2004-10

A Method to Find the Best Mixed Polarity Reed-Muller Expression Using Transeunt Triangle

Dueck, Gerhard W.

A Method to Find the Best Mixed Polarity Reed-Muller Expression Using Transeunt Triangle

Gerhard W. Dueck, Dmitry Maslov,
Department of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3 CANADA

Jon T. Butler,
Department of Electrical and
Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121, U.S.A.

Vlad P. Shmerko, and Svetlana N. Yanuskevich
Inst. of Comp. Science
Inf. Systems, Technical University
Zolnierska str. 49
71210 Szczecin, POLAND

October 28, 2004

Abstract

In this paper, we use the transeunt triangle in an efficient algorithm to find the minimum mixed polarity Reed-Muller expression of a given function. This algorithm runs in $O(n^3)$ time and uses $O(n^3)$ storage space. We demonstrate this algorithm on benchmark functions, and we extend it to multi-output functions.

1 List of symbols

$\oplus$	modulo-2 addition
$+$	integer addition
$(\gamma_1, \gamma_2, ..., \gamma_n)$	ternary number of the element of RTT
$(\epsilon_1, \epsilon_2, ..., \epsilon_n)$	binary number of needed for a given polarity element
$e_{i,j}$	(i, j)-th element of transeunt triangle
$P = (\rho_1, \rho_2, ..., \rho_n)$	polarity of a mixed polarity Reed-Muller expression
RTT	restricted transeunt triangle
T	transeunt triangle for the function $f(x_1, x_2, ..., x_n)$
T_0, T_1, T_2	leftmost, rightmost and upper sub-triangles of T

2 Introduction

Much research has been devoted to *sum-of-product* expressions in which *sum* is the Exclusive-OR function and *product* is the AND of variables or complements of variables [1]. It is known that such expressions require, on the average, fewer product terms than OR sum-of-products [8]. Another advantage is ease of testability [7].

AND-EXOR circuits have been used in arithmetic, error correcting, and telecommunications applications [13], [9].

Such expression can be used to classify a given function into an equivalence class of functions, where two functions are equivalent if one is transformed into the other by permuting variables, complementing variables, and/or complementing the function [12], and is useful for determining library cells in CAD tools. This is called *Boolean matching* and is important in the determination of library cells for use by computer-aided design tools.

This paper focuses on a special class of AND-EXOR circuits, called mixed polarity Reed-Muller (MPRM) expressions. In an MPRM expression of a given function $f(x_1, x_2, ..., x_n)$, every variable appears either 1. complemented, 2. uncomplemented, or 3. in *both* forms, in which case all terms contain the variable. If all variables are uncomplemented (complemented), the MPRM expression is called the Positive (Negative) Polarity Reed-Muller or PPRM (NPRM) form. A fixed polarity Reed Muller (FPRM) expression is one where each variable appears either complemented or uncomplemented, but never in both forms. FPRM expression are a subset of MPRM expressions. MPRM expressions are *unique* [2]. Thus, only one representation exists for the PPRM or NPRM or indeed any MPRM of $f(x_1, x_2, ..., x_n)$. This leads to the question of which of the MPRM's produces the fewest product terms [5], [6], [10].

3 Background

In this section we give the formal definitions of the terms used in this paper.

Definition 1. Let $P = (\rho_1, \rho_2, ..., \rho_n)$ where $\rho_1, \rho_2, ..., \rho_n \in \{0, 1, 2\}$ be the polarity of a mixed polarity Reed-Muller expression for the function $f(x_1, x_2, ..., x_n)$, such that x_i appears complemented if $\rho_i = 0$, x_i appears uncomplemented if $\rho_i = 1$ and x_i appears mixed if $\rho_i = 2$.

If $\rho_i = 2$, then f can be decomposed as

$$f(x_1, x_2, ..., x_n) = x_i f_0(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) \oplus \bar{x_i} f_1(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n),$$

where $f_0(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ and $f_1(x_1, .., x_{i-1}, x_{i+1}, ..., x_n)$ have polarity $(\rho_1, ..., \rho_{i-1}, \rho_{i+1}, ..., \rho_n)$. ►

Kronecker

Definition 2. The cost of a Reed-Muller expression is the number of all non-zero coefficients of the corresponding polynomial. ►

4 Minimization Algorithm

4.1 Transeunt Triangle

Let $f(x_1, x_2, ..., x_n)$ be a switching function of n variables, where $x_i \in \{0, 1\}$. We seek a mixed polarity Reed-Muller expression for f with minimal cost. An design algorithm based on the transeunt triangle, previously applied to symmetric functions, [**?**], [**?**], [**?**], [11], is also useful for arbitrary functions.

Definition 3. The **transeunt triangle** for $f(x_1, x_2, ..., x_n)$ is a triangle of 0's and 1's where the bottom row is the truth vector of f. The j-th element in the i-th row of the triangle is denoted by $e_{i,j}$, where the indices run from left-to-right and top-to-bottom starting with $i = 0$ and $j = 0$, respectively. . The truth vector corresponds to the elements $e_{2^n,0}, e_{2^n,1}, ..., e_{2^n,2^n-1}$. The element $e_{2^n,k}$ corresponds to $f(\alpha_1, \alpha_2, ..., \alpha_n)$, where $k = (\alpha_1, \alpha_2, ..., \alpha_n)$ is the binary representation of integer k. Other elements are related by $e_{i,j} = e_{i+1,j} \oplus e_{i+1,j+1}$. ►

Example 1. The transeunt triangle for $f(x_1, x_2) = 1 \oplus x_1 \oplus x_2$ is shown in Fig. 1.

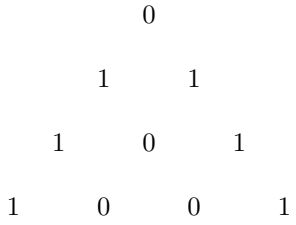

Figure 1: transeunt triangle for $f(x_1, x_2) = 1 \oplus x_1 \oplus x_2$.

Note that a transeunt triangle for an n-variable function has a width of 2^n and a height of 2^n. There are $(4^n + 2^n)/2$ elements in total. Besides the defining relation, there are other relations among elements in the transeunt triangle, as shown below.

Lemma 1. For the transeunt triangle of $f(x_1, x_2, ..., x_n)$, $e_{i,j} = e_{i,j+2^k} \oplus e_{i+2^k,j+2^k}$ where $k \geq 0$.

Proof. By induction: If $k = 0$, then $e_{i,j} = e_{i+1,j} \oplus e_{i+1,j+1}$ is true by the definition of transeunt triangle. Assume the hypothesis is true for $k = m - 1$. Therefore,

$$e_{i,j} = e_{i,j+2^{m-1}} \oplus e_{i+2^{m-1},j+2^{m-1}} \tag{1}$$

Similarly,

$$e_{i,j+2^{m-1}} = e_{i,j+2^{m-1}+2^{m-1}} \oplus e_{i+2^{m-1},j+2^{m-1}+2^{m-1}} \tag{2}$$

and

$$e_{i+2^{m-1},j+2^{m-1}} = e_{i+2^{m-1},j+2^{m-1}+2^{m-1}} \oplus e_{i+2^{m-1}+2^{m-1},j+2^{m-1}+2^{m-1}} \tag{3}$$

Substituting (2) and (3) into (1) yields the hypothesis. ■

Since the truth vector for functions with one or more variables has an even number of entries, it can be divided evenly into two equal parts. Each part produces, on its own, two subtriangles. Indeed, we can identify three subtriangles, as shown in Fig. 2.

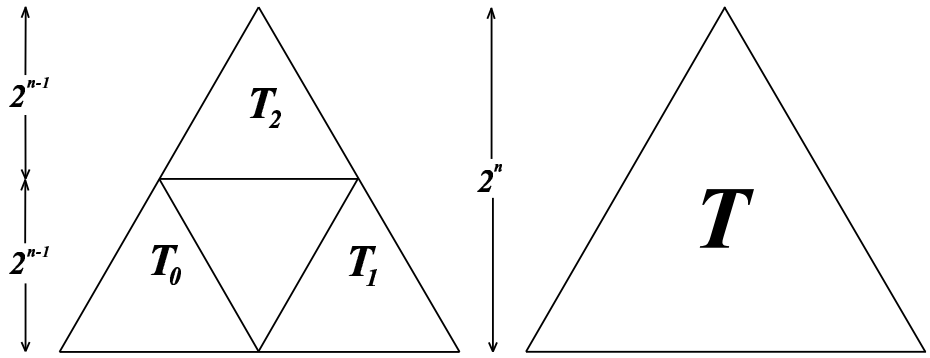

Figure 2: Decomposition of the transeunt triangle T.

Lemma 2. Let T be transeunt triangle for a switching function $f(x_1, x_2, ..., x_n)$. If we divide T onto 3 triangles T_0, T_1 and T_2 as shown in Fig. 2, then

T_0 is the transeunt triangle for the function $f(0, x_2, .., x_n)$,

T_1 is the transeunt triangle for the function $f(1, x_2, ..., x_n)$ and T_2 is the transeunt triangle for the function $f(0, x_2, ..., x_n) \oplus f(1, x_2, ..., x_n)$.

Proof. First, denote $e^0_{i,j}$ to be the (i, j)-th element from T_0, $e^1_{i,j}$ to be the (i, j)-th element from T_1 and $e^2_{i,j}$ to be the (i, j)-th element from T_2. T_0 can be considered as the transeunt triangle for $f(0, x_2, ..., x_n)$ because it's bottom row is the truth vector of $f(0, x_2, ..., x_n)$ and every element $e^0_{i,j} = e^0_{i+1,j} \oplus e^0_{i+1,j+1}$ by the construction of T. Similarly, T_1 is the transeunt triangle for $f(1, x_2, ..., x_n)$.

Note, that by Lemma 1 and the statement that the height of transeunt triangle is a power of two $e^2_{i,j} = e^0_{i,j} \oplus e^1_{i,j}$. Therefore, the bottom of T_2 will be "exclusive or" of the elements of bottom row of transeunt triangles for $f(0, x_2, ..., x_n)$ and $f(1, x_2, ..., x_n)$, which is the

"exclusive or" of truth vectors of $f(0, x_2, ..., x_n)$ and $f(1, x_2, ..., x_n)$, which is the truth vector for $f(0, x_2, ..., x_n) \oplus f(1, x_2, ..., x_n)$. Other (non-bottom) elements of T_2 by the construction of T are corresponding sums. So, T_2 is the transeunt triangle for the function $f(0, x_2, ..., x_n) \oplus f(1, x_2, ..., x_n)$. ■

The algorithm can be described intuitively, as follows. First, given a function $f(x_1, x_2, ..., x_n)$ as a truth vector and a polarity $P = (\rho_1, \rho_2, ..., \rho_n)$, we construct the corresponding transeunt triangle. To obtain the cost, we divide the triangle into parts, extracting elements. First, divide the transeunt triangle T onto the three triangles T_0, T_1 and T_2. Consider variable x_1 and the corresponding ρ_1 in the polarity P. If $\rho_1 = 0$ retain T_0 and T_2. If $\rho_1 = 1$ retain T_1 and T_2. And, if $\rho_1 = 2$ retain T_0 and T_1. In other words, if $\rho_1 = i$ where $i \in \{0, 1, 2\}$, then delete T_i. Next, repeat this process with x_2 (and ρ_2). At the end of this step we have four triangles. Continue this operation for all variables until each triangle is a single element. Such a one-element triangles are called *elementary triangles*. There will be 2^n of them. The sum of all these 2^n values is the cost for the given polarity (view logic values as integers and sum as integer sum).

A formal version of the algorithm is shown in Fig. 3.

Algorithm 1

```
GetCost(T, P)
    /* Calculate the cost for polarity P              */
    /* for a function T given as transeunt triangle   */
    if ( |P| = 0 ) then
        return e_{0,0} /* T is a triangle consisting of one element */
    if ( ρ_1 = 0) then
        return GetCost(T_1, P') + GetCost(T_2, P')
            /* where P' is P with ρ_1 deleted */
    if ( ρ_1 = 1) then
        return GetCost(T_0, P') + GetCost(T_2, P')
    if ( ρ_1 = 2) then
        return GetCost(T_0, P') + GetCost(T_1, P')
```

Figure 3: Recursive algorithm to find the cost for a given polarity.

Theorem 1. *Given function $f(x_1, x_2, ..., x_n)$, Algorithm 1 finds the cost for a given polarity P.*

Proof. The proof is by the induction.

1. If $n = 1$ write $f(x_1) = \bar{x_1} f(0) \oplus x_1 f(1)$ which gives us polarity $P = (2)$, or $f(x_1) = f(0) \oplus x_1(f(0) \oplus f(1))$ which gives us polarity $P = (1)$, or $f(x_1) = f(1) \oplus \bar{x_1}(f(0) \oplus f(1))$ which gives us polarity $P = (1)$. In this simple case the cost of the polarity $P = (2)$ is $f(0) + f(1)$, the cost of the polarity $P = (1)$ is $f(0) + (f(0) \oplus f(1))$and the cost of the polarity $P = (0)$ is $f(1) + (f(0) \oplus f(1))$. The transeunt triangle for the given function will look like:

$$f(0) \oplus f(1)$$

$$f(0) \qquad f(1)$$

Note, that $T_0 = f(0)$ is the transeunt triangle for $f(0)$, $T_1 = f(1)$ is the transeunt triangle for $f(1)$ and $T_2 = f(0) \oplus f(1)$ is the transeunt triangle for $f(0) \oplus f(1)$. By the algorithm, if $\rho_1 = 0$ we should consider T_1 and T_2. These triangles are elementary, so we take the sum of their elements. It is equal to the $f(1) + (f(0) \oplus f(1))$ which is actually the cost for the given polarity as it was shown before. If $\rho_1 = 1$ consider T_0 and T_2 and conclude that the sum of their elements is equal to the cost of polarity $P = (1)$ as it was shown. And the same goes for $P = (2)$.

2. Assume the theorem holds for $n = k - 1$. We can decompose $f(x_1, x_2, ..., x_k)$ as follows:

$$\begin{aligned} f(x_1, x_2, ..., x_k) &= \bar{x_1} f(0, x_2, ..., x_k) \oplus x_1 f(1, x_2, ..., x_k) \\ &= f(0, x_2, ..., x_k) \oplus x_1 (f(0, x_2, ..., x_k) \oplus f(1, x_2, ..., x_k)) \\ &= f(1, x_2, ..., x_k) \oplus \bar{x_1} (f(0, x_2, ..., x_k) \oplus f(1, x_2, ..., x_k)) \end{aligned}$$

From this formula we conclude that the final cost for every polarity $P = (2, \rho_2, ..., \rho_k)$ will be the sum of the costs of polarity $P' = (\rho_2, \rho_3 ..., \rho_k)$ for functions $f(0, x_2, ..., x_k)$ and $f(1, x_2, ..., x_k)$. By the algorithm we had to consider triangles T_0 and T_1, which are transeunt triangles for $f(0, x_2, ..., x_k)$ and $f(1, x_2, ..., x_k)$ correspondingly. By induction, the algorithm works for functions $f(0, x_2, ..., x_k)$ and $f(1, x_2, ..., x_k)$ as functions from $(k-1)$ variables and the cost for the given polarity P' will be the sum of given by the algorithm elementary triangles inside T_0 and T_1. But this sum is equal to the sum of all given by the algorithm elementary triangles of transeunt triangle for $f(x_1, x_2, ..., x_k)$ because at the first step we took T_0 and T_1 due to the algorithm. And, finally, the number which we obtained can be considered as the sum of costs of polarity P' for $f(0, x_2, ..., x_k)$ and $f(1, x_2, ..., x_k)$ which is the cost of polarity $P = (2, \rho_2, ..., \rho_k)$ for $f(x_1, x_2, ..., x_k)$ as it was shown before.

For cases $P = (1, \rho_2, ..., \rho_k)$ and $P = (0, \rho_2, ..., \rho_k)$ the induction step is similar to the previous proof. ■

**** Text below was changed **** **** In my opinion, there should be a black box end marker for examples, as well as proofs. ****

Example 2. Given the function $f(x_1, x_2, x_3)$ whose truth vector is $(0, 0, 1, 0, 1, 0, 1, 1)$ find the cost of polarity $P = (1, 2, 0)$. First, we build the transeunt triangle, and then, we identify the MPRM coefficients associated with the given polarity according to Algorithm 1. The coefficients identified are shown in Fig. 4 in black. Finally, we sum the coefficients in a cost of 4.

In the computer program implementation of the algorithm, we only consider a transeunt triangle that has 3^n elements. We call this a *restricted transeunt triangle (RTT)*. The construction of a RTT proceeds as follows: enumerate elements $e_{i,j}$ of transeunt triangle. The ternary number $(\gamma_1, \gamma_2, ..., \gamma_n)$ which corresponds the element $e_{i,j}$ of RTT for the function $f(x_1, x_2, ..., x_n)$ is built as follows: first, γ_1 is k, if $e_{i,j} \in T_k$, where $k \in \{0, 1, 2\}$. Second, γ_2, is zero if $e_{i,j}$ lies in the bottom-left triangle, one if $e_{i,j}$ lies in the bottom-right triangle and two if $e_{i,j}$ lies in the top triangle inside the chosen T_k. Continue this operation until all values are obtained. For example, $e_{7,3}$ in (Fig. 5) has ternary number $(0, 1, 2)$, because it is in the second

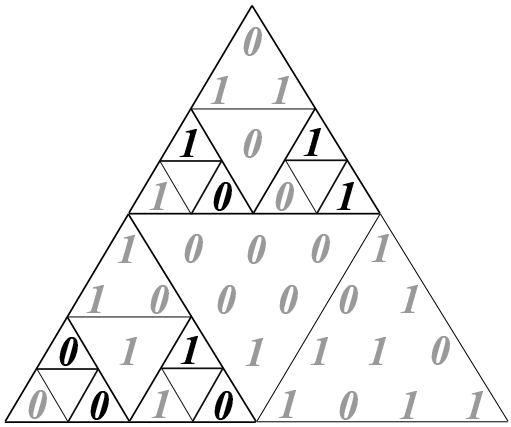

Figure 4: transeunt triangle for the example

smallest triangle, which is inside bigger triangle with number 1, which in its turn, lies in the bigger triangle number 0.

Lemma 3. If the element $e_{i,j}$ with the ternary number $(\gamma_1, \gamma_2, ..., \gamma_n)$ has $\gamma_k = 2$ for $k \in \{1, 2, ..., n\}$, then it is equal to the exclusive or of two elements $e_{s,t}$ and $e_{u,v}$ of RTT with ternary numbers $(\gamma_1, ..., \gamma_{k-1}, 0, \gamma_{k+1}, ..., \gamma_n)$ and $(\gamma_1, ..., \gamma_{k-1}, 1, \gamma_{k+1}, ..., \gamma_n)$ respectively.

Proof. Notice, that by the construction of a ternary number for $e_{i,j}$ the change of k-th position from 2 to 0 gives us element $e_{s,t}$, where $s = i + 2^{n-k}$ and $t = j$. For the same reason the change of k-th position of $e_{i,j}$ from 2 to 1 gives us element $e_{u,v}$, where $u = i + 2^{n-k}$ and $v = j + 2^{n-k}$. Now we can apply Lemma 1 to prove the statement. ■

The fact that all elements with ternary numbers $(\gamma_1, \gamma_2, ..., \gamma_n)$ where $\gamma_i \in \{0, 1\}$ lie in lexicographic order in the bottom of the RTT together with the definition of transeunt triangle gives us the following lemma.

Lemma 4. All elements with ternary numbers $(\gamma_1, \gamma_2, ..., \gamma_n)$ where $\gamma_i \in \{0, 1\}$ are equal to $f(\gamma_1, \gamma_2, ..., \gamma_n)$.

Given the truth vector of a function (the bottom row) we need $3^n - 2^n$ "exclusive or" operations to build the remaining elements in the RTT. That is, there are 3^n elements in the RTT, of which 2^n are the truth vector. Next, to find the best mixed polarity Reed-Muller expression in our computer program implementation of the algorithm, we interpret switching variables $0, 1$ as integers and do the following: first, take 3^{n-1} 3-element triangles of RTT. Each of them has elements $e_{i,j}, e_{i+1,j}, e_{i+1,j+1}$ and we

1. set $e_{i,j}$ to $e_{i+1,j} + e_{i+1,j+1}$,
2. set $e_{i+1,j}$ to $e_{i,j} + e_{i+1,j+1}$,
3. set $e_{i+1,j+1}$ to $e_{i,j} + e_{i+1,j}$.

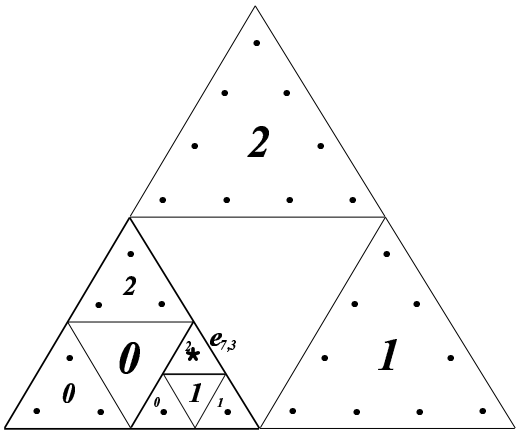

Figure 5: The example of building the ternary number

Note: the operations are performed simultaneously. To do this for the single 3-element triangle we need 3 addition operations. For the first step we need $3 * 3^{n-1} = 3^n$ addition operations. In general, for the k-th step we take 3^{n-k} of 3^k-element triangles (decomposition of T into smaller triangles is shown on the Fig. 7) and, within each of them by the same rules, obtain: element-wise sum of $T_{\beta_1,\beta_2,...,\beta_{k-1},0}$ and $T_{\beta_1,\beta_2,...,\beta_{k-1},1}$ in $T_{\beta_1,\beta_2,...,\beta_{k-1},2}$, element-wise sum of $T_{\beta_1,\beta_2,...,\beta_{k-1},1}$ and $T_{\beta_1,\beta_2,...,\beta_{k-1},2}$ in $T_{\beta_1,\beta_2,...,\beta_{k-1},0}$ and element-wise sum of $T_{\beta_1,\beta_2,...,\beta_{k-1},2}$ and $T_{\beta_1,\beta_2,...,\beta_{k-1},0}$ in $T_{\beta_1,\beta_2,...,\beta_{k-1},1}$, where $\beta_i \in \{0,1,2\}$ for $i \in \{1,2,...,k-1\}$. To do this we need $3^k * 3^{n-k} = 3^{k+n-k} = 3^n$ addition operations. After all the steps (there will be (n-1) of them) are done, costs of all 3^n different mixed polarities are represented by the elements in RTT. It is easy to see, that for polarity $P = (\rho_1, \rho_2, ..., \rho_n)$ it's cost is in the cell with ternary number $(\gamma_1, \gamma_2, ..., \gamma_n)$. To find the minimal cost we may apply any known method which finds the minimal element of the array.

The final complexity will be:

- $3^n - 2^n$ binary additions to build RTT,
- $(n-1)3^n$ natural addition operations to find all costs,
- $3^n - 1$ comparisons to find the minimal cost.

Needed memory is:

- 3^n integers plus a few integer variables for intermediate assignments (we used 3).

4.2 Multiple Output Functions

Our method also works for multiple output functions. The truth vector is given as a list of words rather than bits. To build transeunt triangle the words are "exclusive or"-ed bit by bit. Before the cost is calculated, all entries in transeunt triangle are set to 1 if the word contains at least one nonzero bit and are set to 0 otherwise. The cost of the best polarity is calculated as before.

Algorithm 2

CalcAllCosts(T, n)
    /* Calculate the cost for all polarities */
    /* for a function T given as transeunt triangle */
    /* Note: elements in T will contain costs */
    if ($n > 1$) then
        CalcAllCosts(T_0, $n-1$)
        CalcAllCosts(T_1, $n-1$)
        CalcAllCosts(T_2, $n-1$)
    end if
    $T'_0 = T_1 + T_2$ /* element by element addition */
    $T'_1 = T_0 + T_2$
    $T'_2 = T_0 + T_1$
    $T = T'_0, T'_1, T'_2$ /* replace T with the 3 sub-triangles */
end CalcAllCosts

Figure 6: Recursive algorithm to find costs for all mixed polarities for a given function.

5 Experimental Results

We have implemented the algorithm using the C programming language. The program was applied to several benchmark functions running on a Sun Enterprise 250 with two 400Mhz Ultra Sparc II processors and 1GB of main memory. The results are summarized in Table 5. In the first column, *name*, denotes the name of the function (from NCMC benchmarks *** Maybe, this should be MCNC.***). Note: *co14* is a symmetric function where the output is one if exactly 1 input variable is 1. In general *co***n** is an n variable symmetric function whose output is 1 if exactly one input is 1. The number of inputs (outputs) are given in *in (out)*. The minimum number of terms required for a fixed (mixed) polarity Reed Muller expression are shown in *cost fixed (cost mixed)*. *OFDD time* denotes the cpu time in seconds for the OFDD implementation reported in [3] (run on a HP Apollo series 700 workstation). *RTT time* gives the cpu time in seconds for our algorithm. Our results compare favorably with previous implementations. We are able to minimize function with up to 18 variables. Memory requirements make it impossible to minimize functions with more than 18 variables.

Green's [4] algorithm (which has the same complexity) was never implemented. We claim that our method is easier to program. Drechsler, Theobald and Becker's [3] implementation is significantly slower than our program and their algorithm considers only fixed polarity expressions. Derchsler *et al.* don't provide a complexity analysis for their algorithm. However, the execution time depends on the function as well as on the number of variables, whereas our algorithm depends only on the number of variables.

Lui and Muzio [5] describe an algorithm with space complexity of $O(2^n)$ and time complexity of $O(6^n)$.

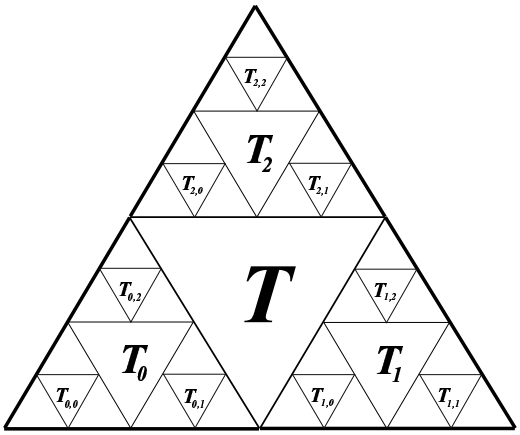

Figure 7: Decomposition of T into smaller triangles

6 Conclusion

We presented a new algorithm to minimize fixed and mixed polarity Reed Muller expressions. The algorithm is easy to understand and to program. The results compare favorably to previous algorithms, although we have not been able to reduce the complexity of $O(3^n)$ both in storage and execution time.

An algorithm involving transeunt triangles has been devised to find the best polarity for totally symmetric functions in $O(n^3)$ operations and $O(n^2)$ storage space [**?**]. Some of the dramatic reduction in complexity can be achieved for partially symmetric functions. This area is currently under investigation.

References

[1] M. Cohn. Inconsistent canonical forms of switching functions. *IRE Transactions of Electronic Computers*, EC-11:284–285, 1962.

[2] M. J. Davio, P. Deschamps, and A. Thayse. Discrete and switching functions. *McGraw-Hill Int. Book Co.*, 1978.

[3] R. Drechsler, M. Theobald, and B. Becker. Fast OFDD-based minimization of fixed polarity Reed-Muller expressions. *IEEE Transactions on Computers*, 45:1294–1299, Nov. 1996.

[4] D. H. Green. Reed-Muller canonical forms with mixed polarity and their manipulations. *IEE Proceedings, Part E.*, 137:103–113, Jan. 1990.

[5] P. K. Lui and J. C. Muzio. Boolean matrix transforms for the minimization of modulo-2 canonical expressions. *IEEE Transactions on Computers*, 41:342–347, Mar. 1992.

name	*in*	*out*	*cost fixed*	*cost mixed*	*OFDD time (secs.)*	*RTT time (secs.)*
9sym	9	1	173	173	8.1	< 0.001
add6	12	7	132	132	295.1	0.17
co14	14	1	14	14	448.4	1.99
co15	15	1	15	15		6.68
co16	16	1	16	16		20.41
co17	17	1	17	17		64.51
co18	18	1	18	18		214.11
dist	8	5	185	157	12.5	< 0.001
gary	15	11	349	242	16216.1	6.44
mixex3	14	14	3536	1421		2.02
rd53	5	3	20	20	0.5	< 0.001
rd73	7	3	63	63	2.3	< 0.001
rd84	8	4	107	107	5.5	< 0.001
root	8	5	118	83	8.8	< 0.001
sao2	10	4	100	76	8.8	0.02
table3	14	14	1845	407		2.01
table5	17	15	2458	559		65.88
tial	14	8	3683	2438	8480.4	2.04

Table 1: Experimental results

[6] A. Mukhopadhyay and G. Schmitz. Minimization of exclusive-or and logical-equivalence switching circuits. *IEEE Trans. on Computers*, pages 132–140, 1970.

[7] S. M. Reddy. Easily testable realisations for logic functions. *IEEE Trans. on Computers*, pages 1183–1188, 1972.

[8] T. Sasao and Ph. W. Besslich. On the complexity of mod-2 sum pla's. *IEEE Trans. on Computers*, C-29:262–266, Feb. 1990.

[9] J. Saul. Logic synthesis for arithmetic circuits using the Reed-Muller representation. *Proc. Euro. Conf. Design Automation*, pages 109–113, 1992.

[10] I. Schafer and M. A. Perkowski. Multiple-valued input generalized Reed-Muller forms. *Proc. of the Inter. Symp. on Multiple-Valued Logic*, pages 40–48, 1991.

[11] V. P. Suprun. Fixed polarity Reed-Muller expressions of symmetric Boolean functions. *Proc. IFIP WG 10.5 Workshop on Application of the Reed-Muller Expansions in Circuit Design*, pages 246–249, 1995.

[12] Chien-Chang Tsai and M. Marek-Sadowska. Boolean Functions Classification via Fixed Polarity Reed-Muller Forms. *IEEE Trans. on Computers*, C-46(2):173–186, Feb. 1997.

[13] T.Sasao. Switching theory for logic synthesis. *Kluwer Academic Publishers, Norwell, MA*, 1999.