Eighth International Workshop on Optimal Codes and Related Topics
July 10-14, 2017, Sofia, Bulgaria *pp. 55-60*

On Walsh transform and matrix factorization[1]

Iliya Bouyukliev iliyab@math.bas.bg
Paskal Piperkov ppiperkov@math.bas.bg
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences

Abstract. In this paper we present different matrix factorizations and their use in butterfly algorithms realizing fast Walsh and related transforms.

1 Introduction

The core of the discrete transforms is the matrix by vector multiplication. The specification of the particular transformation is what type of matrices is used. For the purposes of a fast algorithm the main transform matrix has been represented as a product of sparse matrices. Each row of the matrices we consider consists of zeros except one, two or a few elements that are usually 1's or -1's. This leads to suitable algorithms that are much more effective than the usual matrix by vector multiplication.

Walsh (Walsh-Hadamard) transform is a square-wave analog of Fourier transform, based on Walsh functions [10]. Hadamard matrices of Sylvester type can be obtained from Walsh transform matrices by interchanging of certain rows [3]. The Hadamard matrix H_n are the n-th Kroneker power of the matrix $H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. This allows us to represent the matrices H_n as products of rare matrices [2] which leads to a fast algorithm.

Nowadays Walsh and related transforms have different applications [1], including the signal processing, communications and cryptography. A special application is implementation for digital devices that relates to spectral logic [6]. In coding theory, Walsh transform can be used for computing the weight spectrum of a linear code [5]. Walsh spectrum is important in the study of the cryptographic properties of boolean functions [4, 8, 9].

The aim of this paper is to show some methods for factorization of transform matrices when they are represented as a Kroneker product. Each factorization corresponds to an algorithm for a fast computation of the considered transform. Exploring the mathematical foundations of these algorithms is very important for their effective software implementation. Most of these algorithms are suitable for parallel implementation.

[1]This research was partially supported by Grant DN 02/2/13.12.2016 of the Bulgarian National Science Fund

2 Walsh transform: definition and fast algorithm

Let n be a positive integer and $\mathbb{F}_2 = \{0, 1\}$ be the Galois field of two elements. There is one-to-one correspondence between the nonnegative integers $u < 2^n$ and the binary vectors of length n defined by $u \mapsto \bar{u} = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{F}_2^n$, where $u = u_0 2^{n-1} + u_1 2^{n-2} + \cdots + u_{n-1}$ is the binary representation of u.

Let $h : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function of n variables. Truth Table TT_h of the function h is a 2^n-dimensional column vector whose $(i+1)$-th coordinate is $h(\bar{i})$, $i = 0, \ldots, 2^n - 1$.

Walsh transform of the function h is defined as follows

$$W_h(\omega) = \sum_{u=0}^{2^n-1} h(\bar{u})(-1)^{\langle \bar{u}, \bar{\omega} \rangle} \tag{1}$$

where $\langle x, y \rangle$ means the inner product of the vectors x and y. We will denote by W_h the column vector with coordinates $W_h(\omega)$, $\omega = 0, 1, \ldots, 2^n - 1$.

We use a special type of Hadamard matrices defined inductively as follows

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad H_n = \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}, \; n > 1. \tag{2}$$

Theorem 1. *If h is a Boolean function of n variables then $W_h = H_n \cdot TT_h$.*

Thus the computation of Walsh transform is reduced to the matrix by vector multiplication. The elements of the matrix H_n are only 1's and -1's. So when we multiply row of H_n by vector we have to do only additions and subtractions.

There is a fast algorithm to multiply H_n by a vector $\boldsymbol{v}$. First, we partition $\boldsymbol{v}$ into vectors of two coordinates. For each such vector we have

$$H_1 \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_1 + v_2 \\ v_1 - v_2 \end{pmatrix}.$$

Second, we partition $\boldsymbol{v}$ into vectors of four coordinates and multiply H_2 by the new parts using the previous results. Thus in the k-th step we have to calculate the multiplication H_k by a part $\boldsymbol{v'}$ of $\boldsymbol{v}$ where $\boldsymbol{v'}$ has 2^k coordinates. Let in the previous step $\boldsymbol{v'}$ has been partitioned into two 2^{k-1}-dimensional vectors $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$ and $H_{k-1} \cdot \boldsymbol{v}^{(i)}$, $i = 1, 2$, has been computed. Then

$$H_k \cdot \boldsymbol{v'} = \begin{pmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{v}^{(1)} \\ \boldsymbol{v}^{(2)} \end{pmatrix} = \begin{pmatrix} H_{k-1} \cdot \boldsymbol{v}^{(1)} + H_{k-1} \cdot \boldsymbol{v}^{(2)} \\ H_{k-1} \cdot \boldsymbol{v}^{(1)} - H_{k-1} \cdot \boldsymbol{v}^{(2)} \end{pmatrix}.$$

Thereby we may consequently calculate $H_n \cdot \boldsymbol{v}$.

In this way we do n steps and in every step we have one addition or subtraction by row. So we have to do $n.2^n$ arithmetic operations instead of $2^n.2^n$ operations for the usual multiplication.

3 Math explanation: factorization of Kroneker power

By the notion of Kroneker power we can explain the algorithm described in the previous section. Kroneker product of the matrices $A = (a_{ij})_{m\times n}$ and $B = (b_{ij})_{s\times t}$ is the $(ms \times nt)$-matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \dots & & & \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix}.$$

We give some elementary properties of Kroneker product:

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \tag{3}$$

$$(A + B) \otimes (C + D) = A \otimes C + A \otimes D + B \otimes C + B \otimes D \tag{4}$$

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD) \tag{5}$$

Kroneker product is not commutative.

The n-th Kroneker power of a square matrix M is defined as follows

$$M^{[2]} = M \otimes M, \quad M^{[n+1]} = M \otimes M^{[n]}, n > 1.$$

If M is a square matrix of size t then $M^{[n]}$ has size t^n.

Theorem 2 (Good[2]). *Let M be a square matrix of size t and n be a positive integer. Then*

$$M^{[n]} = B_1 \cdot B_2 \cdots B_n, \tag{6}$$

where $B_i = I_{t^{i-1}} \otimes M \otimes I_{t^{n-i}}$, $1 \le i \le n$, I_k is the identity matrix of size k.

Proof by induction is based on equation $M^{[n]} = (M \otimes I_{t^{n-1}}) \cdot (I_t \otimes M^{[n-1]})$.

Example 1. Let $M = H_1$. Comparing with (2) we see that $H_1^{[n]} = H_n$. For $n = 3$ we have $H_3 = (H_1 \otimes I_4) \cdot (I_2 \otimes H_1 \otimes I_2) \cdot (I_4 \otimes H_1)$.

$$H_3 = \begin{pmatrix} I_4 & I_4 \\ I_4 & -I_4 \end{pmatrix} \cdot \begin{pmatrix} I_2 & I_2 & & \\ I_2 & -I_2 & & \\ & & I_2 & I_2 \\ & & I_2 & -I_2 \end{pmatrix} \cdot \begin{pmatrix} H_1 & & & \\ & H_1 & & \\ & & H_1 & \\ & & & H_1 \end{pmatrix}$$

Each row in the above multipliers consists of only two nonzero elements, namely 1 or -1. So the row by vector multiplication costs one arithmetic operation (addition or subtraction). Multiplying by B_i from (6) corresponds to an iterative step of the fast algorihtm described in section 2.

[2]Originally Good [2] formulate this theorem for different Kroneker multipliers.

Example 2. Reed-Muller (or Möbius) transform [4] is based on the matrices

$$R_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad R_n = R_1 \otimes R_{n-1},\, n > 1. \tag{7}$$

Applying Theorem 2 we obtain a corresponding factorization. For $n = 3$ we have $R_3 = (R_1 \otimes I_4) \cdot (I_2 \otimes R_1 \otimes I_2) \cdot (I_4 \otimes R_1)$.

$$R_3 = \begin{pmatrix} I_4 & \\ I_4 & I_4 \end{pmatrix} \cdot \begin{pmatrix} I_2 & & & \\ I_2 & I_2 & & \\ & & I_2 & \\ & & I_2 & I_2 \end{pmatrix} \cdot \begin{pmatrix} R_1 & & & \\ & R_1 & & \\ & & R_1 & \\ & & & R_1 \end{pmatrix}$$

In every row of the multipliers there are one or two nonzero elements, namely 1's. The complexity of the fast algorithm to multiply by R_n is $n.2^{n-1}$ additions.

In the above examples we see that different factorizations lead to different algorithms for multiplying by transform matrices. In fact, an algorithm depends on the factorization. In the next theorem we see that multipliers in (6) commute themselves. Lechner [7] applied (6) to Walsh transform, but in reverse order of multipliers, and mentioned that factors are commutative.

Theorem 3. *Factors in* (6) *are commutative, so their order does not matter.*

Proof. Applying properties of Kroneker product we have

$$\begin{aligned} B_i \cdot B_j &= (I_{t^{i-1}} \otimes M \otimes I_{t^{n-i}}) \cdot (I_{t^{j-1}} \otimes M \otimes I_{t^{n-j}}) \\ &= I_{t^{i-1}} \otimes ((M \otimes I_{t^{j-i}}) \cdot (I_{t^{j-i}} \otimes M)) \otimes I_{t^{n-j}} \\ &= I_{t^{i-1}} \otimes (M \cdot I_t) \otimes (I_{t^{j-i}} \cdot (I_{t^{j-i-1}} \otimes M)) \otimes I_{t^{n-j}} \\ &= I_{t^{i-1}} \otimes M \otimes I_{t^{j-i-1}} \otimes M \otimes I_{t^{n-j}} \\ &= I_{t^{i-1}} \otimes (I_t \cdot M) \otimes ((I_{t^{j-i-1}} \otimes M) \cdot I_{t^{j-i}}) \otimes I_{t^{n-j}} \\ &= I_{t^{i-1}} \otimes ((I_{t^{j-i}} \otimes M) \cdot (M \otimes I_{t^{j-i}})) \otimes I_{t^{n-j}} = B_j \cdot B_i. \end{aligned}$$

□

Thus we can interchange the iterative steps in a fast algorithm.

4 Other factorizations

We present another factorization proposed by Good [2].

Let t, n, r and s be positive integers, $r, s \leq t^n$,

$$r = r_0 t^{n-1} + r_1 t^{n-2} + \cdots + r_{n-1} + 1 \quad \text{and} \quad s = s_0 t^{n-1} + s_1 t^{n-2} + \cdots + s_{n-1} + 1$$

where $0 \leq r_i, s_i < t, 0 \leq i < n$. We denote $\boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})$ and $\boldsymbol{s} = (s_0, s_1, \ldots, s_{n-1})$.

Theorem 4 (Good)**.** *Let M be a square matrix of size t and A be a $(t^n \times t^n)$-matrix whose elements are*

$$A_{rs} = M_{r_0+1,s_{n-1}+1}\delta_{r_1s_0}\delta_{r_2s_1}\dots\delta_{r_{n-1}s_{n-2}}, \tag{8}$$

where $\delta_{ij} = 1$ when $i = j$ and $\delta_{ij} = 0$ otherwise. Thus $M^{[n]} = A^n$.

The implementation of the above theorem gives us a fast algorithms with uniform iterative steps.

Example 3. For $M = H_1$, $n = 3$ the elements of A are listed below

$r \setminus s$	000	001	010	011	100	101	110	111	r
000	1	1	0	0	0	0	0	0	1
001	0	0	1	1	0	0	0	0	2
010	0	0	0	0	1	1	0	0	3
011	0	0	0	0	0	0	1	1	4
100	1	−1	0	0	0	0	0	0	5
101	0	0	1	−1	0	0	0	0	6
110	0	0	0	0	1	−1	0	0	7
111	0	0	0	0	0	0	1	−1	8
s	1	2	3	4	5	6	7	8	

Thus $M^{[3]} = A^3$. Since H_1 is a symmetric matrix, then $M^{[3]} = (A^T)^3$.

We may combine factorizations from theorems 2 and 4.

Example 4. Let A be the matrix from Theorem 4 for $M = H_1$ and $n = 3$. Combining this with Theorem 2 we have

$$H_4 = (H_1 \otimes I_8) \cdot (I_2 \otimes H_3) = (H_1 \otimes I_8) \cdot (I_2 \otimes A^3) = (H_1 \otimes I_8) \cdot (I_2 \otimes A)^3.$$

Another way to produce a new factorization (respectively a fast algorithm) is by inserting permutation matrices as multipliers. This method is based on the following properties: (1) each permutation matrix is invertible and it is equal to the transpose of its inverse matrix; (2) multiplying any matrix to the left with permutation matrix leads to a permutation of rows; (3) multiplying any matrix to the right with a permutation matrix leads to a permutation of columns. So in any factorization we may permute rows (or columns) of a given multiplier and this leads to a suitable permutation of columns (or rows) of a neighboring multiplier or the output vector.

Example 5. Let A be the matrix from Theorem 4 for a fixed M of size t. By a suitable permutation of rows of A one can obtain the matrix $I_{t^{n-1}} \otimes M$. By a suitable permutation of columns of A one can obtain the matrix $M \otimes I_{t^{n-1}}$.

5 Conclusion

The study of the different factorizations of the Hadamard matrices H_n can be helpful for the software implementations of algorithms for computing the Walsh spectrum of a Boolean function. Calculation process has one and the same complexity in sense of the number of arithmetic operations. But there is a possibility that different algorithms work with different speed depending on the used hardware or software platforms. The present paper gives a guidance how to make a specific fast algorithm. Described methods can be applied to any discrete transform when the transform matrices are presented as Kroneker powers of a given matrix.

References

[1] D. F. Elliott and K. R. Rao, *Fast Transforms: Algorithms, Analises, Applications,* Academic Press, Orlando, Florida, 1982.

[2] I. J. Good, The Interaction Algorithm and Practical Fourier Analysis, *J. of the Royal Stat. Soc.,* Ser. B., **20**, 361-372, 1958; Addendum, **22**, 372-375, 1960.

[3] K. W. Henderson, Some Notes on the Walsh Functions, *IEEE Transactions on Electronic Computers*, **EC-13**, 50-52, 1964.

[4] A. Joux, Algorithmic cryptanalysis, Chapman & Hall/CRC, Boca Raton, 2009.

[5] M. G. Karpovsky, On the Weight Distribution of Binary Linear Codes, *IEEE Trans. Inform. Theory,* **IT-25**, 105-109, 1979.

[6] M. G. Karpovsky, R. S. Stanković and J. T. Astola, *Spectral Logic and its Applications for the Design of Digital Devices,* Wiley-Interscience, Hoboken, New Jersey, 2008.

[7] R. J. Lechner, Comment on "Computation of the Fast Walsh-Fourier Transform", *IEEE Trans. Comp.,* **C-19**, 174, 1970.

[8] S. Maitra and P. Sarkar, Cryptographically Significant Boolean Functions with Five Valued Walsh Spectra, *Theoretical Comp. Sci.,* **276**, 133-146, 2002.

[9] E. Uyan, Ç. Çalik and A. Doğanaksoy, Counting Boolean Functions with Specified Values in Their Walsh Spectrum, *J. Comp. and Appl. Math.,* **259**, 522-528, 2014.

[10] J. L. Walsh, A Closed Set of Normal Orthogonal Functions, *American J. of Math.,* **45**, 5-24, 1923.