Sliding conjugate symmetric sequency-ordered complex Hadamard transform: fast algorithm and applications

Jiasong Wu, Lu Wang, Lotfi Senhadji, Huazhong Shu

▶ **To cite this version:**

Jiasong Wu, Lu Wang, Lotfi Senhadji, Huazhong Shu. Sliding conjugate symmetric sequency-ordered complex Hadamard transform: fast algorithm and applications. European Signal Processing Conference (EUSIPCO), Jul 2010, AALBORG, Denmark. pp.1742-6. inserm-00530948

HAL Id: inserm-00530948

https://inserm.hal.science/inserm-00530948

Submitted on 5 Dec 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Author manuscript, published in "European Signal Processing Conference (EUSIPCO), AALBORG : Denmark (2010)"

inserm-00530948, version 1 - 5 Dec 2013

SLIDING CONJUGATE SYMMETRIC SEQUENCY-ORDERED COMPLEX HADAMARD TRANSFORM: FAST ALGORITHM AND APPLICATIONS

Jiasong Wu[1,2,3,4], *Lu Wang*[1,4], *Lotfi Senhadji*[2,3,4], *and Huazhong Shu*[1,4]

[1]LIST, Southeast University, 2 Sipailou, 210096, Nanjing, China
phone: +86-25-83794249, fax: +86-25-83792698, email: jswu@seu.edu.cn, wanglu@seu.edu.cn, shu.list@seu.edu.cn
[2]INSERM, U642, Rennes, F-35000, France
[3]LTSI, Université de Rennes 1, Campus Beaulieu, Rennes, F-35042, France
phone: +33-2 23235577, fax: +33-2 23236917, email: lotfi.senhadji@univ-rennes1.fr
[4]Centre de Recherche en Information Biomédicale Sino-français (CRIBs), Rennes, France
www.imagetech.com.cn; www.ltsi.univ-rennes1.fr

ABSTRACT

This paper presents a fast algorithm for the computation of forward and backward sliding conjugate symmetric sequency-ordered complex Hadamard transform (CSSCHT). The forward CSSCHT algorithm calculates the values of window i+N/4 from those of window i and one length-N/4 CSSCHT, one length-N/4 Walsh Hadamard transform (WHT) and one length-N/4 modified WHT. The backward CSSCHT algorithm can be obtained by transposing the signal flow graph of that of the forward one. The proposed algorithm requires O(N) arithmetic operations, which is more efficient than the block-based algorithm and those based on the sliding FFT and the sliding DFT. The applications of the sliding CSSCHT in spectrum estimation and transform domain adaptive filtering (TDAF) are also provided with supporting simulation results.

1. INTRODUCTION

The discrete orthogonal transforms including discrete Fourier transform (DFT), discrete cosine transform (DCT), discrete Hartley transform (DHT), and Walsh-Hadamard transform (WHT) play an important role in the fields of digital signal processing, filtering and communications [1, 2]. Recently, Aung *et al*. introduced a new transform named the Conjugate Symmetric Sequency-ordered Complex Hadamard Transform (CSSCHT) [3], which can be an alternative of DFT and DCT in some applications needing lower computational complexity, such as spectrum estimation and image compression. A fast decimation-in-sequency (DIS) block-based algorithm was also reported in [3], which requires $N/2-1$ multiplications with the imaginary number j, $2N\log_2 N$ real additions and $2N$ memory for length-N CSSCHT.

When dealing with a nonstationary process, such as speech, radar, biomedical, and communication signals, the commonly used method is sliding orthogonal transform (also called short time orthogonal transform), whose computation is an intensive task. Therefore, many fast algorithms were proposed [4-7]. Besides the commonly used sliding FFT [4] and sliding DFT [5], the sliding WHT [6-8] was also attractive in the real-time pattern matching applications [9]. A fast algorithm, which decomposes a length-N WHT into two length-$N/2$ WHTs plus $4N-4$ real additions and $2N(\log_2 N-1)$ size of memory, for the sliding WHT was proposed in [6]. Ben-Artzi *et al.* [7] proposed a gray code kernel WHT algorithm, which requires $4N$ real additions and $4N$ size of memory. Ouyang and Cham [8] presented a more efficient algorithm to compute the sliding WHT, which derives the length-N WHT from two length-$N/4$ WHTs plus $3N+2$ real additions and $3N$ size of memory. Note that the computational complexity and the memory storage requirements are considered for complex input data.

Inspired by a research work presented in [8], we propose in this paper a fast algorithm for the computation of sliding CSSCHT, which computes the values of window $i+N/4$ from those of window i and one length-$N/4$ CSSCHT, one length-$N/4$ WHT and one length-$N/4$ modified WHT.

2. PRELIMINARY

Let $\mathbf{X}_N(i)=[x_i, x_{i+1}, \ldots, x_{i+N-1}]^T$ and $\mathbf{Y}_N(i)=[y_i, y_{i+1}, \ldots, y_{i+N-1}]^T$ be respectively the complex input vector and the transformed complex vector of the i^{th} window, where T denotes the transposition, and let $N = 2^n$, $n \geq 1$, the length-N forward and backward sliding CSSCHTs are defined as [3, 10]

$$\mathbf{Y}_N(i) = \mathbf{H}_N \mathbf{X}_N(i), \tag{1}$$

$$\mathbf{X}_N(i) = \frac{1}{N}\mathbf{H}_N^H \mathbf{Y}_N(i), \tag{2}$$

where the superscript H denotes the Hermitian transposition. $\mathbf{H}_N$ is the order-N CSSCHT matrix whose elements are given by

$$h(a,b) = (-1)^{\tilde{\mathbf{A}}\bullet\mathbf{B}}(-j)^{\hat{\mathbf{A}}\bullet\mathbf{B}}, 0 \leq a,b \leq 2^n-1, n=\log_2 N, \tag{3}$$

where $\tilde{\mathbf{A}} = (\tilde{a}_0, \tilde{a}_1, ..., \tilde{a}_{n-1})$, $\hat{\mathbf{A}} = (\hat{a}_0, \hat{a}_1, ..., \hat{a}_{n-1})$, and $\mathbf{B} = (b_0, b_1, ..., b_{n-1})$. The dot '$\bullet$' denotes the scalar product of two vectors. a_r and b_r are respectively the binary representation of a and b, $r = 0, 1, ..., n-1$, being the index of the binary bit position. $\tilde{a}_r$ is a binary gray code of the bit reversal of a_r and $\hat{a}_r$ is the r^{th} bit of the binary bits of the highest power of 2 in $c(a)/2$ where $c(a)$ is the decimal number obtained through a bit-reversed conversion of the decimal a.

Let $\mathbf{H}_N = \left[\mathbf{H}_N(0), \mathbf{H}_N(1), ..., \mathbf{H}_N(N-1)\right]$,

inserm-00530948, version 1 - 5 Dec 2013

	x_i	x_{i+1}	x_{i+2}	x_{i+3}	x_{i+4}	Proposed algorithm
$y_4(0,i)$	1	1	1	1		$y_4(0,i+1)= y_4(0,i)-$
$y_4(0,i+1)$		1	1	1	1	(x_i-x_{i+4})
$y_4(1,i)$	1	j	-1	$-j$		$y_4(1,i+1)=-j[y_4(1,i)-$
$y_4(1,i+1)$		1	j	-1	$-j$	$(x_i-x_{i+4})]$
$y_4(2,i)$	1	-1	1	-1		$y_4(2,i+1)=-[y_4(2,i)-$
$y_4(2,i+1)$		1	-1	1	-1	$(x_i-x_{i+4})]$
$y_4(3,i)$	1	$-j$	-1	j		$y_4(3,i+1)=j[y_4(3,i)-$
$y_4(3,i+1)$		1	$-j$	-1	j	$(x_i-x_{i+4})]$

Table 1 - Fast algorithm for length-4 CSSCHT

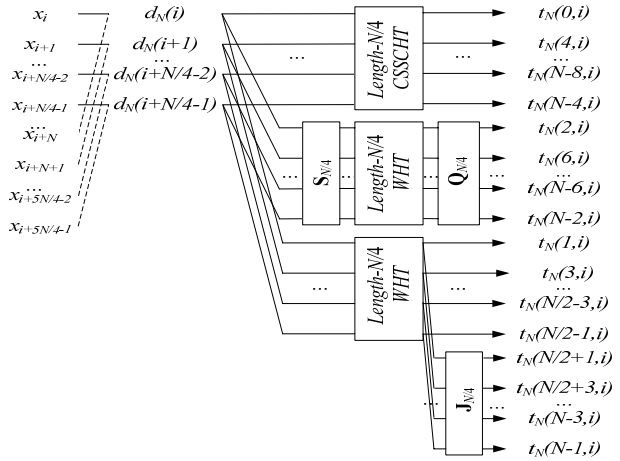


Figure 1 - Signal flow graph of the length-N sliding CSSCHT transform

$$\mathbf{H}_N^{1/m}=\left[\mathbf{H}_N(0),\mathbf{H}_N(1),...,\mathbf{H}_N\left(\frac{N}{m}-1\right)\right] \quad (4)$$
$$=\left[\mathbf{H}_N^{1/m}(0),\mathbf{H}_N^{1/m}(1),...,\mathbf{H}_N^{1/m}(N-1)\right]^T, m=2,4.$$

where $\mathbf{H}_N(k)$, k = 0, 1, …, N–1, is the k^{th} column of CSSCHT matrix, $\mathbf{H}_N^{1/m}(k)$, k = 0, 1, …, N–1, is the k^{th} row of $\mathbf{H}_N^{1/m}$, and the subscript T denotes the transpose.

Let $y_N(k, i)$ be the k^{th} CSSCHT projection value for the i^{th} window :

$$y_N(k,i)=\mathbf{H}_N^T(k)\mathbf{X}_N(i),$$

for k = 0, 1, …N–1; i = 0, 1, …, M–N, $N=2^n$, $n\geq 1$

where M is the length of the input data sequence.

3. FAST ALGORITHMS FOR SLIDING CSSCHT

In this section, we derive a fast algorithm for computing the sliding CSSCHT.

3.1 Fast Algorithm for N=4

The proposed algorithm is shown in Table 1, from which we have

$$y_4(k,i+1)=(-j)^k\left[y_4(k,i)-t_4(k,i)\right],k=0,1,2,3$$
$$\left[t_4(0,i),t_4(1,i),t_4(2,i),t_4(3,i)\right]^T=\mathbf{H}_4^{1/4}\left[d_4(i)\right]$$
$$=\left[d_4(i),d_4(i),d_4(i),d_4(i)\right]^T, d_4(i)=x_i-x_{i+4}.$$

where $\mathbf{H}_4^{1/4}$ is defined in (4). Therefore, 2 multiplications with j, 10 real additions, and a memory size of 10 are needed.

3.2 Fast Algorithm for N=8

The proposed algorithm is shown in Table 2, from which we have

$$y_8(k,i+2)=(-j)^k\left[y_8(k,i)-t_8(k,i)\right],k=0,1,...,7.$$
$$\left[t_8(0,i),t_8(1,i),...,t_8(7,i)\right]^T=\mathbf{H}_8^{1/4}\left[d_8(i),d_8(i+1)\right]^T,$$
$$\mathbf{H}_8^{1/4}=\mathbf{P}_8\begin{bmatrix}\mathbf{H}_4^{1/2}\\ \mathbf{W}_2\\ \mathbf{J}_2\mathbf{W}_2\end{bmatrix}=\mathbf{P}_8\begin{bmatrix}\mathbf{H}_2\\ \mathbf{W}_2\mathbf{S}_2\\ \mathbf{W}_2\\ \mathbf{J}_2\mathbf{W}_2\end{bmatrix},\mathbf{S}_2=\begin{bmatrix}1&\\ &j\end{bmatrix},$$
$$d_8(i+u)=x_{i+u}-x_{i+8+u},\ u=0,1.$$

where $\mathbf{P}_8$ is defined in (8). $\mathbf{J}_2$ and $\mathbf{W}_2$ are described in the following subsection. Therefore, 5 multiplications with j, 26 real additions, and 36 size of memory are needed.

3.3 Fast Algorithm for $N=2^n$, $n\geq 3$

By using the same strategy as for N=4 and N=8, we have

$$y_N(k,i+N/4)=(-j)^k\left[y_N(k,i)-t_N(k,i)\right], \quad (5)$$
$$k=0,1,...,N-1,$$
$$\left[t_N(0,i),t_N(1,i),...,t_N(N-1,i)\right]^T$$
$$=\mathbf{H}_N^{1/4}\left[d_N(i),d_N(i+1),...,d_N(i+N/4-1)\right]^T$$

$$\mathbf{H}_N^{1/4}=\mathbf{P}_N\begin{bmatrix}\mathbf{H}_{N/2}^{1/2}\\ \mathbf{W}_{N/4}\\ \mathbf{J}_{N/4}\mathbf{W}_{N/4}\end{bmatrix}=\mathbf{P}_N\begin{bmatrix}\mathbf{H}_{N/4}\\ \mathbf{Q}_{N/4}\mathbf{W}_{N/4}\mathbf{S}_{N/4}\\ \mathbf{W}_{N/4}\\ \mathbf{J}_{N/4}\mathbf{W}_{N/4}\end{bmatrix} \quad (6)$$

$$\mathbf{S}_N=\begin{bmatrix}\mathbf{I}_{N/2}&\\ &j\mathbf{I}_{N/2}\end{bmatrix}$$

$$d_N(i+u)=x_{i+u}-x_{i+N+u},\ u=0,1,\ldots,N/4-1, \quad (7)$$

$$[x_i,\ x_{i+1},\ \ldots,\ x_{i+N-1}]^T=\mathbf{P}_N\times[x_i,\ x_{i+4},\ \ldots,\ x_{i+N-4},\ x_{i+2},\ x_{i+6},\ldots,x_{i+N-2},$$
$$x_{i+1},\ x_{i+3},\ \ldots,\ x_{i+N/2-1},\ x_{i+N/2+1},\ x_{i+N/2+2},\ \ldots,\ x_{i+N-1}]^T \quad (8)$$

$$[x_i,\ x_{i+1},\ \ldots,\ x_{i+N-1}]^T=\mathbf{Q}_N\times$$
$$[x_i,\ x_{i+N-1},\ x_{i+1},\ x_{i+N-2},\ldots,\ x_{i+N/2-1},\ x_{i+N/2+1}]^T \quad (9)$$

where $\mathbf{H}_N^{1/4}$ and $\mathbf{H}_N^{1/2}$ are defined in (4). $\mathbf{W}_N$ is the N^{th} order WHT matrix. $\mathbf{I}_N$ is the identity matrix and $\mathbf{J}_N$ is the reverse identity matrix. Figure 1 shows the signal graph of the proposed algorithm, whose computational complexity and memory storage requirement are analyzed as follows (assuming that the algorithm is implemented in parallel):

1) The computation of (7) for u = N/4–1 needs only 2 real addition. Note that the values of $d_N(i+u)$, u = 0, 1,…, N/4–2, have already been obtained during the computation of $y_N(k,i+v)$, v = 1, 2,…, N/4–1, respectively. A memory size of N/2 is required for storing $d_N(i+u)$, u = 0, 1, …, N/4–1. The input x_{i+u} and x_{i+N+u} for u = 0, 1, …, N/4–1, needs N memory, which can be released after performing (7) since it will not be used in the following steps.

2) The computation of (6) needs one length-N/4 CSSCHT, one length-N/4 WHT, which can be computed by [8], one length-N/4 modified WHT ($\mathbf{W}_{N/4}\mathbf{S}_{N/4}$), which can be computed by [6] plus N/8 multiplications with j (multiplied by $\mathbf{S}_{N/4}$).

inserm-00530948, version 1 - 5 Dec 2013

	x_i	x_{i+1}	x_{i+2}	x_{i+3}	x_{i+4}	x_{i+5}	x_{i+6}	x_{i+7}	x_{i+8}	x_{i+9}	Proposed algorithm
$y_8(0,i)$	1	1	1	1	1	1	1	1			$y_8(0,i+2)=$
$y_8(0,i+2)$			1	1	1	1	1	1	1	1	$y_8(0,i)-[(x_i - x_{i+8})+(x_{i+1} - x_{i+9})]$
$y_8(1,i)$	1	1	j	j	-1	-1	$-j$	$-j$			$y_8(1,i+2)=$
$y_8(1,i+2)$			1	1	j	j	-1	-1	$-j$	$-j$	$-j\{y_8(1,i)-[(x_i - x_{i+8})+(x_{i+1} - x_{i+9})]\}$
$y_8(2,i)$	1	j	-1	$-j$	1	j	-1	$-j$			$y_8(2,i+2)=$
$y_8(2,i+2)$			1	j	-1	$-j$	1	j	-1	$-j$	$-\{y_8(2,i)-[(x_i - x_{i+8})+j(x_{i+1} - x_{i+9})]\}$
$y_8(3,i)$	1	-1	$-j$	j	-1	1	j	$-j$			$y_8(3,i+2)=$
$y_8(3,i+2)$			1	-1	$-j$	j	-1	1	j	$-j$	$j\{y_8(3,i)-[(x_i - x_{i+8})-(x_{i+1} - x_{i+9})]\}$
$y_8(4,i)$	1	-1	1	-1	1	-1	1	-1			$y_8(4,i+2)=$
$y_8(4,i+2)$			1	-1	1	-1	1	-1	1	-1	$y_8(4,i)-[(x_i - x_{i+8})-(x_{i+1} - x_{i+9})]$
$y_8(5,i)$	1	-1	j	$-j$	-1	1	$-j$	j			$y_8(5,i+2)=$
$y_8(5,i+2)$			1	-1	j	$-j$	-1	1	$-j$	j	$-j\{y_8(5,i)-[(x_i - x_{i+8})-(x_{i+1} - x_{i+9})]\}$
$y_8(6,i)$	1	$-j$	-1	j	1	$-j$	-1	j			$y_8(6,i+2)=$
$y_8(6,i+2)$			1	$-j$	-1	j	1	$-j$	-1	j	$-\{y_8(6,i)-[(x_i - x_{i+8})-j(x_{i+1} - x_{i+9})]\}$
$y_8(7,i)$	1	1	$-j$	$-j$	-1	-1	j	j			$y_8(7,i+2)=$
$y_8(7,i+2)$			1	1	$-j$	$-j$	-1	-1	j	j	$j\{y_8(7,i)-[(x_i - x_{i+8})+(x_{i+1} - x_{i+9})]\}$

Table 2 - Fast algorithm for length-8 CSSCHT.

Proposed algorithm	*Muls(j)*	$5N/6$-$4/3$, $N=2^n$, $n=4,6,\ldots$ $5N/6$-$5/3$, $N=2^n$, $n=3,5,\ldots$
	Adds	$17N/3+6\log_2 N-86/3$, $N=2^n$, $n=4,6,\ldots$ $17N/3+6\log_2 N-103/3$, $N=2^n$, $n=3,5,\ldots$
	Me	$N/2+\max\left\{7N/2,\ Me_{N/4}^{CSSCHT}+N\left(2\log_2 N-3\right)/4\right\}$
Algorithm [3]	*Muls(j)*	$N/2$-1
	Adds	$2N\log_2 N$
	Me	$2N$
Sliding FFT [4]	*Muls*	$4N$–$8\log_2 N$
	Muls(j)	$\log_2 N$–1
	Adds	$4N$–$4\log_2 N$-2
	Me	$2N\log_2 N$-8
Sliding DFT [5]	*Muls*	$4N$-16
	Muls(j)	2
	Adds	$4N$-6
	Me	$4N$-6

Table 3 - Comparison results of the proposed algorithms with the block-base one in [3], the sliding FFT in [4] and the sliding DFT in [5] for $N = 2^n$, $n \geq 4$. "*Muls*" represents real multiplications, "*Muls (j)*" means multiplication with j , "*Adds*" means real additions. "*Me*" denotes memory (words).

Size $3N/2$ memory is needed for storing the values $t_N(i+u)$, $u \in [0, N\text{-}1]$ but $u \notin \{N/2+1, N/2+3,\ldots, N\text{-}1\}$, since $\mathbf{J}_{N/4}\mathbf{W}_{N/4}$ is just row change operations of $\mathbf{W}_{N/4}$. We also assume that the memory storage requirements of length-$N/4$ CSSCHT, length-$N/4$ WHT, and length-$N/4$ modified WHT are $Me_{N/4}^{CSSCHT}$, $Me_{N/4}^{WHT}$, and $Me_{N/4}^{MWHT}$, respectively. Note that the multiplication by j or $-j$ can be realized by switching the real and imaginary parts of the input with one sign changing, so that there is no memory requirement.

3) The computation of (5) needs $N/2$ multiplications with j and $2N$ real additions. The values of $y_N(k, i)$, $y_N(k, i+1),\ldots$, $y_N(k, i+N/4-1)$ can be obtained by zero padding method ($5N/4-1$ zeros) as proposed in [9]. For the implementation, we first distribute $2N$ memory for $y_N(k, i)$, $k = 0, 1, \ldots, N-1$, which is then overlaid by $y_N(k, i+N/4)$, $k = 0, 1, \ldots, N-1$ after performing (5). Thus, the computational complexity and memory requirement of the proposed algorithm is given by

$$M_N^{CSSCHT} = M_{N/4}^{CSSCHT} + 5N/8$$

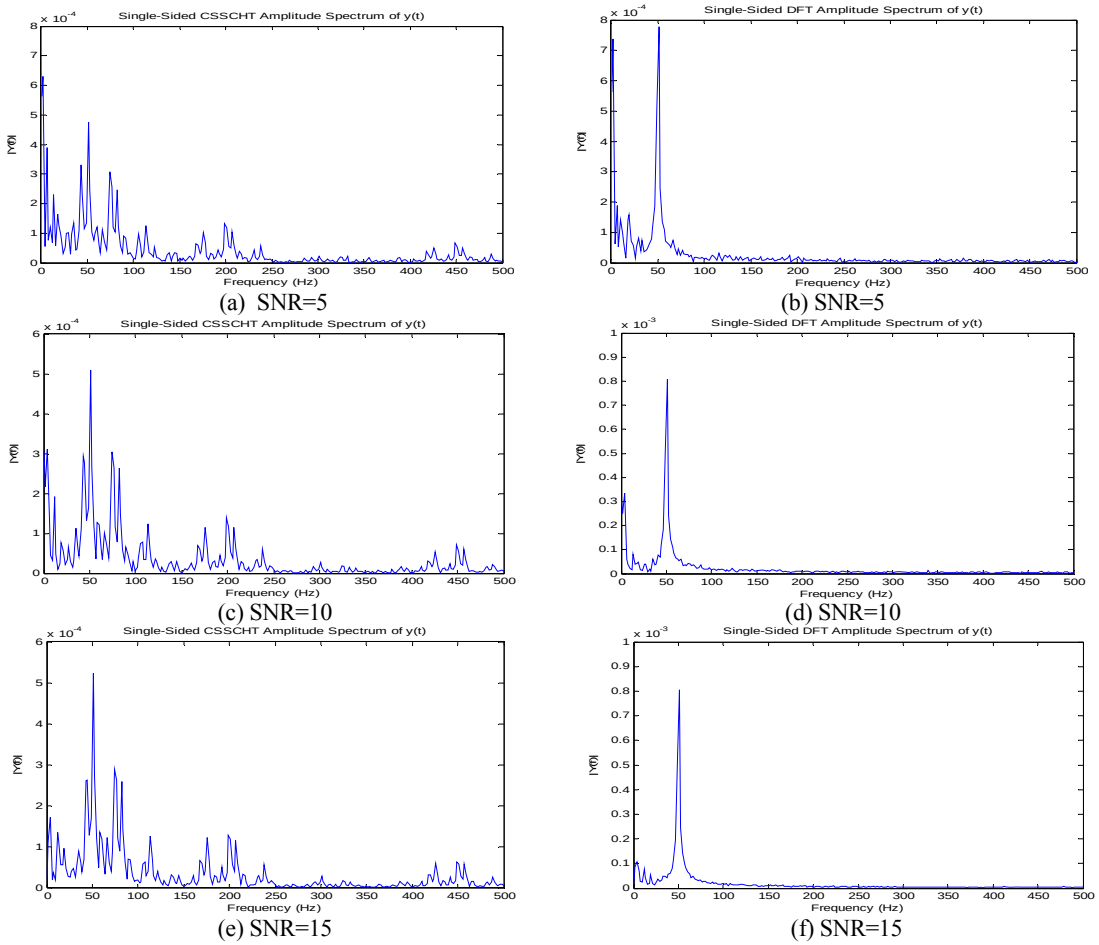
$$A_N^{CSSCHT} = A_{N/4}^{CSSCHT} + A_{N/4}^{WHT} + A_{N/4}^{MWHT} + 2N + 2$$

$$Me_N^{CSSCHT} = \frac{N}{2} + \max\left\{\frac{7}{2}N, Me_{N/4}^{CSSCHT} + Me_{N/4}^{WHT} + Me_{N/4}^{MWHT}\right\}$$

with the initial values $M_4^{CSSCHT} = 2, A_4^{CSSCHT} = 10;$ $M_8^{CSSCHT} = 5, A_8^{CSSCHT} = 26;$ $Me_4^{CSSCHT} = 10, Me_8^{CSSCHT} = 36;$ and $Me_4^{WHT} = 10, Me_8^{WHT} = 24$.

The comparison results of the proposed algorithm and the algorithms in [3-5] are shown in Tables 3 and 4. It can be seen from the tables that the proposed algorithm reduces significantly the real additions compared to the algorithm in [3], but at the cost of a little more memory requirement. The proposed algorithm is also more efficient than the sliding FFT in [4] and sliding DFT in [5]. This is because the proposed algorithm only needs the multiplications with j and real additions, and can save the memory for storing the twiddle factors. For comparison purpose, Tables 3 and 4 give the real multiplications, multiplications with j and real additions where one complex multiplication is implemented by four real multiplications and two real additions. It should be noted that for the computation of $y_N(k, i)$, $k = 0, 1, \ldots N-1$; $i = 0, 1, \ldots, M-N$, $N = 2^n$, $n \geq 1$, the proposed algorithm should perform $(M+N/4)$ times the module shown in Figure 1. However, the sliding DFT [5] and the block-base algorithm [3] are only performed M and $M-N+1$ times, respectively. Sliding FFT [4] should perform a radix-2 FFT algorithm first and then perform $M-N$ times the sliding algorithm. The computation of the backward CSSCHT, if ignoring the normalization factor $1/N$ in (2), can be simply realized by transposing the signal flow graph of the forward CSSCHT.

inserm-00530948, version 1 - 5 Dec 2013



(a) SNR=5 (b) SNR=5

(c) SNR=10 (d) SNR=10

(e) SNR=15 (f) SNR=15

Figure 2 - Spectrogram of a 50Hz sinusoidal signal with additive white noise: (a)(c)(e) sliding CSSCHT (b)(d)(f) sliding DFT

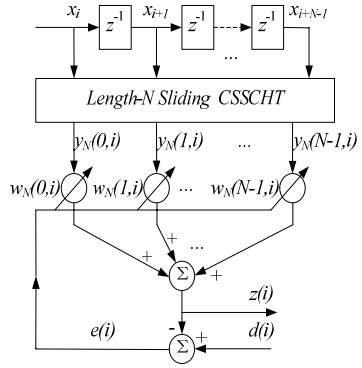


Figure 3 - Block diagram of CSSCHT domain adaptive filtering

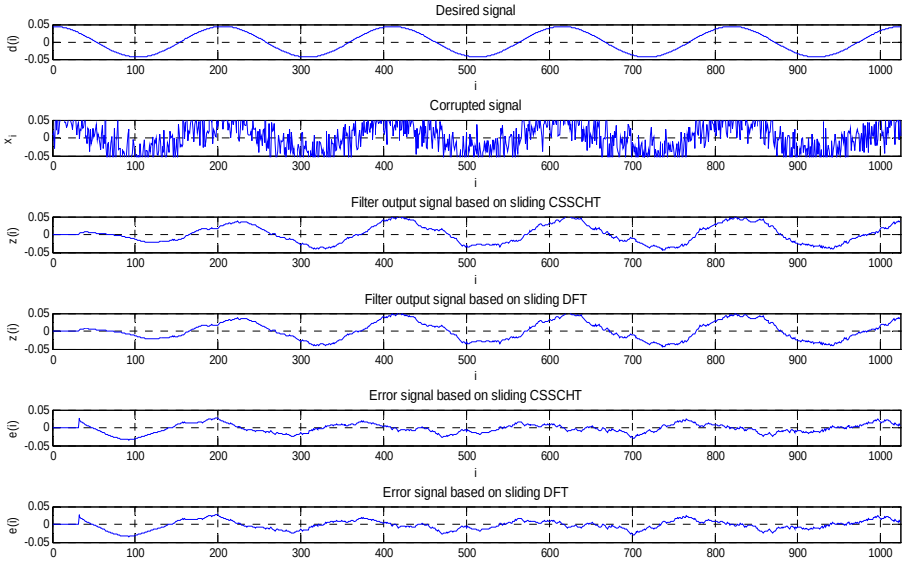


Figure 4 - Transform domain adaptive filtering using sliding CSSCHT and sliding DFT

Table 4- The quantitative comparison results of the proposed algorithms with other three algorithms ([3]-[5]) for $N = 2^n$, $n \geq 4$.

N	Proposed algorithm			Algorithm [3]			Sliding FFT [4]				Sliding DFT [5]			
	Muls (j)	*Adds*	*Me*	*Muls(j)*	*Adds*	*Me*	*Muls*	*Muls (j)*	*Adds*	*Me*	*Muls*	*Muls (j)*	*Adds*	*Me*
4	2	10	10	1	16	8	0	1	6	8	0	2	10	10
8	5	25	36	3	48	16	8	2	18	40	16	2	26	26
16	12	86	64	7	128	32	32	3	46	120	48	2	58	58
32	25	177	128	15	320	64	88	4	106	312	112	2	122	122

inserm-00530948, version 1 - 5 Dec 2013

4. APPLICATION EXAMPLES

In this section, we will give some application examples of the sliding CSSCHT.

4.1 Spectrum estimation

CSSCHT is applied in spectrum estimation of a sinusoidal signal. In the proposed spectrum analysis, the window shifts one step each time leading to a highly overlapping window, which is exactly the sliding CSSCHT/DFT. Note that the use of a sliding window can decrease the variance of the estimation when compared to that of using a single window. Figure 3 shows the single-sided amplitude spectrum of 512-point sliding CSSCHT and sliding DFT of a 50Hz sinusoidal signal with additive white Gaussian noise and signal-to-noise ratio (SNR) being from 5 dB to 15 dB. The length of the input data sequence is $M = 4096$ and the sampling frequency is 1000 Hz. We compute the transformations of the whole sliding windows, and then calculate the average of those coefficients to obtain the estimated spectrum. From the figure, it can be seen that the CSSCHT frequency magnitude is matched with that of DFT. The desired peaks occur in the same locations as that of DFT even though there are some spurious spikes in the CSSCHT spectrum. Therefore, it is worth considering sliding CSSCHT for spectrum analysis instead of sliding DFT when it is necessary to achieve significant hardware savings and reduced computational time [3].

4.2 CSSCHT domain LMS adaptive filter

Transform domain least-mean-square adaptive filters (TDLMSAF), introduced by Narayan *et al.*[11], exploit the de-correlation properties of some well-known signal transforms such as DFT, DCT, DHT and WHT, in order to pre-whiten the input data and speed up filter convergence [12].

Similar to the DFT domain LMS adaptive filter [11, 12], the CSSCHT domain LMS adaptive filter algorithm, shown in Figure 3, is described as follows:

$$\mathbf{Y}_N(i) = \mathbf{H}_N \mathbf{X}_N(i), z(i) = \mathbf{W}_N^H(i)\mathbf{Y}_N(i), e(i) = d(i) - z(i),$$

$$\mathbf{W}_N(i+1) = \mathbf{W}_N(i) + 2\mu D^{-1}(i)e(i)\mathbf{Y}_N^*(i),$$

where * denotes the complex conjugate operator, $\mathbf{X}_N(i)$ is the input signal vector, $\mathbf{Y}_N(i)=[y_N(0,i), y_N(1,i), \ldots, y_N(N\text{-}1,i)]^T$ is the CSSCHT domain coefficients. $\mathbf{W}_N(i)=[w_N(0,i), w_N(1,i), \ldots, w_N(N\text{-}1,i)]^T$ is the adaptive weight vector. $z(i)$, $d(i)$, $e(i)$ are the filter output signal, the desired signal, the error signal, respectively. μ is a positive step-size and $D(i)$ is a diagonal matrix of the estimated input powers which is given by

$$D(i) = diag\{\sigma^2(k,i)\}, k = 0,\ldots,N-1$$

$$\sigma^2(k,i) = \beta\sigma^2(k,i-1) + (1-\beta)\left|y_N(k,i)\right|^2, 0 < \beta < 1$$

The sliding CSSCHT and sliding DFT are compared in a simulation for this type of adaptive filter. Using a 32-tap filter, a 5 Hz sinusoid with 1024 samples per second corrupted by Gaussian white noise with SNR equal to 0 db is processed through the filter. The parameters are set as $\mu = 0.01$ and $\beta = 0.9$. Figure 4 shows the desired signal, corrupted signal and the filtered signals by TDLMSAF using sliding DFT and sliding CSSCHT. It can be seen that the result of TDLMSAF using sliding CSSCHT is almost the same as that obtained with sliding DFT.

5. CONCLUSION

In this paper, we have presented a fast algorithm for computing the forward and backward sliding CSSCHT. The arithmetic complexity order of the proposed algorithms is N, a factor of $\log_2 N$ improvement is made over the block-based algorithm for the length-N CSSCHT. The proposed algorithm is also more efficient than the sliding FFT algorithm and the sliding DFT algorithm. The applications of sliding CSSCHT to spectrum estimation and transform domain adaptive filtering (TDAF) have been discussed.

6. ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China under Grant 60873048, by the National Basic Research Program of China under Grant 2010CB732503, by a Program of Jiangsu Province under Grant SBK200910055 and the Natural Science Foundation of Jiangsu Province under Grant BK2008279. This work is also supported by the "Eiffel Doctorate" excellence grant of the French Ministry of Foreign and European Affairs. The authors are also thankful to Dr. Aung for providing his MATLAB code constructing the CSSCHT matrix and Dr. Ouyang for helpful discussion.

REFERENCES

[1] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*. New York: Springer, 1975.

[2] O.K. Ersoy, "A comparative review of real and complex Fourier-related transforms," *Proc. IEEE*, vol. 82, pp. 429-447, 1994.

[3] A. Aung, B. P. Ng, and S. Rahardja, "Conjugate Symmetric Sequency-Ordered Complex Hadamard Transform," *IEEE Trans. Signal Process.*, vol. 57, pp. 2582–2593, Jul. 2009.

[4] B. Farhang-Boroujeny and Y. C. Lim, "A comment on the computational complexity of sliding FFT," *IEEE Trans. Circuits Syst.,* vol. 39, pp. 875-876, Dec. 1992.

[5] E. Jacobsen and R. Lyons, "The sliding DFT," *IEEE Signal Process. Mag.,* vol. 20, pp. 74–80, Mar. 2003.

[6] Y. Hel-Or and H. Hel-Or, "Real time pattern matching using projection kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, pp. 1430-1445, Sept. 2005.

[7] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or, "The gray-code filter kernels," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 29, pp.382-393, Mar. 2007.

[8] W. Ouyang and W.K. Cham, "Fast algorithm for Walsh Hadamard transform on sliding windows," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 32, pp. 165-171, Jan. 2010.

[9] Yair Moshe and H. Hel-Or, "Video Block Motion Estimation Based on Gray-Code Kernels," *IEEE Trans. Image Process.*, vol. 18, pp. 2243-2254, Oct. 2009.

[10] V. Kober, "Fast algorithms for the computation of sliding discrete sinusoidal transforms," *IEEE Trans. Signal Process.*, vol. 52, pp. 1704–1710, Jun. 2004.

[11]S.S. Narayan, A.M. Peterson, and M.J. Narashima, "Transform domain LMS algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, pp. 609-615, Jun. 1983.

[12]A.H. Sayed, *Adaptive Filters*. New York: Wiley, 2008.