Computing Walsh Transform from the Algebraic Normal Form of a Boolean Function

Kishan Chand Gupta and Palash Sarkar
Cryptology Research Group
Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road
Kolkata 700108, India
e-mail:{kishan_t,palash}@isical.ac.in

Abstract

We study the relationship between the Walsh transform and the algebraic normal form of a Boolean function. In the first part of the paper, we carry out a combinatorial analysis to obtain a formula for the Walsh transform at a certain point in terms of parameters derived from the algebraic normal form. The second part of the paper is devoted to simplify this formula and develop an algorithm to evaluate it. Our algorithm can be applied in situations where it is practically impossible to use the fast Walsh transform algorithm. Experimental results show that under certain conditions it is possible to execute our algorithm to evaluate the Walsh transform (at a small set of points) of functions on a few scores of variables having a few hundred terms in the algebraic normal form.

Keywords : Boolean function, Algebraic Normal Form, Walsh Transform.

1 Introduction

Boolean functions are used for a wide variety of applications in engineering and computer science. An m-variable Boolean function $g(x)$ is a map $g : \{0,1\}^m \rightarrow \{0,1\}$. One of the most useful tools for the study of Boolean functions is the Walsh transform, which is essentially the Fourier transform applied to the function $(-1)^{g(x)}$. The Walsh transform measures the correlations between an m-variable Boolean function and all the m-variable linear functions. These correlations uniquely determine the function and hence it is possible to work entirely with the Walsh transform. In fact, many properties of Boolean functions are most easily stated in terms of Walsh transform.

For practical applications it is of utmost importance to be able to compute the Walsh transform of a Boolean function. It turns out that there is an excellent algorithm to do so, namely, the fast Walsh transform. For an m-variable function the fast Walsh transform takes time $O(m2^m)$ and hence can be used for functions of around 30 variables. The fast Walsh transform is computed from the description of the function itself. More precisely, the fast Walsh transform takes as input the bit string $f(\sigma_0)\ldots f(\sigma_{2^m-1})$ of length 2^m, where for $0 \leq i \leq 2^m - 1$, σ_i is the m-bit binary representation of i.

There is another way to uniquely represent a Boolean function, namely by its algebraic normal form, which expresses a Boolean function as a multivariate polynomial over F_2, the finite field of two elements. The number of nonzero terms in the polynomial can be 2^m in the worst case. However, many interesting classes of Boolean functions do have compact algebraic normal form representation. (For example, an $m = 2k$-variable bent function can have as low as k many terms in its ANF.)

In this paper, we study the relationship between the algebraic normal form and the Walsh transform of a Boolean function. We obtain a formula for the Walsh transform of a Boolean function at a certain point $v \in F_2^m$ in terms of certain parameters derived from the algebraic normal form. We present an efficient algorithm to evaluate the formula and hence compute the Walsh transform at v.

Our algorithm can be used to compute the Walsh transform in cases where it is not possible to use the fast Walsh transform. For example, it is possible in certain cases to run our algorithm for 50 to 100 variable functions having a few hundred terms in its algebraic normal form. For such functions it is possible to compute the Walsh transform for a small set of points v. Note that it is practically impossible to compute the Walsh transform of an m-variable function at all points in F_2^m if m is around 50 or more. Our algorithm provides a method to probe the spectral domain of large variable functions. This will provide some useful information about the function such as the size of its support and an estimate of its nonlinearity. Note that for small number of variables, the fast Walsh transform is faster than our algorithm. Hence we do not provide a substitute to the fast Walsh transform; rather we provide a tool to analyse a Boolean function in situations where the fast Walsh transform cannot be used.

2 Preliminaries

Let $F_2 = GF(2)$. An m-variable Boolean function $g(x)$ is a map $g : F_2^m \to F_2$. We consider the domain of an m-variable Boolean function to be the vector space $(F_2^m, \oplus)$ over F_2, where $\oplus$ is used to denote the addition operator over both F_2 and the vector space F_2^m. The symbol $+$ is used to denote addition over integers. The inner product of two vectors $u, v \in F_2^m$ will be denoted by $\langle u, v \rangle$. The weight of an m-bit vector u is the number of ones in u and will be denoted by $wt(u)$. The *support* of a Boolean function g is denoted by $Sup(g)$ and is defined to be $Sup(g) = \{x : g(x) = 1\}$. The weight of g is denoted by $wt(g)$ and is defined to be $wt(g) = |Sup(g)|$. The *Walsh Transform* of a Boolean function $g(x)$ is an integer valued function $W_g : \{0,1\}^m \to [-2^m, 2^m]$ defined by (see [2, page 414])

$$W_g(u) = \sum_{w \in F_2^m} (-1)^{g(w) \oplus \langle u, w \rangle}. \tag{1}$$

Let H_m be the $2^m \times 2^m$ Hadamard matrix (see [2, page 44]) whose rows and columns are indexed by the elements of F_2^m such that $H_m(u, v) = (-1)^{\langle u, v \rangle}$. Then we can write

$$[(-1)^{g(0)}, \ldots, (-1)^{g(2^m-1)}]H_m \quad = \quad [W_g(0), \ldots, W_g(2^m - 1)]. \tag{2}$$

An m-variable Boolean function $g(x_1, x_2, \cdots, x_m)$ can be uniquely written as

$$g(x_1, x_2, \ldots, x_m) = \bigoplus_{(\alpha_1, \alpha_2, \cdots, \alpha_m) \in F_2^m} A_g(\alpha_1, \alpha_2, \cdots, \alpha_m) x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_m^{\alpha_m} \tag{3}$$

where $A_g(x_1, x_2, \cdots, x_m)$ is a Boolean function. This representation is called the *algebraic normal form (ANF)* of g. We will call the Boolean function A_g to be the ANF of g.

3 Walsh Transform

In this section, we express the Walsh transform of a Boolean functions in terms of certain parameters derived from its ANF. The approach that we take is the following. Equation (2) expresses the relation between $(-1)^{g(x)}$ and $W_g(u)$. Our first task is to obtain a formula for $(-1)^{g(x)}$ in terms of the ANF of $g(x)$. Then using Equation (2) we obtain the desired relationship between the Walsh transform and the ANF.

In obtaining a formula for $(-1)^{g(x)}$ in terms of the ANF, we first tackle the special case when $g(x) = g(x_1, \ldots, x_m) = x_1 \ldots x_m$. The result of this case is used to analyse the general case.

3.1 Case $\mathbf{g(x_1, x_2, \cdots, x_m) = x_1 x_2 \cdots x_m}$

For $x = (x_1, \cdots, x_m) \in \{0,1\}^m$, define

$$N_r^{(m)}(x) \triangleq N_r^{(m)}(x_1, \cdots, x_m) \triangleq \sum_{1 \le i_1 < i_2 < \cdots < i_r \le m} (-1)^{x_{i_1} \oplus \cdots \oplus x_{i_r}}.$$

Lemma 3.1 *Let $x = (x_1, \ldots, x_m) \in F_2^m$ be such that $wt(x) = k$. Then*

$$N_r^{(m)}(x) = \sum_{j=0}^{r} (-1)^j \binom{k}{j} \binom{m-k}{r-j}.$$

Consequently, $N_r^{(m)}(x)$ is the Krawtchouk polynomial $p_r(k,m)$ [2, page 130].

Proof : In the vector $(x_1, \cdots, x_m)$, k of the x_i's are 1 and $(m-k)$ of the x_i's are 0. Hence the number of terms of the type $(x_{i_1} \oplus \cdots \oplus x_{i_r})$ with j number of 1's $(0 \le j \le r)$ and $(r-j)$ number of 0's is equal to $\binom{k}{j}\binom{m-k}{r-j}$. Since j of the x_i's are ones, we have, $(-1)^{x_{i_1} \oplus \cdots \oplus x_{i_r}} = (-1)^j$. Hence we get $N_r^{(m)}(x_1, \cdots, x_m) = \sum_{j=0}^{r} (-1)^j \binom{k}{j}\binom{m-k}{r-j}$ which is the Krawtchouk polynomial $p_r(k,m)$ [2, page 130]. ■

Corollary 3.1 *1. $\sum_{r=0}^{m} (-1)^{r-1} N_r^{(m)}(x_1, \cdots, x_m) = 0$ for any vector $x = (x_1, \cdots, x_m) \ne (1, \cdots, 1)$.*

2. $\sum_{r=0}^{m} (-1)^{r-1} N_r^{(m)}(1, \cdots, 1) = -2^m$.

Proof : From [2, Equation (16), page 130] we have $\sum_{r=0}^{m} (-1)^{r-1} p_r(k,m) = 0$ for $0 \le k < m$ and $\sum_{r=0}^{m} (-1)^{r-1} p_r(k,m) = -2^m$ for $k = m$. Hence using Lemma 3.1 the result follows. ■

Theorem 3.1 *Let $g(x) = x_1 x_2 \cdots x_m$. Then*

$$(-1)^{g(x)} = \frac{1}{2^{m-1}} \left(2^{m-1} + \sum_{r=0}^{m} (-1)^{r-1} N_r^{(m)}(x) \right).$$

Proof : *Case 1*: $x \in \{0,1\}^m$ such that $x = (1, \cdots, 1)$. Then L.H.S $= (-1)^1 = -1$. By Corollary 3.1(2) we have $\sum_{r=0}^{m} (-1)^{r-1} N_r^{(m)}(x) = \sum_{r=0}^{m} (-1)^{r-1} N_r^{(m)}(1, \cdots, 1) = -2^m$. Hence R.H.S $= \frac{1}{2^{m-1}}(2^{m-1} - 2^m) = -1$. *Case 2*: $x \in \{0,1\}^m$ such that $x \ne (1, \cdots, 1)$. Then L.H.S $= (-1)^0 = 1$. By Corollary 3.1(1) we have $\sum_{r=0}^{m} (-1)^{r-1} N_r^{(m)}(x) = 0$. Hence R.H.S $= \frac{1}{2^{m-1}}(2^{m-1} - 0) = 1$. ■

3.2 Arbitrary g

Let g be a Boolean function and A_g be its ANF. For $\alpha, x \in F_2^m$ and $r \in [m] = \{1, \ldots, m\}$ define

$$N_r^{(m)}(\alpha, x) \triangleq \sum_{1 \le i_1 < i_2 < \cdots < i_r \le m} (-1)^{{x_{i_1}}^{\alpha_{i_1}} \oplus \cdots \oplus {x_{i_r}}^{\alpha_{i_r}}}.$$

For $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_m) \in F_2^m$ and $x = (x_1, \cdots, x_m)$, define $x^\alpha \triangleq x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_m^{\alpha_m}$. Note that if $\alpha_i = 1$, then $x_i^{\alpha_i} = x_i$ else $x_i^{\alpha_i} = 1$.

Proposition 3.1

$$(-1)^{x^\alpha} = \frac{1}{2^{m-1}}\left(2^{m-1} + \sum_{r=0}^{m}(-1)^{r-1}N_r^{(m)}(\alpha, x)\right)$$

Proof : Let $\mathbf{1} = (1, \dots, 1)$ and note that $N_r^{(m)}(\mathbf{1}, x) = N_r^{(m)}(x)$. We define $y = (y_1, \cdots, y_m)$ as follows. For $1 \le i \le m$, if $\alpha_i = 1$ then $y_i = x_i$ else $y_i = 1$. Now it is easy to check the following

1. $x_i^{\alpha_i} = y_i$ and hence $(-1)^{x^\alpha} = (-1)^y$.
2. $N_r^{(m)}(\alpha, x) = N_r^{(m)}(y)$.

From this the result follows. ■

The next result expresses $(-1)^{g(x)}$ in terms of A_g.

Proposition 3.2

$$(-1)^{g(x)} = \left(\frac{1}{2^{m-1}}\right)^{wt(A_g)} \prod_{\alpha : A_g(\alpha)=1}\left(2^{m-1} + \sum_{r=0}^{m}(-1)^{r-1}N_r^{(m)}(\alpha, x)\right)$$

Proof :

$$(-1)^{g(x)} = (-1)^{\bigoplus_{\alpha\in F_2^m} A_g(\alpha)x^\alpha} = \prod_{\alpha\in F_2^m}(-1)^{A_g(\alpha)x^\alpha} = \prod_{\alpha\in F_2^m}\left((-1)^{x^\alpha}\right)^{A_g(\alpha)} = \prod_{\alpha : A_g(\alpha)=1}(-1)^{x^\alpha}$$

$$\begin{aligned} &= \prod_{\alpha : A_g(\alpha)=1}\frac{1}{2^{m-1}}\left(2^{m-1} + \sum_{r=0}^{m}(-1)^{r-1}N_r^{(m)}(\alpha, x)\right) \\ &= \left(\frac{1}{2^{m-1}}\right)^{wt(A_g)} \prod_{\alpha : A_g(\alpha)=1}\left(2^{m-1} + \sum_{r=0}^{m}(-1)^{r-1}N_r^{(m)}(\alpha, x)\right) \end{aligned}$$

■

For $\alpha = (\alpha_1, \cdots, \alpha_m)$ and $u = (u_1, \cdots, u_m)$ we define $u \le \alpha$ if $u_i \le \alpha_i$ for $1 \le i \le m$. *In the rest of the paper we will denote* ($wt(u)$ mod 2) *by* $wt_2(u)$. In the next two results we present a two step simplification of the sum $\sum_{r=0}^{m}(-1)^{r-1}N_r^{(m)}(\alpha, x)$ which occurs in Proposition 3.2.

Proposition 3.3

$$N_r^{(m)}(\alpha, x) = (-1)^r \sum_{\{u\in F_2^m : u\le\alpha, wt(u)\le r\}} \binom{m-wt(\alpha)}{r-wt(u)}(-1)^{\langle u,x\rangle \oplus wt_2(u)}$$

Proof : Note ${x_{i_1}}^{\alpha_{i_1}} = x_{i_1}$ if $\alpha_{i_1} = 1$ and ${x_{i_1}}^{\alpha_{i_1}} = 1$ if $\alpha_{i_1} = 0$. So ${x_{i_1}}^{\alpha_{i_1}} = 1 \oplus (1 \oplus x_{i_1})\alpha_{i_1} = 1 \oplus (\overline{x}_{i_1})\alpha_{i_1}$ and hence ${x_{i_1}}^{\alpha_{i_1}} \oplus \cdots \oplus {x_{i_r}}^{\alpha_{i_r}} = (r \bmod 2) \oplus \langle(\overline{x}_{i_1}, \cdots, \overline{x}_{i_r}), (\alpha_{i_1}, \cdots, \alpha_{i_r})\rangle$. We have

$$\begin{aligned} N_r^{(m)}(\alpha, x) &= \sum_{1\le i_1<i_2<\cdots<i_r\le m}(-1)^{{x_{i_1}}^{\alpha_{i_1}}\oplus\cdots\oplus {x_{i_r}}^{\alpha_{i_r}}} \\ &= \sum_{1\le i_1<i_2<\cdots<i_r\le m}(-1)^{(r \bmod 2)\oplus\langle(\overline{x}_{i_1},\cdots,\overline{x}_{i_r}),(\alpha_{i_1},\cdots,\alpha_{i_r})\rangle}. \\ &= (-1)^r \sum_{1\le i_1<i_2<\cdots<i_r\le m}(-1)^{\langle(\overline{x}_{i_1},\cdots,\overline{x}_{i_r}),(\alpha_{i_1},\cdots,\alpha_{i_r})\rangle}. \end{aligned}$$

Given $\alpha = (\alpha_1, \dots, \alpha_m)$ and $1 \le i_1 < i_2 < \dots < i_r \le m$, define an m-bit vector u in the following manner. For $1 \le j \le m$, if $j \in \{i_1, \dots, i_r\}$, then $u_j = \alpha_j$ else $u_j = 0$. We say that $\{i_1, \dots, i_r\}$ *produces* u from α. If $\{i_1, \dots, i_r\}$ produces u from α, then it is easy to verify the following relations.

1. $\langle(\overline{x}_{i_1},\cdots,\overline{x}_{i_r}),(\alpha_{i_1},\cdots,\alpha_{i_r})\rangle = \langle(\overline{x}_1,\ldots,\overline{x}_m),(u_1,\ldots,u_m)\rangle = \langle(x_1,\ldots,x_m),(u_1,\ldots,u_m)\rangle \oplus wt_2(u)$;

2. $u \leq \alpha$ and $wt(u) \leq r$.

It is possible that two distinct sets $\{i_1,i_2,\ldots,i_r\}$ and $\{i'_1,i'_2,\ldots,i'_r\}$ produce the same $u \leq \alpha$. This will happen if and only if $wt(\alpha_{i_1},\ldots,\alpha_{i_r}) = wt(\alpha_{i'_1},\ldots,\alpha_{i'_r}) = wt(\alpha_{j_1},\ldots,\alpha_{j_s})$, where $\{j_1,\ldots,j_s\} = \{i_1\ldots,i_r\}\cap\{i'_1,\ldots,i'_r\}$. We now claim that the number of distinct sets $\{i_1,i_2,\ldots,i_r\}$ which produce the same u is $\binom{m-k}{r-l}$, where $k = wt(\alpha)$ and $l = wt(u)$. To see this fix an $u \leq \alpha$ with $wt(u) = l \leq r$. Let $j_1,\ldots,j_l$ be such that $u_{j_1} = \ldots = u_{j_l} = 1$ and $u_j = 0$ for $j \notin \{j_1,\ldots,j_l\}$. If $\{i_1,\ldots,i_r\}$ produces u, we must have $\{j_1,\ldots,j_l\} \subseteq \{i_1,\ldots,i_r\}$ and $\alpha_j = 0$ for $j \in S = \{i_1,\ldots,i_r\}\setminus\{j_1,\ldots,j_l\}$

Thus the number of sets $\{i_1,\ldots,i_r\}$ which produce u is the number of ways we can choose the set S of $r-l$ elements such that $\alpha_j = 0$ for $j \in S$. Since $wt(\alpha) = k$, the number of $j \in \{1,\ldots,m\}$ such that $\alpha_j = 0$ is $m-k$. Since S has $r-l$ elements, the number of possible sets S is $\binom{m-k}{r-l}$, which proves our claim. Hence we can write

$$N_r^{(m)}(\alpha,x) \quad = \quad (-1)^r \sum_{\{u\in F_2^m : u\leq\alpha, wt(u)\leq r\}} \binom{m-k}{r-l} (-1)^{\langle u,x\rangle \oplus wt_2(u)}$$

This completes the proof. ■

Proposition 3.4

$$\sum_{r=0}^{m}(-1)^{r-1}N_r^{(m)}(\alpha,x) = (-1)\times \sum_{\{u\in F_2^m : u\leq\alpha\}} 2^{m-wt(\alpha)}(-1)^{\langle u,x\rangle\oplus wt_2(u)}$$

Proof : By Proposition 3.3

$$\begin{aligned}
\sum_{r=0}^{m}(-1)^{r-1}N_r^{(m)}(\alpha,x) &= (-1)^{2r-1}\sum_{r=0}^{m}\sum_{\{u\in F_2^m : u\leq\alpha, wt(u)\leq r\}}\binom{m-wt(\alpha)}{r-wt(u)}(-1)^{\langle u,x\rangle\oplus wt_2(u)} \\
&= (-1)\sum_{\{u\in F_2^m : u\leq\alpha\}}\sum_{r=wt(u)}^{m}\binom{m-wt(\alpha)}{r-wt(u)}(-1)^{\langle u,x\rangle\oplus wt_2(u)} \\
&= -\sum_{\{u\in F_2^m : u\leq\alpha\}} 2^{m-wt(\alpha)}(-1)^{\langle u,x\rangle\oplus wt_2(u)}.
\end{aligned}$$

This completes the proof. ■

Let $Sup(A_g) = \{\alpha^{(1)},\cdots,\alpha^{(p)}\}$ and $k_i = wt(\alpha^{(i)})$. Define

$$V(Sup(A_g)) \triangleq V(\{(\alpha^{(1)}),\cdots,(\alpha^{(p)})\}) \triangleq \{u^{(i_1)}\oplus\cdots\oplus u^{(i_r)} : u^{(i_j)} \leq \alpha^{(i_j)}, \{i_1,\ldots,i_r\}\subseteq[p], 1\leq j\leq r\}.$$

Note that $V(Sup(A_g))$ is a subspace of $(F_2^m,\oplus)$. Further, it is easy to verify that $V(Sup(A_g)) = F_2^m$ if and only if g is non-degenerate. Suppose $v = u^{(i_1)}\oplus\cdots\oplus u^{(i_r)}$ where $u^{(i_j)} \leq \alpha^{(i_j)}$ then we say that $R = \{(u^{(i_1)},\alpha^{(i_1)}),\cdots,(u^{(i_r)},\alpha^{(i_r)})\}$ is *a representation of* v. For $v \in F_2^m$, define

$$\mathcal{S}(v) \triangleq \text{set of all representations of } v \text{ in the vector space } V(Sup(A_g)).$$

Note:

1. $\mathcal{S}(v) = \phi$ if and only if $v \notin V(Sup(A_g))$.

2. If $v = (0, \cdots, 0)$, then $\mathcal{S}(v) \neq \phi$.

Let $v \in F_2^m$ and $R = \{(u^{(i_1)}, \alpha^{(i_1)}), \cdots, (u^{(i_r)}, \alpha^{(i_r)})\} \in \mathcal{S}(v)$. Define

$$\left.\begin{array}{lcl} C(R) & \triangleq & \frac{(-1)^r}{2^{(k_{i_1}-1)+\cdots+(k_{i_r}-1)}}. \\ C_v & \triangleq & (-1)^{wt_2(v)} \sum_{R \in \mathcal{S}(v)} C(R). \end{array}\right\} \tag{4}$$

Note that if $\mathcal{S}(v) = \phi$, then $\sum_{R \in \mathcal{S}(v)} C(R) = 0$ and hence, if $v \notin V(Sup(A_g))$ then $C_v = 0$.

The parameters C_v for $v \in F_2^m$ are derived entirely from the ANF of g. Our next result expresses $(-1)^{g(x)}$ in terms of C_v.

Theorem 3.2

$$(-1)^{g(x)} = 1 + \sum_{v \in F_2^m} C_v (-1)^{\langle v,x \rangle}$$

where C_v is as defined by Equation 4.

Proof : As before let $Sup(A_g) = \{\alpha^{(1)}, \ldots, \alpha^{(p)}\}$ where $wt(\alpha^{(i)}) = k_i$. Using Proposition 3.2 and Proposition 3.4 we can write.

$$(-1)^{g(x)} = \prod_{\alpha : A_g(\alpha)=1} \left(1 - \frac{1}{2^{m-1}} \sum_{\{u \in F_2^m : u \leq \alpha\}} 2^{m-wt(\alpha)} (-1)^{\langle u,x \rangle \oplus wt_2(u)}\right)$$

The first term in the expansion of the above expression is clearly 1. For $1 \leq r \leq p = wt(A_g)$, the general term is of the form

$$\left(\frac{-1}{2^{m-1}}\right)^r \sum_{u^{(i_1)} \leq \alpha^{(i_1)}} \cdots \sum_{u^{(i_r)} \leq \alpha^{(i_r)}} 2^{(m-k_{i_1})+\cdots+(m-k_{i_r})} (-1)^{\langle u^{(i_1)} \oplus \cdots \oplus u^{(i_r)}, x \rangle \oplus wt_2(u^{(i_1)} \oplus \cdots \oplus u^{(i_r)})}$$

Let $v = u^{(i_1)} \oplus \cdots \oplus u^{(i_r)}$. Then $R = \{(u^{(i_1)}, \alpha^{(i_1)}), \cdots, (u^{(i_r)}, \alpha^{(i_r)})\}$ is a representation of v. Therefore the general term is of the form

$$\sum_{v \in F_2^m} \sum_{R \in \mathcal{S}(v)} C(R)(-1)^{\langle v,x \rangle \oplus wt_2(v)} = \sum_{v \in F_2^m} C_v (-1)^{\langle v,x \rangle}.$$

This gives us the desired result. ■

Now we are in a position to state the main result of this section which relates $W_g(v)$ to C_v.

Theorem 3.3 *If g is an m-variable Boolean function then*

$$W_g(v) = 2^m (C_v + \delta_v) \tag{5}$$

where $\delta_v = 1$ if $v = 0$ else $\delta_v = 0$.

Proof : From Theorem 3.2 we have

$$(-1)^{g(x)} - 1 = \sum_{v \in F_2^m} C_v (-1)^{\langle v,x \rangle}.$$

So $[(-1)^{g(0)} - 1, \ldots, (-1)^{g(2^m-1)} - 1] = [C_0, \ldots, C_{2^m-1}] H_m$, where H_m is the $(2^m \times 2^m)$ Hadamard matrix (see [2, page 44]). Post multiplying both sides by H_m and noting that $H_m H_m = 2^m I_{2^m}$ we get

$$[(-1)^{g(0)}, \ldots, (-1)^{g(2^m-1)}] H_m - [1, \ldots, 1] H_m = 2^m [C_0, \ldots, C_{2^m-1}].$$

i.e. $[W_g(0), \ldots, W_g(2^m-1)] - [2^m, 0, \ldots, 0] = 2^m [C_0, \ldots, C_{2^m-1}]$. Hence $2^m C_v = W_g(v) - 2^m \delta_v$ ■

4 Simplifying C_v

Theorem 3.3 relates $W_g(v)$ to C_v. Thus to compute $W_g(v)$ it is sufficient to compute C_v. However, the definition of C_v given by 4 is in purely algebraic terms. We need to obtain a formula for C_v which can be computed by an algorithm. In this section we perform this task of simplifying C_v.

We start by defining certain terms. As in Section 3, we assume $Sup(A_g) = \{\alpha^{(1)}, \ldots, \alpha^{(p)}\}$. Given $\emptyset \neq S \subseteq [p] = \{1, 2, \cdots, p\}$ and $v \in F_2^m$, we define $\Delta(S, v)$ as follows. Let $v = (v_1, \cdots, v_m)$ and $S = \{i_1, \cdots, i_r\}$. Write $\alpha^{(i_j)} = (\alpha_{j,1}, \cdots, \alpha_{j,m})$ where $\alpha_{j,k} \in \{0, 1\}$ for $1 \leq j \leq r$, $1 \leq k \leq m$. Let $\phi(S, v)$ be a Boolean formula which is true if and only if there is a $k \in [m]$, such that $\alpha_{j,k} = 0$ for all $j \in [r]$ and $v_k = 1$. Let $\mu(S) = wt(\alpha^{(i_1)} \vee \cdots \vee \alpha^{(i_r)})$. Then we define

$$\left.\begin{array}{rcll} \Delta(S,v) & = & 0 & \text{if } \phi(S,v) = 1 \\ & = & \frac{1}{2^{\mu(S)-|S|}} & \text{otherwise.} \end{array}\right\} \tag{6}$$

Remark: If $\Delta(S, v) > 0$, then the value of $\Delta(S, v)$ is independent of v and depends only on S.

Theorem 4.1

$$C_v = (-1)^{wt_2(v)} \sum_{\emptyset \neq S \subseteq [p]} (-1)^r \Delta(S, v).$$

Proof : Fix a set S, such that, $\emptyset \neq S = \{i_1, \ldots, i_r\} \subseteq [p]$ and a vector $v = (v_1, \ldots, v_m) \in F_2^m$. Define $\Gamma(S) = \frac{2^r}{2^{k_{i_1} + \cdots + k_{i_r}}}$ where $k_{i_j} = wt(\alpha^{(i_j)})$. As before, let $\alpha^{(i_j)} = (\alpha_{j,1}, \cdots, \alpha_{j,m})$, $1 \leq j \leq r$ and define $\lambda_k = \sum_{j=1}^r \alpha_{j,k}$. For $1 \leq k \leq m$, define

$$\left.\begin{array}{rcll} b_k & = & 2^{\lambda_k - 1} & \text{if } \lambda_k \neq 0 \\ & = & 1 & \text{if } \lambda_k = 0, v_k = 0 \\ & = & 0 & \text{if } \lambda_k = 0, v_k = 1 \end{array}\right\} \tag{7}$$

Define $n(S, v) = b_1 \cdots b_m$.

Claim 1: $\Delta(S, v) = n(S, v)\Gamma(S)$

Proof of Claim 1: There are two cases to consider.

Case $\phi(S, v) = 1$: In this case, $\Delta(S, v) = 0$ by Equation 6. Also $\phi(S, v) = 1$ implies that there is a $k \in [m]$ such that $\alpha_{j,k} = 0$ for all $j \in [r]$ and $v_k = 1$. This implies that $\lambda_k = 0$ and $v_k = 1$. Hence $b_k = 0$ and so $n(S, v) = 0$. Thus in this case we have $\Delta(S, v) = n(S, v)\Gamma(S)$.

Case $\phi(S, v) = 0$: In this case $\Delta(S, v) = \frac{1}{2^{m_1 - r}}$ by Equation 6, where $m_1 = \mu(S)$ and $r = |S|$. Also $b_k = 2^{\lambda_k - z_k}$ where $z_k = 0$ if $\lambda_k = 0$; and $z_k = 1$ if $\lambda_k > 0$. So

$$n(S, v) = (2^{\lambda_1 - z_1})(2^{\lambda_2 - z_2}) \cdots (2^{\lambda_m - z_m}) = 2^{(\lambda_1 + \cdots + \lambda_m) - (z_1 + \cdots + z_m)}.$$

Since $m_1 = \mu(S) = wt(\alpha^{(i_1)} \vee \cdots \vee \alpha^{(i_r)})$ we have $(z_1 + z_2 + \cdots + z_m) = m_1$. Also we have

$$\sum_{k=1}^m \lambda_k = \sum_{k=1}^m \sum_{j=1}^r \alpha_{j,k} = \sum_{j=1}^r \sum_{k=1}^m \alpha_{j,k} = \sum_{j=1}^r k_{i_j}.$$

So we get

$$n(S, v) = 2^{(\lambda_1 + \cdots + \lambda_m) - m_1} = \frac{2^{(k_{i_1} + \ldots + k_{i_r})}}{2^{m_1}}.$$

By definition $\Gamma(S) = \frac{2^r}{2^{k_{i_1} + \cdots + k_{i_r}}}$ and so $\Delta(S, v) = \frac{1}{2^{m_1 - r}} = n(S, v)\Gamma(S)$. This completes the proof of Claim 1.

Claim 2:

$$C_v = (-1)^{wt_2(v)} \sum_{\emptyset \neq S \subseteq [p]} (-1)^r n(S,v)\Gamma(S).$$

Proof of Claim 2: From Equation 4, we have $C_v = (-1)^{wt_2(v)} \sum_{R \in \mathcal{S}(v)} C(R)$.

Given $S = \{i_1, \cdots, i_r\}$, the vectors $\alpha^{(i_1)}, \ldots, \alpha^{(i_r)}$ are fixed. Let $D(S,v)$ be the set of all representations of v of the form $\{(u^{(i_1)}, \alpha^{(i_1)}), \ldots, (u^{(i_r)}, \alpha^{(i_r)}\}$. The value of $C(R)$ for any such representation R is equal to $\frac{2^r}{2^{k_{i_1}+\cdots+k_{i_r}}}$ and depends only on S. From definition, this value is equal to $\Gamma(S)$. In evaluating C_v we have to sum over all representations of v. The contribution of the set S to this sum is clearly $|D(S,v)|\Gamma(S)$. Thus we obtain

$$C_v = (-1)^{wt_2(v)} \sum_{\emptyset \neq S \subseteq [p]} |D(S,v)|\Gamma(S).$$

From this it is sufficient to show that $n(S,v) = |D(S,v)|$. There are two cases to consider.
Case $b_k = 0$ for some $k \in [m]$: This implies that v cannot be represented as $\{(u^{(i_1)}, \alpha^{(i_1)}), \ldots, (u^{(i_r)}, \alpha^{(i_r)}\}$ for any choice of $u^{(i_j)} \leq \alpha^{(i_j)}$. Hence $D(S,v) = \emptyset$ and so we have $n(S,v) = 0 = |D(S,v)|$.
Case $b_k > 0$ for all $k \in [m]$: In this case $D(S,v) \neq \emptyset$. Suppose $R = \{(u^{(i_1)}, \alpha^{(i_1)}), \ldots, (u^{(i_r)}, \alpha^{(i_r)}\} \in D(S,v)$, where for $1 \leq j \leq r$, $u^{(i_j)} = (u_{j,1}, \ldots, u_{j,m}) \in F_2^m$ and $\alpha^{(i_j)} = (\alpha_{j,1}, \ldots, \alpha_{j,m}) \in F_2^m$.

Define two $r \times m$ matrices M_1 and M_2 in the following manner. The (j,k)th entry of M_1 (resp. M_2) is $\alpha_{j,k}$ (resp. $u_{j,k}$). The conditions $u^{(i_j)} \leq \alpha^{(i_j)}$ is equivalent to $M_2 \leq M_1$, where the matrices are compared entrywise. Since R is a representation of v, we must have,

$$u_{1,k} \oplus \ldots \oplus u_{r,k} = v_k \text{ for each } k \in [m]. \tag{8}$$

Thus the problem of enumerating the set $D(S,v)$ is equivalent to enumerating matrices M_2 such that $M_2 \leq M_1$ and Equation (8) holds. Let c_k be the number of possible choices for the kth column of M_2. Then $|D(S,v)| = c_1 \ldots c_m$ and it is sufficient to show that $c_k = b_k$ for each $k \in [m]$. There are two cases to consider.
Subcase $\lambda_k = 0$: In this case $v_k = 0$ and $b_k = 1$. Since $\lambda_k = 0$, the kth column of M_1 is the all zero column. Hence the only possible choice for the kth column of M_2 is also the all zero column and so $c_k = 1 = b_k$.
Subcase $\lambda_k > 0$: In this case $b_k = 2^{\lambda_k - 1}$. We first observe that the XOR of the ones in the kth column of M_2 must be equal to v_k. Thus we have to consider two cases according as $v_k = 0$ or $v_k = 1$. First suppose $v_k = 0$. The kth column of M_1 contains λ_k many ones. Since $v_k = 0$, we can choose an even number of these ones to form a column for the matrix M_2. Thus the number of choices for the kth column of M_2 is $c_k = \binom{\lambda_i}{0} + \binom{\lambda_i}{2} + \cdots + \binom{\lambda_i}{2\lfloor \frac{\lambda_i}{2} \rfloor} = 2^{\lambda_k - 1} = b_k$. A similar argument holds when $v_k = 1$.

This completes the proof of Claim 2. Theorem 4.1 is a direct consequence of Claim 1 and Claim 2. ∎

Theorem 4.1 provides a method for computing C_v. However, this requires an algorithm to consider all possible subsets of $[p] = \{1, \ldots, p\}$. If p is even moderately large, this yields an impractical algorithm. Our next task is to show that it is actually not required to consider all the subsets of $[p]$. In this section we present one such special case and in the next section we present a general algorithm to compute C_v without generating all the subsets of $[p]$.

Theorem 4.2 *Let $Sup(A_g) = \{\alpha_1, \cdots, \alpha_p\}$ and $t > 0$ be such that for any $\{i_1, \ldots, i_{t+1}\} \subseteq \{1, \ldots, p\}$, we have $wt(\alpha_{i_1} \vee \cdots \vee \alpha_{i_{t+1}}) = m$. Then*

$$C_v = (-1)^{wt_2(v)} \left(\sum_{j=t+1}^{p} \frac{(-1)^j}{2^{m-j}} \binom{p}{j} + \sum_{\emptyset \neq S \subseteq [p], |S| \leq t} (-1)^{|S|} \triangle (S,v) \right)$$

Proof : We have from Theorem 4.1

$$\begin{aligned} C_v &= (-1)^{wt_2(v)} \sum_{\emptyset \neq S \subseteq [p]} (-1)^{|S|} \triangle (S, v) \\ &= (-1)^{wt_2(v)} \left(\sum_{\emptyset \neq S \subseteq [p], |S| \geq t+1} (-1)^{|S|} \triangle (S, v) + \sum_{\emptyset \neq S \subseteq [p], |S| \leq t} (-1)^{|S|} \triangle (S, v) \right). \end{aligned}$$

Under the given condition if $|S| \geq t+1$ then using Equation 6 we have $\mu(S) = m$. Hence

$$\begin{aligned} C_v &= (-1)^{wt_2(v)} \left(\sum_{\emptyset \neq S \subseteq [p], |S| \geq t+1} (-1)^{|S|} \frac{1}{2^{m-r}} + \sum_{\emptyset \neq S \subseteq [p], |S| \leq t} (-1)^{|S|} \triangle (S, v) \right) \\ &= (-1)^{wt_2(v)} \left(\sum_{j=t+1}^{p} (-1)^{j} \frac{1}{2^{m-r}} \binom{p}{j} + \sum_{\emptyset \neq S \subseteq [p], |S| \leq t} (-1)^{|S|} \triangle (S, v) \right) \end{aligned}$$

This completes the proof. ∎

Under the condition of Theorem 4.2, to evaluate C_v we only have to consider all non empty subsets of $[p]$ of cardinality at most t. If t is reasonably small, this is much better than considering all non empty subsets of $[p]$. In the next section, we develop this idea to obtain an algorithm to compute C_v.

5 Algorithm

In general we are interested in computing the Walsh transform of g at all the points $u \in F_2^m$. However, if m is relatively large (say around 50), then it will be practically impossible to compute the Walsh transform at all the 2^m points. In such a situation, it will be of interest to compute the Walsh transform at a particular point or for a small set of points. The fast Walsh transform takes time $O(m2^m)$ and computes the Walsh transform at all the 2^m points. In fact, to the best of our knowledge, there is no known algorithm which can compute the Walsh transform at a particular point in time less than 2^m.

Our approach is to design an algorithm that consists of two parts. In the first part, the algorithm does a certain amount of preprocessing and prepares a list. In the second part, the algorithm takes as input a particular $v \in F_2^m$ and computes C_v. (Using Theorem 3.3 this also gives us $W_g(v)$). Once the preprocessing is complete, the second part can be run for different v without running the first part. This makes it efficient to compute $W_g(v)$ for a set of v.

We start by defining certain parameters. For $\emptyset \neq S \subseteq [p]$ define $\sigma(S) \triangleq \bigvee_{i \in S} \alpha^{(i)}$. Then $\mu(S) = wt(\sigma(S))$. For $j = 0, \cdots, m$ define

$$B_j \triangleq \sum_{\mu(S)=j} (-2)^{|S|} \text{ and } B \triangleq \sum_{j=0}^{m-1} \sum_{\mu(S)=j} (-1)^{|S|} \Delta(S, v).$$

Note that if $S = \{i\}$ and $\alpha^{(i)} = (0, \ldots, 0)$, then $\mu(S) = 0$. The vector $(0, \ldots, 0) \in Sup(A_g)$ implies that the constant term in the ANF of g is equal to 1. The values of $B_0, \ldots, B_m$ are independent of v and only the value of B depends on v.

Theorem 5.1 *For any $v \in F_2^m$, we have*

$$C_v = (-1)^{wt_2(v)} \left[\frac{(-1)^p - 1 - B_0 - \cdots - B_{m-1}}{2^m} + B \right]. \tag{9}$$

Proof : Define

$$A \stackrel{\triangle}{=} \sum_{\emptyset \neq S \subseteq [p]} (-2)^{|S|} = \sum_{j=1}^{p} (-2)^j \binom{p}{j} = -1 + (-1)^p.$$

Then $A = \sum_{j=0}^{m} B_j$. Consequently, $B_m = A - B_0 - \cdots - B_{m-1}$. To see this note that $\sum_{j=0}^{m} B_j = \sum_{j=0}^{m} \sum_{\mu(S)=j} (-2)^{|S|} = \sum_{\emptyset \neq S \subseteq [p]} (-2)^{|S|} = A$.

From Theorem 4.1 we have

$$\begin{aligned}
C_v &= (-1)^{wt_2(v)} \sum_{\emptyset \neq S \subseteq [p]} (-1)^{|S|} \Delta(S,v) \\
&= (-1)^{wt_2(v)} \left[\sum_{j=0}^{m} \sum_{\mu(S)=j} (-1)^{|S|} \Delta(S,v) \right] \\
&= (-1)^{wt_2(v)} \left[\sum_{\mu(S)=m} (-1)^{|S|} \Delta(S,v) + \sum_{j=0}^{m-1} \sum_{\mu(S)=j} (-1)^{|S|} \Delta(S,v) \right] \\
&= (-1)^{wt_2(v)} \left[\frac{B_m}{2^m} + B \right] \\
&= (-1)^{wt_2(v)} \left[\frac{A - B_0 - \cdots - B_{m-1}}{2^m} + B \right]
\end{aligned}$$

This completes the proof. ∎

Our algorithm is based on Equation (9). The intuition behind the algorithm is the following. For most sets S, the value of $\mu(S)$ will be equal to m and will be accounted for by B_m. Thus if we can avoid computing B_m directly, then we will be saving a lot of computation. We will compute $B_0, \ldots, B_{m-1}$ and then use Equation (9) to compute the value of C_v. However, the value of B has to be computed. We next describe how this is done.

Define

$$\mathcal{D} = \{str \in F_2^m : str = \bigvee_{i \in S} \alpha^{(i)}, \emptyset \neq S \subseteq [p], wt(str) < m\}.$$

Write $\mathcal{D} = \{str_1, \ldots, str_n\}$ for some $n > 0$. For $1 \leq i \leq n$ define $val_i = \sum_{\sigma(S)=str_i} (-2)^{|S|}$. In the first part of our algorithm we prepare the list $\mathcal{L} = ((str_1, val_1), \ldots, (str_n, val_n))$. Note that this part does not depend on v.

For $str, v \in F_2^m$, define $\psi(str, v) \stackrel{\triangle}{=} \overline{str} \wedge v$ where $\overline{str}$ is bitwise complement of str. The operation $\wedge$ is performed bitwise on str and v. Suppose $\emptyset \neq S \subseteq [p]$ such that $\sigma(S) = str$ and $v \in F_2^m$. Then $\Delta(S, v) > 0$ if and only if $\psi(str, v) = (0, \ldots, 0)$. Thus

$$B = \sum_{\psi(str_i, v)=(0,\ldots,0)} \frac{val_i}{2^{wt(str_i)}}.$$

Also for $j = 0, \ldots, m-1$,

$$B_j = \sum_{wt(str_i)=j} val_i.$$

Hence once the list $\mathcal{L}$ is prepared, it is easy to compute $B_0, \ldots, B_{m-1}$ and B. Now we describe a method for preparing the list $\mathcal{L}$.

First we describe a rooted directed tree $\mathcal{T}$ whose nodes are the subsets of $[p] = \{1, \ldots, p\}$. The root node of $\mathcal{T}$ is the empty set. The children of a set S are the sets $S_1, \ldots, S_k$, where for $1 \leq i \leq k$, $S_i = S \cup \{\max(S) + i\}$ and $k = p - \max(S)$. This ensures that if S' is a node in the subtree rooted at S, then $S \subset S'$.

Our algorithm will traverse all the nodes S of $\mathcal{T}$ for which $\mu(S) < m$. (Note that $\mathcal{T}$ has 2^p nodes and if an algorithm is required to traverse all the nodes of $\mathcal{T}$, then the algorithm will be exponential in p.) From the structure of $\mathcal{T}$ we know that if $\mu(S) = m$ for some node S, then $\mu(S') = m$ for all nodes S' in the subtree rooted at S. This crucial fact makes the traversal particularly efficient. If during the the traversal we reach a node S with $\mu(S) = m$, then we need not visit any of the nodes in the subtree rooted at S. This means that we are effectively *pruning* the subtree rooted at S from $\mathcal{T}$. The more we encounter this pruning effect, the more efficient is our algorithm.

While traversing $\mathcal{T}$ we prepare the list $\mathcal{L}$ in the following manner. Initially $\mathcal{L}$ is the empty list. Let $\mathsf{First}(\mathcal{L}) = \{str : (str, val) \in \mathcal{L}\}$. Suppose we have reached a node S with $\mu(S) < m$. If $\sigma(S) \notin \mathsf{First}(\mathcal{L})$, then we add $(\sigma(S), (-2)^{|S|})$ to $\mathcal{L}$. On the other hand, if $\sigma(S) \in \mathsf{First}(\mathcal{L})$, then we update val to $val + (-2)^{|S|}$.

Thus the operations on $\mathcal{L}$ are search and insert. We implement $\mathcal{L}$ using a height balanced binary tree (see [1]). Hence each search/insert operation requires time $O(\log \mathcal{L})$. One such search or insert operation is required for each S such that $\mu(S) < m$. Also the total time spent at any node which is visited is $O(m)$. Hence the total time required by the algorithm is $O(m|\mathcal{S}_1| \log(|\mathcal{S}_2|))$, where

$$\left.\begin{array}{lcl} \mathcal{S}_1 & = & \{A \subseteq sup(A_g) : wt(\bigvee_{\alpha \in A} \alpha) < m\}; \\ \mathcal{S}_2 & = & \{B : B = \bigvee_{\alpha \in A} \alpha, wt(B) < m, \text{ for some } A \subseteq sup(A_g)\}. \end{array}\right\} \tag{10}$$

We next present the algorithm for computing C_v.

Algorithm ComputeCv

Inputs :

1. $sup(A_g)$, where g is an m-variable Boolean function.
2. $v \in F_2^m$.

Output : C_v.

Part 1 : Computation of list $\mathcal{L}$.
Set $\mathcal{L}$ equal to the empty list.
Set str equal to the empty string.
Traverse(str).
Assume $\mathcal{L} = ((str_1, val_1), \ldots, (str_n, val_n))$ at the end of Traverse.

Part 2 : Computation of C_v.
Set $B_0 = \cdots = B_{m-1} = 0$.
For $i = 1$ to n do
  if $(wt(str_i) = j))$ then $B_j = B_j + val_i$.
  if $(\psi(str_i, v) = (0, \ldots, 0))$ then $B = B + \frac{val_i}{2^{wt(str_i)}}$.
End For.
$C_v = (-1)^{wt_2(v)}(\frac{1}{2^m}((-1)^p - 1 - \sum_{i=0}^{m-1} B_i) + B)$.
Return C_v.

End Algorithm ComputeCv.

The subroutine Traverse() performs a depth first search on the tree $\mathcal{T}$ described before. The details of the algorithm Traverse() are given below.

Algorithm Traverse(str)

Input : a binary string str of length at most p.

For $i = |str|$ to $p-1$ do
Set $j = i - |str| - 1$.
Set $tstr = str||0^j||1$.
If $(\mu(tstr) < m)$ then
If $(\mu(tstr) \notin \mathsf{First}(\mathcal{L})$, then
Add $(\mu(tstr), (-2)^{wt(tstr)}$ to $\mathcal{L}$.
Else suppose $(\mu(tstr), val)$ is present in $\mathcal{L}$.
Set $val = val + (-2)^{wt(tstr)}$.
End If.
If $(|tstr| < p)$, then $\mathsf{Traverse}(tstr)$.
End If.
End For.
End Algorithm Traverse

From the above discussion we obtain the following result.

Theorem 5.2 *Let $v_1, \ldots, v_t \in F_2^m$ and g be an m-variable Boolean function. Then $\{W_g(v_i) : 1 \leq i \leq t\}$ can be computed in time $O(m(|\mathcal{S}_1|\log(|\mathcal{S}_2|) + t|\mathcal{S}_2|))$.*

Proof : Part 1 of Algorithm ComputeCv has to executed only once for all the vectors $v_1, \ldots, v_t$. This takes time $O(|\mathcal{S}_1|\log(|\mathcal{S}_2|))$. Part 2 of Algorithm ComputeCv has to run once for each of the vectors $v_1, \ldots, v_t$. Each execution of Part 2 takes time $O(|\mathcal{S}_2|)$. This gives the time complexity of the algorithm. Correctness of the algorithm follows from the previous discussion. ■

Remark : It is important to note that for small t it is possible to have $|\mathcal{S}_1|\log(|\mathcal{S}_2|) + t|\mathcal{S}_2| \ll \min(2^p, 2^m)$. In such situations it is possible to efficiently compute the Walsh transform of g for a small set of points. In Section 6, we provide some experimental data to support this intuition.

6 Experimental Results

Similar to Section 3, let $Sup(A_g) = \{\alpha^{(1)}, \ldots, \alpha^{(p)}\}$. We write $\alpha^{(i)} = (\alpha_{i,1}, \cdots, \alpha_{i,m})$ where $\alpha_{i,j} \in \{0,1\}$ for $1 \leq i \leq p$, $1 \leq j \leq m$. For $1 \leq j \leq m$, define $\rho_j = p - \sum_{i=1}^{p} \alpha_{i,j}$. Let $\rho_{max} = max_{j\in[m]}(\rho_j)$, $\rho_{avg} = \sum_{i=1}^{m} \frac{\rho_i}{m}$ and $w_{avg} = \sum_{i=1}^{p} \frac{wt(\alpha_i)}{p}$. We use the notation $\mathcal{S}_1$ and $\mathcal{S}_2$ as defined in Equation 10.

Tables 1 and 2 consider the cases $p > m$ and $p < m$ respectively. (Here we note that the class of Boolean functions for which p is less than m is also very rich. For example this class includes an important subset of the class of bent functions.) The results in these two tables show that ρ_i's (specially ρ_{max}) are crucial to the complexity of the algorithm. If ρ_{max} is comparatively larger than the other ρ_i's then the size of $|\mathcal{S}_1|$ is exponential in ρ_{max}. Also note that in general $|\mathcal{S}_1| \geq 2^{\rho_{max}}$.

m, p	$\lvert\mathcal{S}_2\rvert$	$\lvert\mathcal{S}_1\rvert, 2^{\rho_{max}}$	$\rho_{max}, \rho_{avg}, w_{avg}$
$100, 250$	546	$2135470, 1048576$	$20, 15, 94$
$100, 250$	547	$3184044, 2097152$	$21, 15, 94$
$100, 250$	549	$5288601, 4194304$	$22, 15, 94$
$100, 250$	549	$9482897, 8388608$	$23, 15, 94$
$100, 250$	550	$17871500, 16777216$	$24, 15, 94$
$100, 250$	552	$34648698, 33554432$	$25, 15, 94$
$100, 250$	553	$68203122, 67108864$	$26, 15, 94$

Table 1

m, p	$\lvert\mathcal{S}_2\rvert$	$\lvert\mathcal{S}_1\rvert, 2^{\rho_{max}}$	$\rho_{max}, \rho_{avg}, w_{avg}$
$100, 50$	2258	$365465, 131072$	$17, 15, 70$
$100, 50$	2392	$1340716, 1048576$	$20, 15, 70$
$100, 50$	2429	$8678491, 8388608$	$23, 15, 70$
$100, 50$	2465	$67386792, 67108864$	$26, 15, 70$

Table 2

From the definition, the parameters m, p, ρ_{avg} and w_{avg} satisfy the following relationship: $m \times \rho_{avg} = p \times (m - wt(w_{avg}))$. For fixed m and p, ρ_{avg} decreases as w_{avg} increases. Similarly, for fixed m and ρ_{avg}, as p increases so does w_{avg}. In Table 3, we show the change of w_{avg} with p for $m = 100$ and $\rho_{avg} = 16$.

m, p	$\lvert\mathcal{S}_2\rvert$	$\lvert\mathcal{S}_1\rvert, 2^{\rho_{max}}$	$\rho_{max}, \rho_{avg}, w_{avg}$
$100, 50$	9976	$5313186, 524288$	$19, 16, 68$
$100, 100$	2934	$6482400, 524288$	$19, 16, 84$
$100, 200$	1414	$4609688, 524288$	$19, 16, 92$
$100, 400$	827	$6644436, 524288$	$19, 16, 96$

Table 3

From Theorem 5.2, we note that the complexity of the algorithm is linear in m. Hence it is possible to run the algorithm in cases where m is large but ρ_{max} is small. Table 4 provides some data which illustrates this fact.

m, p	$\lvert\mathcal{S}_2\rvert$	$\lvert\mathcal{S}_1\rvert, 2^{\rho_{max}}$	$\rho_{max}, \rho_{avg}, w_{avg}$
$200, 200$	725	$96213, 32768$	$15, 12, 188$
$400, 200$	2175	$207969, 16384$	$14, 12, 376$
$800, 200$	1938	$528605, 65536$	$16, 12, 752$
$3200, 100$	2290	$243188, 16384$	$14, 12, 2816$
$3200, 100$	2502	$4436706, 4194304$	$22, 12, 2816$

Table 4

In the above examples we see that $|\mathcal{S}_2|$ is small. In Table 5 we provide some examples for which $|\mathcal{S}_2|$ is high but w_{avg} is low.

m, p	$\lvert\mathcal{S}_2\rvert$	$\lvert\mathcal{S}_1\rvert, 2^{\rho_{max}}$	$\rho_{max}, \rho_{avg}, w_{avg}$
$50, 25$	134260	$21432167, 1048576$	$20, 16, 18$
$60, 30$	150554	$48852312, 8388608$	$23, 16, 28$
$70, 35$	92849	$61196173, 16777216$	$24, 16, 38$
$80, 40$	13403	$17968626, 33554432$	$26, 16, 48$
$90, 45$	22728	$585495610, 134217728$	$27, 16, 58$

Table 5 :

As long as ρ_{max} is small (say less than 25), it is possible to run the algorithm for quite large values of both m and p. Table 6 provides some evidence of this fact.

m, p	$\lvert\mathcal{S}_2\rvert$	$\lvert\mathcal{S}_1\rvert, 2^{\rho_{max}}$	$\rho_{max}, \rho_{avg}, w_{avg}$
$400, 800$	1578	$1442226, 32768$	$15, 14, 393$
$400, 800$	1588	$2489671, 1048576$	$20, 14, 393$

Table 6 :

Note that the following are true for the experiments performed above.

1. $|\mathcal{S}_1|$ is exponential in ρ_{max} and hence the run time of the algorithm is exponential in ρ_{max}.
2. In most cases, $|\mathcal{S}_2|$ is small compared to $|\mathcal{S}_1|$. Also $|\mathcal{S}_2|$ tends to grow as w_{avg} decreases.
3. In all the examples given above, the chosen functions were non-degenerate on all the variables.

Finally, in summary we note that for certain types of Boolean functions on large number of variables algorithm ComputeCv can be used to compute the Walsh transform for a small set of points. This can provide some useful information about the Boolean function. For example, one can obtain the weight (or size of support) of g by computing $W_g(0)$. Also the ability to probe the spectral domain of the function at random points can provide an estimate of the nonlinearity of the function.

7 Possible Improvements

In this section we discuss possible ways of improving the algorithm. From Theorem 5.2, we see that the complexity of the algorithm depends on $|\mathcal{S}_1|$ and $|\mathcal{S}_2|$. These two parameters are determined by the ANF of the function. We discuss two ways of improving the efficiency.

Let g be the function under consideration whose ANF is A_g. Suppose we apply an affine transformation to the variables of g to obtain f such that $|sup(A_f)| < |sup(A_g)|$. In such a situation it might be possible to improve the run time of the algorithm. The problem is in obtaining an affine transformation which will perform this task.

Another approach to improve the efficiency is to look for a more general and compact algebraic representation of a function. We briefly discuss this approach below and indicate the difficulty of this method.

Let g be of the form

$$g(x_1, x_2, \ldots, x_m) = \bigoplus_{(\alpha,\beta)\in list} (x_1 \oplus \beta_1)^{\alpha_1} \cdots (x_m \oplus \beta_m)^{\alpha_m}$$

where $list = ((\alpha^{(1)}, \beta^{(1)}), \cdots, (\alpha^{(p)}, \beta^{(p)}))$ and $\alpha^{(i)}, \beta^{(i)} \in F_2^m$. The ANF of g is a special case of the above form where $\beta_1 = \cdots = \beta_m = 0$.

Let r be such that $1 \leq r \leq |list|$ and suppose $v = u^{(i_1)} \oplus \cdots \oplus u^{(i_r)}$ where $u^{(i_j)} \leq \alpha^{(i_j)}$ then we say $R = \{(u^{(i_1)}, \alpha^{(i_1)}, \beta^{(i_1)}), \cdots, (u^{(i_r)}, \alpha^{(i_r)}, \beta^{(i_r)})\}$ is a representation of v .

Let $R \in \mathcal{S}(v)$, such that $R = \{(u^{(i_1)}, \alpha^{(i_1)}, \beta^{(i_1)}), \cdots, (u^{(i_r)}, \alpha^{(i_r)}, \beta^{(i_r)})\}$ where $u^{(i_j)} \leq \alpha^{(i_j)}$ and for $1 \leq j \leq r$, let $k_{i_j} = wt(\alpha^{(i_j)})$. Define

$$C(R) = \frac{(-1)^r}{2^{(k_{i_1}-1)+\cdots+(k_{i_r}-1)}} (-1)^{\langle u^{(i_1)}, \beta^{(i_1)}\rangle \oplus \cdots \oplus \langle u^{(i_r)}, \beta^{(i_r)}\rangle}.$$

Now it is simple to check that Theorem 3.3 holds. But in this situation it seems difficult to simplify the formula and obtain an algorithm to compute C_v.

8 Concluding Remarks

In this paper we have developed an algorithm to compute the Walsh transform of Boolean function at a point from its algebraic normal form. This can also be extended to evaluate the Walsh transform for a set of points. The advantage of our method is that the run time is linear in m and hence can be used in situations where the fast Walsh transform is not applicable.

An important parameter in the study of a Boolean function is its nonlinearity. To compute nonlinearity, it is necessary to compute the Walsh transform at all the points. For Boolean functions on a large number of variables (≥ 40), it is practically impossible to evaluate to perform this task. In certain situations, our algorithm can be used to evaluate the Walsh transform at any particular point. In such situations, it is possible to use our algorithm in conjunction with some randomized heuristic (simulated annealing/ hill climbing/ genetic algorithms) to estimate the nonlinearity of the function. Thus we feel that the algorithm developed in this paper is a valuable technique for study and analysis of Boolean functions.

References

[1] E. Horowitz and S. Sahni. *Fundamentals of Data Structures*, W. H. Freeman and Co., 1983.

[2] F. J. MacWillams and N. J. A. Sloane. *The Theory of Error Correcting Codes.* North Holland, 1977.