設 計 自 動 化 67－5
（1993．6．25）

多入力論理関数に対する効率的なWalshスペクトルの計算手法とそのテクノロジマッピングへの応用

藤田昌宏（富士通研究所）
Edmund M. Clarke　Xudong Zhao （CMU）
Jerry Yang （Stanford Univ.）

Walsh変換は、論理関数の解析などに応用でき、重要な技術である。しかし、従来の手法では、変換できる論理関数の入力数は２０が限度であり、実用規模の論理回路を扱えなかった。本稿では、二分決定グラフを利用することで、数百入力の論理関数のWalsh変換を行える新しい手法を示す。また、具体的な応用例として、論理関数を考慮したセルをマップするBoolean technology mappingへのWalsh変換の応用結果について報告し、従来手法に比べ、50%以上高速化できる場合のあることを示す。

Walsh spectral transforms for large Boolean Functions with Application to Technology Mapping

Masahiro Fujita (FUJITSU LABORATORIES LTD.)
Edmund M. Clarke　Xudong Zhao (CMU)
Jerry Yang (Stanford Univ.)

The Walsh transform has numerous applications in computer-aided design fields, but the usefulness of these techniques in practice has been limited by the size of the Boolean functions that can be transformed. Currently available techniques limit the functions to less than 20 variables. In this paper, we show how to compute concise representations of the Walsh transform for functions with several hundred variables. We have applied our techniqeus to Boolean technology mapping and, in certan cases, we obtained a speed up of as much as 50% for the maching phase.

1 はじめに

Walsh 変換は、テストや論理合成をはじめとして、論理設計支援の様々な分野に応用できる重要な技術である [3, 4]。しかし、Walsh 変換の計算結果は、2^n個の要素をもつベクトルの形で得られるために、n が 20 以上になると結果をリストアップすることはできない。また、計算手法自体も、真理値表や積和形論理式が利用されてきたために、入力数が多い場合には、処理しようがなかった。

本稿では、数百入力をもつ論理関数に対する Walsh 変換も計算可能な手法について述べる。ベクトルや再帰的に定義された行列（Walsh 変換の行列が当てはまる）の表現に二分決定グラフ（BDD）[1] を利用し、Walsh 変換自体を BDD 間の演算として計算する。ただし、Walsh 変換時には、計算結果として、整数値が求まるので、BDD の定数ノードに任意の整数を定義できる拡張された BDD を用いる（定数ノードが 3 個以上あることを認めるように拡張されているので、Multi Terminal BDD, MTBDD とも呼ばれる）。

$2^n \times 2^n$の整数値行列を $n+m$ 個の Boolean 入力変数から整数への関数とみなし、BDD で表現する。このようにすると、行列演算は、対応する整数関数間の演算として計算することができるため、Walsh 変換も BDD 上の演算として求めることができる。Walsh 行列は、再帰的に定義されており、BDD で表現すると変数の数に比例する大きさになる。

本稿では、提案する手法を Boolean テクノロジマッピングに適用した結果についても報告する。今、与えられた 2 つの論理関数が入力変数順序、極性、それに出力の極性をうまく調整することで一致されることができる時、Boolean matchable であるというとすると、Boolean テクノロジマッピングでは、各部分回路がセルライブラリの各セルと Boolean matchable か否かを判定する技術が基本となり、またもっとも計算時間のかかる部分である。効率化のためには、マッチしそうにない場合をいち早く検出できるようなフィルタでまずチェックするのがよい。ここでは、Walsh 変換の結果の一部を効率的にフィルタとして利用できることを示し、実験結果からその有効性を示す。実際、Walsh 変換を元にしたフィルタを通過した場合はすべて Boolean matchable であった。

2 章では、BDD を用いた整数値行列の表現と、それを用いた行列演算の実現手法について述べる。次に 3 章では Walsh 変換について説明し、4 章では Boolean テクノロジマッピングへの応用を示す。最後に 5 章で結論を述べる。

2 BDD を用いた行列演算

最初に 2 値入力整数値関数を BDD で表現することを考えて、その後、整数値行列を BDD で表現することを考える。

D_nを $\{0, 1, ..., 2^{n+1}-1\}$ の範囲の整数とし、Bを 0,1 の 2 値とする。また、$f : B^m \rightarrow D_n$ を m ビットの 2 値ベクトルから整数値 D_nへの関数とする。fを BDD で表現するには、BDD の定数ノードに D_nの中の任意の整数値を指定できるようにすれば良い。今、関数 fは D_nの内、$\{n_1, ..., n_N\}$ という値のいずれかしか返さないとする。すると、関数 fにより、空間 B^mは、$S_i = \{\overline{x} | f(\overline{x} = n_i\}$ で決まる N個の集合 $\{S_1, ..., S_N\}$ に分けられる。これにより、関数 fの標準形として$\sum_{i=1}^{N} f_i(\overline{x}) \cdot n_i$ として関数 fを表現することができる。例えば、もし $x_1 \vee x_2$ が 1 なら 3 をさもなければ 4 という値を返す関数は、図 1のように表される。このように定数ノードに任意の整数などの 0,1 以外の値を認めるようにした BDD をここでは、MTBDD と呼ぶ。

この表現を利用して、関数間の算術演算は、以下のように行なえる。今、f, gが標準形として与えられているとする。

$$f(\overline{x}) = \sum_{i=1}^{N} f_i(\overline{x}) \cdot n_i, g(\overline{x}) = \sum_{j=1}^{N'} g_j(\overline{x}) \cdot n'_j$$

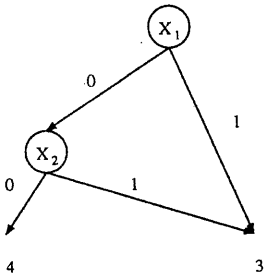

図 1: もし $x_1 \vee x_2$ が 1 なら 3 をさもなければ 4 という値を返す関数に対する BDD

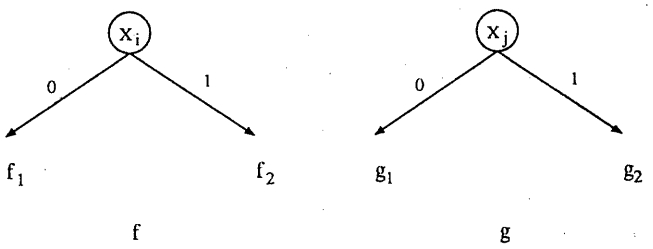

図 2: f と g に対する BDD

すると、f と g の和 h は以下のように計算できる。

$$\begin{aligned} h(\overline{x}) &= f(\overline{x}) + g(\overline{x}) \\ &= \sum_{i=1}^{N} f_i(\overline{x}) \cdot n_i + \sum_{j=1}^{N'} g_j(\overline{x}) \cdot n'_j \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N'} f_i(\overline{x}) g_j(\overline{x}) (n_i + n'_j) \end{aligned}$$

これから、BDD を用いて $f+g$ を以下のように効率的に計算することができる。基本的に BDD に対する *apply* 演算を行なっていけばよい。

- もし f が定数ノードならば、その整数値を g の各定数ノードに加える。
- もし g が定数ノードならば、その整数値を f の各定数ノードに加える。
- どちらでもなければ、f, G は図 reffig:BDDfg のようになっているので、図 3のように X_i と X_j の変数順序に従って、$f+g$ に対する BDD を構成していく。ただし、結果には冗長なノードが生じる可能性があるので、それを取り除く、BDD の reduction 処理が必要となる。以上は、加算の場合であったが、任意の演算も同様に実行できる。

また、行列演算は以下のように実行できる。M を各要素が値として D_N を持つ、2^k 行 2^l 列の行列とする。すると、$M : B^{k+l} \rightarrow D_n$ への関数 M を考え、$M(x,y) = M_i j$ となるようにする。ただし、x は 2 値ベクトル i

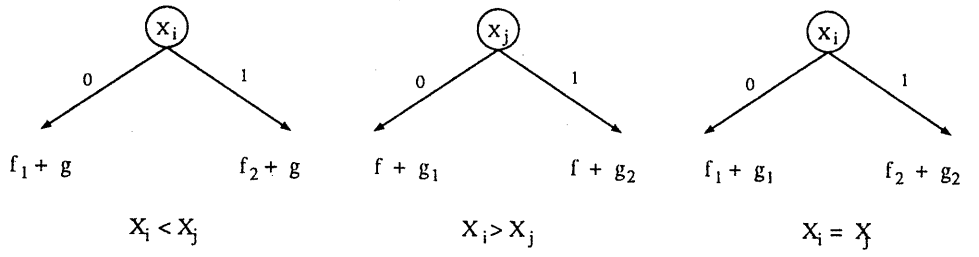

図 3: $f + g$に対する BDD

に対応し、yは 2 値ベクトル jに対応させる。このような関数 Mを用いれば、任意の行列を上に述べた 2 値入力整数値出力の関数として表現でき、従って、BDD として表して行列演算を行なえることになる。

以下、ここでは行列演算の例として、絶対値、加算、ベクトルのソート、行列の行や列ごとの要素の合計の計算、それに行列のかけ算を考えていく。

- 絶対値

  与えられた行列の各要素の絶対値を取るのは、MTBDD の各定数ノードの値の絶対値を取るだけでよいので、MTBDD を一回たどればよい。

- 加算

  これは、行列を対応する整数値関数で表現すれば、上に書いたように各要素ごとの加算は簡単に実現できる。この場合、必要な手間は、2 つの行列に対応する 2 つの MTBDD 間で *apply*演算を行なうことになるので、2 つの MTBDD の大きさの積程度となる。

- ベクトルのソート

  これは、MTBDD では単に定数ノードの値のみをソートすればよい。MTBDD を 1 回辿り、定数ノードの値を列挙してそれをソートする形で実現できる。

- 行列の行や列ごとの要素の合計の計算

  行列の各列ごとの和を計算するには、列に対する変数 yの全ての値に関して各要素値の合計を求める必要があるが、これは、次のように y変数を 1 つずつ除去していくことで計算できる。

$$
\begin{aligned}
& \sum_{y_1 y_2 \cdots y_m} M(\overline{x}, y_1, y_2, \cdots, y_m) \\
= & \sum_{y_1 y_2 \cdots y_{m-1}} \sum_{y_m} M(\overline{x}, y_1, y_2, \cdots, y_m) \\
= & \sum_{y_1 y_2 \cdots y_{m-1}} (M(\overline{x}, y_1, y_2, \cdots, y_{m-1}, 0) + M(\overline{x}, y_1, y_2, \cdots, y_{m-1}, 1))
\end{aligned}
$$

これは、BDD 上で Existential quantified variable を除去する処理と同じで BDD で実行できる。多くの場合効率良く実行できるが、最悪の場合は変数の数に対して指数的な計算量となる。

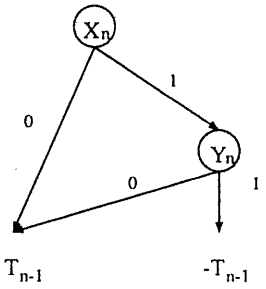

図 4: T_nに対する BDD

- 行列のかけ算

  $2^k \times 2^l$の行列 A と $2^l \times 2^m$の行列 B のかけ算 C = A × C は以下のように計算できる。

$$C(\overline{x},\overline{z}) = \sum_{\overline{y}} A(\overline{x},\overline{y})B(\overline{y},\overline{z})$$

  加算の部分は上に示したように計算できる。これも、一般的には変数の数に対して指数的な計算量となるが、多くの例で実際には効率的に処理できる。

3 Walsh 変換

Walsh 変換は、論理関数の解析手法として重要な技術である [3]。ここでは、前章で示した MTBDD により、Walsh 変換が極めて効率よく実行できることを示す。

Walsh 変換はを表す行列 T_Nは次のように再帰的に定義される。

$$T_0 = 1 \qquad T_n = \begin{bmatrix} T_{n-1} & T_{n-1} \\ T_{n-1} & -T_{n-1} \end{bmatrix}$$

n 次の Walsh 行列は 2^n行 2^n列の行列となり、n 変数の論理関数を解析するのに用いられる。行列の各行を $x_1,...,x_n$でエンコードし、各列を $y_1,...,y_n$でエンコードする。上の定義から Walsh 行列に対応する整数関数 T_nは以下のような性質をもつ。

$$\begin{aligned} T_n(y_n,\cdots,y_1,x_n,\cdots,x_1) &= \begin{cases} T_{n-1}(y_{n-1},\cdots,y_1,xn-1,\cdots,x_1) & if(x_n y_n \neq 1) \\ -T_{n-1}(y_{n-1},\cdots,y_1,x_{n-1},\cdots,x_1) & if(x_n y_n = 1) \end{cases} \\ &= T_{n-1}(y_{n-1},\cdots,y_1,x_{n-1},x_{n-1},\cdots,x_1)\cdot(if x_n y_n = 1 then -1 else 1) \end{aligned}$$

これから、T_nは図 4のような形になることが分かる。

Walsh 変換とは論理関数 fと Walsh 行列 T_nを掛けた結果であり、Walsh 変換の結果は、2^n個の要素をもつベクトルとなる。これらのベクトルの各要素は、0 定数関数、関数 $x_1, x_2, x_3, x_1 \oplus x_2, x_1 \oplus x_2 \oplus x_3$,...などとの真理値表との差を表している。これらベクトルの要素を Walsh スペクトルと呼ぶ。

これらの値は、関数を解析する際の重要な情報となる。例えば、0 定数関数との差を示す 0 次の相関値と関数 $x_1, x_2, ..., x_n$との差を表す 1 次の相関値のみを用いて閾値関数を全て区別することができる。

example circuit					Walsh		distinct
circuit	総入力数	output	総ゲート数	BDD	BDD	time	coefficients
c1355	41	1325gat	546	9451	5102	134	4
c1908	33	9	880	3607	1850	44	18
c3540	50	361	1669	520	15985	171	39
c5315	178	854	2307	210	925	57	8
50-bit adder	100	C_{50}	250	151	7456	23	100
100-bit adder	200	C_{100}	500	301	29906	128	200

表 1: Walsh 変換結果

また、与えられた2つの関数が入力変数の置換と極性とそれに出力値の極性を除いて等価であるか否かの判定は、後の述べる Boolean テクノロジマッピングの基本技術であり、NPN 等価性と呼ばれる。これを完全に判定することは容易ではないが、適切な必要条件として Walsh スペクトルと利用することができる。例えば、NPN 等価であるためには、2つの関数の Walsh スペクトルにおいて、0次の相関値の絶対値が等しく、また、1次の相関値（n 変数なら n 個ある）の絶対値を各々ソートしたものが2つのスペクトルで一致しなければならない（これは、あくまで必要条件であり、十分条件ではない）。従って、0次と1次の Walsh スペクトルを計算して比較する処理を Boolean テクノロジマッピングにおける NPN 等価性判定のフィルタとして利用することができる。

この応用については、次章で述べることとし、ここでは、ISCAS のベンチマーク回路や加算器の1つの出力の論理関数について Walsh スペクトルを計算した結果を表 1に示す。これから分かるように数百入力の論理関数に対する Walsh スペクトルも数分で計算できている。なお、使用計算機は DEC 5000/240 である。

Walsh スペクトルの計算は、従来は真理値表ベースか、積和形論理式から計算する [2] もののみであり、入力数の多い関数や、積和形にすると積項数が爆発してしまうような関数（例えば ISCAS のベンチマーク回路）は扱えなかった。しかし、ここで示したように MTBDD を利用することにより、実用時間で効率よく求めることができる。

4 Boolean テクノロジマッピングへの応用

前章でも述べたように、Walsh 変換の1つの応用として、Boolean テクノロジマッピングで使用される NPN 等価性判定問題のフィルタとしての利用がある。ここでは、スタンフォード大学で開発された *Ceres* [5, 6] と呼ばれる Boolean テクノロジマッパに Walsh 変換に基づくフィルタを実装した結果について報告する。

Ceres にも元々フィルタは実装されている。それは、いくつの入力間で関数が対象となっているかと、いくつの入力について関数が単調になっているかによるものであり、2つの関数が不一致の場合の多くをフィルタできているが、完全ではない（フィルタを通過したが、結果的に不一致である場合もある）。

ここでは、この元のフィルタを使用する場合は、Walsh スペクトルによるフィルタを使用する場合の比較を行なった。ベンチマーク回路を最高9入力まであるセルライブラリにマップした。結果を表 2に示す。表には、NPN 等価性判定に要した時間のみを示している。Depth は、テクノロジマッピングのために切り出す回路の段数であり、これが大きいほどより入力数の多い論理関数の NPN 等価性判定を行なっていることになる。表から分かるように、Depth が大きいほど Walsh フィルタを利用した NPN 等価性判定の方が処理速度が速くなっている。

本結果を真理値表を利用した同様のテクノロジマッピングプログラム [7] と比較すると、高々9入力程度の

Cricuit	Depth3				Depth5				Depth7			
	Walsh		Ceres		Walsh		Ceres		Walsh		Ceres	
	M_w	time	M_c	time	M_w	time	M_c	time	M_w	time	M_c	time
C1355	532	687	968	372	639	840	1096	509	641	1068	1102	628
C3540	1423	1993	3550	1714	1447	3506	4401	4046	1570	7006	4855	16245
C432	272	367	613	336	288	615	740	619	302	1503	813	1755
C880	429	616	973	453	447	1214	1170	1170	460	2600	1329	4471
alu2	697	926	2374	1150	718	1544	3234	2394	1398	7564	4168	8298
alu4	1195	1510	4047	1964	1214	2350	5400	3981	2263	10699	6827	13230
apex6	718	876	1682	835	730	1063	1778	1227	735	1521	1808	2533
f51m	329	442	1158	568	341	788	1550	1351	637	2920	1968	4550
x1	2135	2489	8493	4269	2157	4309	13526	9288	5997	29244	17819	24309
z4ml	265	325	933	417	2374	682	1281	1033	608	2657	1700	3225

表 2: Boolean テクノロジマッピング結果：NPN 等価性判定のみ

セルしかないライブラリを使用しているにもかかわらず、本結果の方が数倍高速である。これから、MTBDD を用いた Walsh 変換は、数入力程度でも意味があることが分かる。

M_w, M_cは各々フィルタを通過した場合の数を示している。これから、いつも Walsh フィルタの方がより強くフィルタとして働いていることが分かる。ここで注目すべき事実として、Walsh フィルタは通過した時は実際に NPN 等価であった。つまり、Walsh フィルタをこの場合、完全に働いていると言える。

5 おわりに

本稿では、ます、定数ノードに任意の整数（あるいは有限集合の要素）を認めるように拡張した MTBDD により、行列演算が実行できることを示した。そして、論理関数の解析でよく利用される Walsh 変換行列は MTBDD でコンパクトに表現でき、従来求められなかったような数百入力あるような論理関数の Walsh スペクトルも計算できることを示した。さらに、応用として、Boolean テクノロジマッピングの Walsh スペクトルの計算を利用して、従来より高速化できることを示した。

Walsh 変換の応用は数多く考えられるため、MTBDD を利用した Walsh スペクトルの計算の利用価値は大きいと考えられる。また、MTBDD の行列演算への応用に関しても研究すべき点は多い。例えば、疎行列に対しては、行列内に異なる値の要素が少ないため、MTBDD による表現は他の手法と比較してもコンパクトになる。この性質を利用して、疎行列の演算が中心となるような、例えば回路シミュレーションなどへの応用も期待できる。今後、検討していきたい。

参考文献

[1] R.E. Bryant. "Graph-based algorithms for boolean function manipulation". *IEEE Trans. Computer*, Vol. C-35, No. 8, pp. 667–691, Aug. 1986.

[2] B.J. Falkowski, I. Schafer, and M.A. Perkowski. "Calculation of the Rademacher-Walsh spectrum from a reduced representation of Boolean functions". In *EURO-DAC '92*, pp. 181–186, 1992.

[3] S.L. Hurst, D.M. Miller, and J.C. Muzio. *"Spectral Techniques in Digital Logic"*. Academic Press, 1985.

[4] R.J. Lechner. "A transform approach to logic design". *IEEE Transactions on Computers*, Vol. C-19, No. 7, 1970.

[5] F. Mailhot and G. De Micheli. "Technology mapping using boolean matching and don't care sets". In *Proc. of European Design Automation Conference*, 1990.

[6] G. De Micheli, David Ku, F. Mailhot, and T.K. Truong. "The olympus synthesis system for digital design". *IEEE design and test of computers*, Oct. 1990.

[7] J. Yang and G. De Micheli. "Spectral techniques for technology mapping". Technical Report Technical Report CSL-TR-91-498, Stanford University, Dec. 1991.