sparse strip spectral correlation analyzer (S^{3}CA) technique accelerates the strip spectral correlation analyzer (SSCA) via: (1) the COMPIDX block that evaluates a subset of the inputs and (2) replacing the N-point FFT with the SFFT.

Fig. 1 with solid lines is a signal flow diagram for the SSCA. In both the channelizer and FFT blocks, the primary computational complexity involves executing FFTs: N_P-point FFTs for the former and N-point for the latter. In practice, the value of N is commonly set within the range of 2^{16} to 2^{24}. N_P represents the number of channelizer bands and is typically chosen from 2^5 to 2^9. The cyclic spectrum is sparse in cycle frequency for all known practical digital signal types [1]. It is continuous in spectral frequency for each cycle frequency exhibited by the signal. When the cycle frequencies are unknown in advance of processing, the entire frequency/cycle-frequency plane must be computed and searched over to find the significant cycle frequencies.

The sparse fast Fourier transform (SFFT) is a recent algorithm designed for efficiently computing a FFT where the frequency domain is approximately κ-sparse, meaning κ coefficients are non-zero [11], [12]. In this paper we present the sparse strip spectral correlation analyzer (S^{3}CA), which enables fast and accurate estimation of the SCD. It is particularly useful for real-time applications involving large signal sizes, as computation and memory requirements are both reduced.

The main contributions of this paper are:

- An algorithm, based on the SFFT, that reduces the computational complexity of the SSCA from $O(NN_P(\log N_P + \log N))$ to $O(N_P \log N_P \log N \sqrt[3]{N\kappa^2 \log N})$.
- An additional optimization in which only a subset of channelizer outputs are computed and stored. This reduces space complexity of an intermediate matrix from $O(N \times N_P)$ to $O(\log N \sqrt[3]{N\kappa^2 \log N} \times N_P)$.
- A comparison of execution time and sparsity between the S^{3}CA and an SSCA using FFTW version 3.3.10 [13].

The remainder of this paper is organized as follows. In Section II, we provide the background on SSCA analysis and the sparse fast Fourier transform. Section III describes our sparse strip spectral correlation analyzer. Section IV presents our experimental results, and conclusions are drawn in Section V.

II. Background

A brief description of the SSCA algorithm and the SFFT is given here. We refer readers to references [5], [8], [14] for more detail on the SSCA and [11], [12], [15] for the SFFT.

A. Spectral Correlation Density Function

As shown in Fig. 1, the initial step involves computing the *complex demodulate*, X_T, at frequency f, from the discrete-time input values $x(n) \in \mathbb{C}$,

$$X_T(n,f) = \underbrace{[\sum_{r=-N_P/2}^{N_P/2-1} a(r)x(n+r)e^{-i2\pi f r T_s}]}_{N_P-\text{point FFT}} \underbrace{e^{-i2\pi f n T_s}}_{\text{down conversion}} \tag{1}$$

where n is a sample index, $a(r)$ is a length $T = N_P T_s$ data tapering window function, T_s is the sampling period and N_P is the number of samples [14]. The computation of the summation is performed using an N_P-point FFT, followed by the *down conversion* step.

Next, the complex demodulate X_T is multiplied by the conjugate input $x^*(n)$ [16] and windowed to produce the channel-data product (CDP) for $k \in [-N_P/2, N_P/2-1]$.

$$X_g(n+m,k) = X_T(n+m,f_k)x^*(n+m)g(m) \tag{2}$$

where the * operator is a complex conjugate, $g(m)$ is a length $\Delta t = NT_s$ windowing function, and $m \in [-N/2, N/2-1]$. The center frequencies of X_T are set to $f_k = k(f_s/N_P)$ for $f_s = 1/T_s$.

Finally, the N-point FFT of each of the N_P CDP values is computed resulting in the SCD estimate

$$S_X^{f_k+q\Delta\alpha}(\frac{f_k}{2} - q\frac{\Delta\alpha}{2})_{\Delta t} = \sum_{m=-N/2}^{N/2-1} X_g(n+m,k)e^{-i2\pi qm/N} \tag{3}$$

where cycle frequency $\alpha = f_k + q\Delta\alpha$, $\Delta\alpha = f_s/N$, $q \in [-N/2, N/2-1]$, and $f = (f_k - q\Delta\alpha)/2$ [14], [17]. In the implementation, both f and α are normalized based on $f_s = 1$, which maps the $S_X^\alpha(f)$ to a range $f \in [-0.5, 0.5]$ and $\alpha \in [-1, 1]$.

B. Sparse Fast Fourier Transform

For an input $u \in \mathbb{C}^N$, we use the notation $\hat{u} \in \mathbb{C}^N$ for its FFT. The SFFT $\hat{u}'$ is an approximation to $\hat{u}$ and assumed κ-sparse. Reference [15] proposes a number of different SFFT algorithms [18]. Although our technique could be applied to any of them, the description that follows refers to SFFT 2.0.

The SFFT 2.0 algorithm applies two randomized inner loops to obtain high probability of achieving an error bound: 1) *Frequency bucketization* involves using a random hash function to hash the κ non-zero Fourier coefficients of $\hat{u}$ into a small number of buckets, and 2) *Frequency estimation* finds the frequency locations of non-zero Fourier coefficients and their corresponding magnitudes. Information obtained from the two inner loops is combined in an outer loop to form the final output.

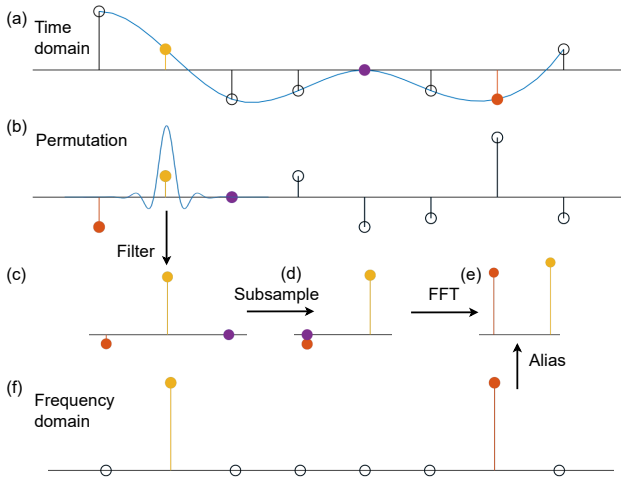

Fig. 2. An example of the SFFT with $N = 8$, $\sigma = 3$, $\tau = 6$, $w = 3$, $B = 2$ and $\kappa = 2$. (a) shows the input signal u, $N = 8$; (b) is the permuted u ($P_{3,6}u$); (c) after filtering with G to restrict the time domain length of $P_{3,6}u$ to 3; (d) subsampled $P_{3,6}u$ to 2 buckets to get v; (e) $\hat{v}$. the FFT of v; (f) The 2-sparse approximation of $\hat{u}$ that is bucketized into subfigure (e).

Fig. 2(a) to (e) illustrates the steps involved in frequency bucketization (FB). Let B be the number of buckets and is an integer that divides N; σ an integer invertible mod N; and κ the number of non-zero Fourier coefficients desired in the output. Fig. 2(b) is the permuted frequency spectrum, achieved via the time domain permutation operator $P_{\sigma,\tau}$, $\tau \in [0, N-1]$. If $(P_{\sigma,\tau}u)(i) = u((\sigma i + \tau) \mod N)$, then $\widehat{(P_{\sigma,\tau}u)}(\sigma i) = \hat{u}(i)e^{-i2\pi\tau}$ [11]. Fig. 2(c) represents the output of a w-dimensional filter function G, which is restricted to a subset of the input in both the time and frequency domain. In this work, a Dolph-Chebyshev function is used which has little leakage between buckets and this restricts the time-domain region of interest to $w = O(B \log \frac{N}{\delta})$ coordinates (δ is the maximum ripple in the passband or stopband), and performs bandpass filtering in the frequency domain [11].[1] Fig. 2(d) to (e) shows that the subsampled FFT $\hat{v} = \hat{u}(iN/B)$ of an N-dimensional vector u can be computed via the B-point FFT of $v = \sum_{j=0}^{N/B-1} u(i+Bj)$ for $i \in [0, B-1]$ [11].

III. S^{3}CA Algorithm

This section presents the S^{3}CA technique. A naive S^{3}CA implementation can be implemented by simply replacing the N_p N-point FFTs with SFFTs, with the input to the k$^{\text{th}}$ FFT being the $X_g(:,k)$ vector. This is shown by the solid line block diagram of Fig. 1, which would involve computing the entire matrix X_g, but not using all of it.

However, as described in the previous section, the FB step within each SFFT only requires w inputs, based on $P_{\sigma,\tau}$, where σ an integer invertible mod N, and $\tau \in [0, N-1]$, are both drawn from a random distribution. The indices of the w inputs form a set $W = \{i * \sigma + \tau \mod N \mid i \in [0, \ldots, w-1]\}$. We denote the union of all sets of indices required for the N_p SFFTs by W'. Our approach involves only computing W', which is significantly smaller than N.

[1]The support of the G filter, i.e. the coordinates of the non-zero coefficients, is limited to the interval $[-(w-1)/2, (w-1)/2]$, and computations outside of this interval are removed.

Algorithm 1 Modified SFFT pseudocode.

function FB(u,σ,τ,w,B,G,N) ▷ Frequency Bucketization
  for $i = 0$ to $w-1$ **do**
    $v[i \bmod B] \mathrel{+}= u[(i*\sigma+\tau) \bmod N]G[i]$
  $\hat{v} \leftarrow$ B-dimensional FFT(v)
  return $\hat{v}$
function SFFT($u, \kappa, B, L, G, d, N, \Sigma, \Upsilon$)
  for $r = 0$ to $L-1$ **do**
    $\hat{v} \leftarrow$FB($u, \frac{N}{B}, \Upsilon_{(2,r)}, B, B,$ ones$(B,1), N$)
    $T_r \leftarrow$ indices of 2κ largest elements of $\hat{v}$
      ▷ $T_r \subset [0, B-1]$
  $T = T_0 \cup \cdots \cup T_{L-1}$
  for $r = 0$ to $L-1$ **do** ▷ location loop
    $\hat{v} \leftarrow$ FB$(u, \Sigma_{(0,r)}, \Upsilon_{(0,r)}, w, B, G, N)$
    $J \leftarrow$ indices of $d\kappa$ largest elements of $\hat{v}$
    $I_r \leftarrow \{i \in [0, N-1] \mid h_\sigma(i) \in J, i \bmod B \in T\}$
      ▷ $h_\sigma(i) = round(\Sigma_{(0,r)} iB/N)$
  $I = I_0 \cup \cdots \cup I_{L-1}$
  $I' \leftarrow i$ values that occur frequently in sets I
  for $r = 0$ to $L-1$ **do** ▷ estimation loop
    $\hat{v} \leftarrow$FB$(u, \Sigma_{(2,r)}, \Upsilon_{(2,r)}, w, B, G, N)$
    $\hat{u}^r_{I'} \leftarrow$ estimate frequency spectrum from $\hat{v}, I'$ [11]
  $\hat{u}'_i = median(\{\hat{u}^r_i \mid i \in I'\})$
  return $\hat{u}'$

To achieve this, we describe a procedure COMPIDX, which precomputes a subset of indices W' for the channelizer, and corresponding arrays Σ of σ and Υ of τ for the SFFT. The channelizer now only computes the outputs $X_T(W', k)$ instead of $X_T(n, k)$, then the CDP, $X'_g = X_g(W', k)$, using Eq. (2). All CDP outputs are then used by the subsequent N_p N-point SFFTs. The output of S^{3}CA is a sparse matrix and only returns non-zero values and corresponding location information. An equivalent approach is using lazy evaluation to avoid computing unnecessary inputs to the SFFT.

Alg. 1 shows how we modified FB to use the precomputed σ and τ, with the SFFT updated to make use of this function. Alg. 2 gives the pseudocode for COMPIDX and S^{3}CA. x is the input signal with length of N, and N_P is the number of channelizers. COMPIDX, which is the dashed block in Fig. 1, randomly selects σ and τ required by our modified FB to compute W for each new input window. It then returns the set W' of all required indices, the array Σ of σ and the array Υ of τ.[2] In Fig. 1, each channelizer performs an independent N-point FFT. Consequently, in our implementation, in each of the different FB calls, the same σ and τ values are used for all k and the w inputs are $X_g(W, k)$. This necessitates a modified SFFT that can accommodate the shared σ and τ.

Table I compares the computational complexity of SSCA and S^{3}CA. Referring to Fig. 1 the SSCA channelizer requires a total of N evaluations of Eq. (2); and the FFT block N_P evaluations of Eq. (3) (using the FFT). In contrast for the S^{3}CA channelizer, the required number of N_p-point FFT evaluations is equal to the sampling complexity of the SFFT, $N_{SFFT} = O(\log N \sqrt[3]{N\kappa^2 \log N})$ [15].

[2]In Alg. 1 and 2, we present the loop value L and buckets value B for simplicity; performance can be improved with different values of L and B for the three for loops in SFFT.

Algorithm 2 S^{3}CA pseudocode.

procedure COMPIDX(L, w, B, N)
    ▷ Compute Indices for X'_g
  $\Upsilon \leftarrow zeros(3, L)$, $\Sigma \leftarrow zeros(2, L)$
  for $r = 0$ to $L-1$ **do**
    $\Upsilon_{(2,r)} \leftarrow$ uniform$(0, B-1)$
    $\Upsilon_{(0,r)}, \Upsilon_{(1,r)} \leftarrow$ uniform$(0, N-1)$
    $\Sigma_{(0,r)}, \Sigma_{(1,r)} \leftarrow 2*$uniform$(0, N/2-1)+1$
    $W^r_0 \leftarrow \{i * \Sigma_{(0,r)} + \Upsilon_{(0,r)} \bmod N \mid i \in [0, w-1]\}$
    $W^r_1 \leftarrow \{i * \Sigma_{(1,r)} + \Upsilon_{(1,r)} \bmod N \mid i \in [0, w-1]\}$
    $W^r_2 \leftarrow \{i * N/b_2 + \Upsilon_{(2,r)} \mid i \in [0, B-1]\}$
  $W' \leftarrow \{W^r_j \mid j \in \{0,1,2\}, r \in [0, L-1]\}$
  return W', Σ, Υ
procedure S^{3}CA(x, N, N_P, L, w, B, N, G)
  $W', \Sigma, \Upsilon \leftarrow$ COMPIDX(L, w, B, N)
  $X'_g \leftarrow X_g(W', k)$ ▷ Eq. 2, $k \in [-\frac{N_P}{2}, \frac{N_P}{2}-1]$
  for $k = -\frac{N_P}{2}$ to $\frac{N_P}{2}-1$ **do**
    $\hat{u}'_k \leftarrow$ SFFT$(X'_g, \kappa, B, L, G, d, N, \Sigma, \Upsilon)$
  $value, \alpha, f \leftarrow$ map$(\hat{u}')$
  return $value, \alpha, f$

TABLE I
COMPARISON OF COMPUTATIONAL COMPLEXITY BETWEEN SSCA AND S^{3}CA.

	SSCA	S^{3}CA
Channelizer	$O(NN_P \log N_P)$	$O(N_{SFFT} N_P \log N_P)$
$N_p\times$ FFT	$O(N_P N \log N)$	$O(N_P N_{SFFT})$
$S^\alpha_X(f)$	$O(NN_P(\log N_P + \log N))$	$O(N_{SFFT} N_P \log N_P)$

IV. RESULTS

We implemented the SSCA and S^{3}CA using the C programming language and the FFTW library [19]. Experiments were conducted using Ubuntu 20.04.6 LTS on an Intel(R) Xeon(R) Silver 4208 CPU running @ 2.10GHz with 256 GB of memory. All the code was compiled using g++ version 9.4.0 with "-O2" optimization flag.

A. Accuracy

Accuracy was tested using a direct-sequence spread-spectrum (DSSS) binary phase-shift keying (BPSK) signal with 10 dB signal-to-noise ratio (SNR), processing gain of 31, chip rate 0.25 and sample rate normalized to 1, in which case the cycle frequencies are multiples of the data rate (0.25/31). The Fig. 3 shows the SCD estimates with $N = 2^{20}$ and $N_P = 2^6$. We configure the remaining parameters of S^{3}CA in accordance with the default parameters outlined in the SFFT library [11]. Fig. 3(a) shows a 3-D plot of the largest κN_P magnitude SSCA outputs, $S_{X_{SSCA}}$, with its alpha profile corresponding to the largest alpha value over all frequencies below. Due to symmetry, the non-redundant interval of normalized cycle frequency, α, is in $[0, 1]$. To highlight the important details, the display area of the alpha profile is restricted to $[0, 0.25]$. Fig. 3(b) shows the S^{3}CA output, $S_{X_{S^3CA}}$, with sparsity parameter $\kappa = 80$, and its alpha profile

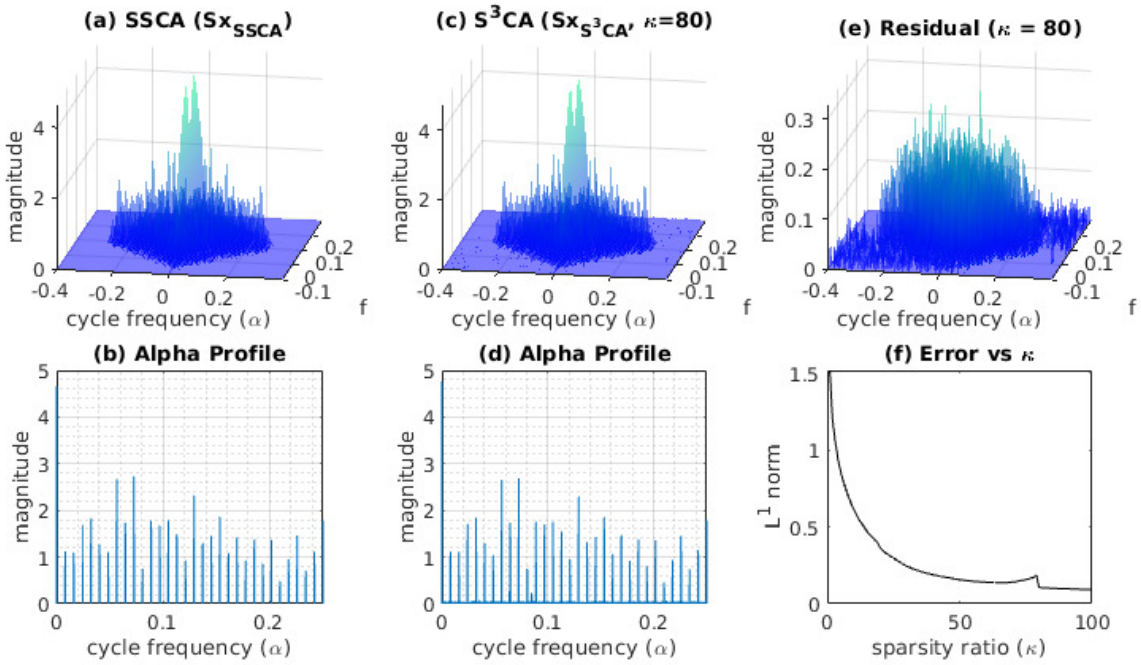

Fig. 3. SCD estimates and alpha profiles using SSCA (a) and (b), and S^{3}CA (c) and (d), their residual (e), and L^1-norm of the residue for different κ (f).

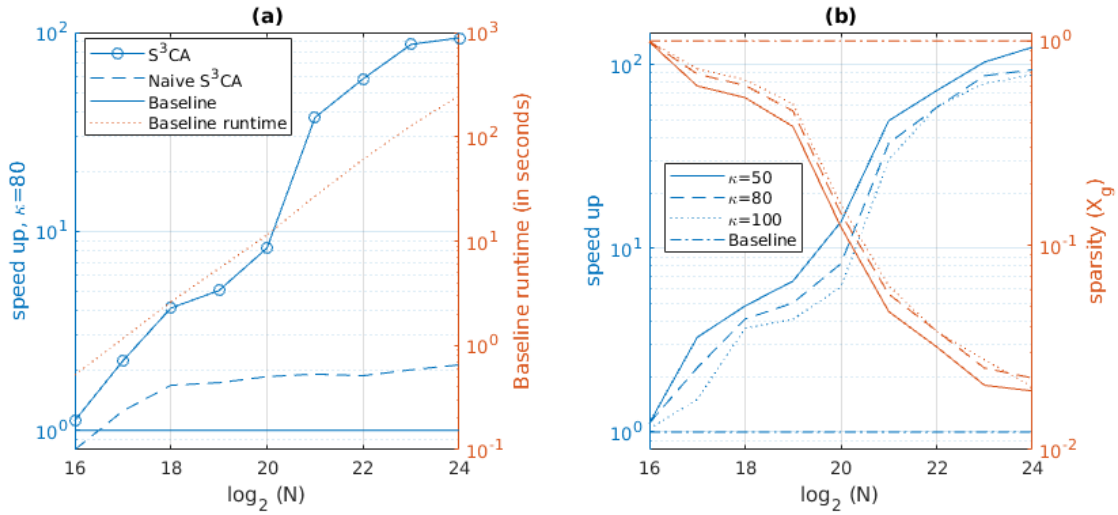

Fig. 4. (a) Speedup of the naive S^{3}CA and S^{3}CA compared with the conventional SSCA. (b) Speedup and X_g' sparsity of S^{3}CA for different values of κ.

in Fig. 3(d). In Fig. 3(c), the residual, $r = S_{X_{SSCA}} - S_{X_{S^3CA}}$, is shown together with the average L^1-norm of the residue, $\sum_i |r_i|/(\kappa N_P)$, below in Fig. 3(f). Again, good correspondence between the SSCA and S^{3}CA is observed.[3]

B. Speedup and storage optimization

The baseline in Fig. 4 is the conventional SSCA. The figure compares the speedup achieved by replacing the FFT with SFFT in SSCA, labeled as naive S^{3}CA, and the speedup obtained by S^{3}CA, for input window sizes from 2^{16} to 2^{24}. For an input size of 2^{24}, the S^{3}CA achieves a speedup that surpasses a factor of 90 when κ is 80, and more than 100 when κ is 50. The naive S^{3}CA achieves a more modest speedup of 2. The baseline runtime on our test computer is also provided.

[3]Verification based on a BPSK signal. The SCD estimate can be found in https://github.com/Jingyi-li/S3CA.

The sparsity of X_g' is $\mathbb{S} = |W'|/N$, where $|.|$ denotes the number of indices in W', hence the storage savings over the full X_g is approximately $1 - \mathbb{S}$. The Fig. 4(b) shows the speedup and the sparsity ratio of S^{3}CA for different κ. The output of the SSCA, has $N \times N_P$ values, whereas the output of S^{3}CA only has $\kappa \times N_P$ values.

V. Conclusion

In this paper, we presented a novel S^{3}CA method that utilizes the SFFT to achieve significant acceleration over the conventional SSCA, particularly for digital radio signals that are always sparse in cycle frequency. The speedup achieved was more than 30 for input windows of 2 million samples. Our S^{3}CA avoids unnecessary computations and employs a sparse CDP matrix to reduce memory requirements.

REFERENCES

[1] W. A. Gardner, A. Napolitano, and L. Paura, "Cyclostationarity: Half a century of research," *Signal processing*, vol. 86, no. 4, pp. 639–697, 2006.

[2] W. A. Gardner, "The spectral correlation theory of cyclostationary time-series," *Signal processing*, vol. 11, no. 1, pp. 13–36, 1986.

[3] A. Napolitano, "Cyclostationarity: New trends and applications," *Signal Processing*, vol. 120, pp. 385–408, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165168415003138

[4] W. A. Gardner, "Exploitation of spectral redundancy in cyclostationary signals," *IEEE Signal Processing Magazine*, vol. 8, pp. 14–36, Apr. 1991.

[5] R. S. Roberts, W. A. Brown, and H. H. Loomis, "Computationally efficient algorithms for cyclic spectral analysis," *IEEE Signal Processing Magazine*, vol. 8, no. 2, pp. 38–49, 1991.

[6] J. Antoni, "Cyclic spectral analysis in practice," *Mechanical Systems and Signal Processing*, vol. 21, no. 2, pp. 597–630, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327006001816

[7] J. Antoni, G. Xin, and N. Hamzaoui, "Fast computation of the spectral correlation," *Mechanical Systems and Signal Processing*, vol. 92, pp. 248–277, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327017300134

[8] W. A. Brown and H. H. Loomis, "Digital implementations of spectral correlation analyzers," *IEEE Transactions on Signal Processing*, vol. 41, no. 2, pp. 703–720, 1993.

[9] P. Borghesani and J. Antoni, "A faster algorithm for the calculation of the fast spectral correlation," *Mechanical Systems and Signal Processing*, vol. 111, pp. 113–118, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327018301869

[10] J. K. Alsalaet, "Fast averaged cyclic periodogram method to compute spectral correlation and coherence," *ISA Transactions*, vol. 129, pp. 609–630, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019057822000441

[11] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Simple and practical algorithm for sparse fourier transform," in *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '12. USA: Society for Industrial and Applied Mathematics, 2012, p. 1183–1194.

[12] ——, "Nearly optimal sparse fourier transform," in *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 563–578. [Online]. Available: https://doi.org/10.1145/2213977.2214029

[13] M. Frigo and S. G. Johnson, "FFTW Library version 3.3.10," accessed: July 4, 2023. [Online]. Available: http://www.fftw.org/download.html

[14] E. April, *On the Implementation of the Strip Spectral Correlation Algorithm for Cyclic Spectrum Estimation*, ser. DREO technical note. Defence Research Establishment Ottawa, 1994. [Online]. Available: https://books.google.com.au/books?id=7QD7MwEACAAJ

[15] H. Hassanieh, *The Sparse Fourier Transform: Theory and Practice*. Association for Computing Machinery and Morgan & Claypool, 2018, vol. 19.

[16] W. A. Brown, "On the theory of cyclostationary signals," Ph.D. dissertation, University of California Davis, 1987.

[17] W. A. Gardner, *Cyclostationarity in communications and signal processing*. New York: IEEE Press, 1994.

[18] D. Katabi, H. Hassanieh, E. Price, and P. Indyk, "Sfft website." [Online]. Available: https://groups.csail.mit.edu/netmit/sFFT/

[19] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".