2002-08-06

A neighborhood decoupling algorithm for truncated sum minimization

Yang, Chyan; Wang, Yao-Ming

IEEE

Yang, Chyan, and Y-M. Wang. "A neighborhood decoupling algorithm for truncated sum minimization." Proceedings of the Twentieth International Symposium on Multiple-Valued Logic. IEEE, 1990.
http://hdl.handle.net/10945/61409

A Neighborhood Decoupling Algorithm for Truncated Sum Minimization*

Chyan Yang
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California, 93943-5004
yang @ cs.nps.navy.mil 408-646-2266

Yao-Ming Wang
Department of Electrical Engineering
Chung-Cheng Institute of Technology
Ta-Hsi, Taoyuan, Taiwan, 33509, R.O.C.

Abstract

There has been considerable interest in heuristic method for minimizing multiple-valued logic functions because exact methods are intractable. This paper describes a new heuristic, called the neighborhood decoupling (ND) algorithm. It first selects a minterm and then selects an implicant, a two step process employed in previous heuristics, e.g., Besslich [2] and Dueck and Miller [4]. The approach taken here more closely resembles the Dueck and Miller heuristic; however, it makes more efficient use of minterms truncated to the highest logic value. The ND-algorithm was developed in conjunction with HAMLET [12], a computer software created at the Naval Postgraduate School for the purpose of designing heuristics for multiple-valued logic minimization. In this paper, we present the algorithm, discuss the implementation, show that it performs consistently better than others and explain the reason for its improved performance.

1 Introduction

The minimization of MVL programmable logic arrays (PLA) is an important and interesting problem. Several heuristics have been developed to the multiple valued logic minimization problems and each claims some advantage in specific examples but none of them is consistently better than the others [2, 4, 5, 8, 9, 10, 12]. Heuristic methods are of interest because exact minimization methods are extremely time-consuming. With the computer software developed at Naval Postgraduate School called HAMLET [12] users can easily extend their own heuristics. For example, the neighborhood decoupling (ND) algorithm is built as one independent option of HAMLET. This article reexamines the methods used in HAMLET and reports the new method, ND-algorithm, which on the average performs better than others.

Generally speaking, the heuristic approaches in HAMLET can all be classified as ad hoc or greedy algorithms. We can abstract these algorithms as follows.

*This research is supported in part by the Naval Research Laboratory through a Naval Postgraduate School Block Funding Grant in the fiscal year 1989-1990.

```
/* ****************************************
input: let the M be the set of minterms of an function f;
output: the minimized sum of product, S, of the original
function;
**************************************** */
{       S ← φ.
        While (M ≠ φ) do {
                pick one minterm α from M;
                find an implicant I_α which covers α;
                S ← I_α ∪ S;
                remove α from M;
              }
}
```

In essence, each algorithm differs from the others in the manner of picking the minterms (α) and finding the implicants (I_α). The Pomper and Armstrong algorithm [8] picks α randomly (as long as α is in M) and finds I_α randomly (as long as I_α covers α) and searches for the "largest" implicant, I_α, which covers the minterm (α). This is a greedy algorithm in nature: by choosing the "largest" implicant it hopes that the chosen implicant includes the maximum number of remaining minterms. Dueck and Miller algorithm picks α from M if the α is the most "isolated" minterm and then finds the I_α which directly covers the α such that the relative break count is minimum [4, 5]. The neighborhood decoupling algorithm is an improvement to the Dueck and Miller algorithm [4, 5, 9, 11] with revised decision rules for making selections of minterms and implicants.

We present the definitions that are used in this paper in Section 2. We then present the neighborhood decoupling algorithm in Section 3. Section 4 and 5 discuss the performance comparisons of the neighborhood decoupling algorithm.

2 Notations and Definitions

We review definitions for the truncated sum operation in Section 2.1, most of which are defined in previous literature [4, 5, 10, 11]. We then present the definitions used in the ND-algorithm in Section 2.2. Certain definitions in Section 2.2 are variations of those defined by Dueck and Miller (DM) [4]. The variations from DM are the sources of the improvement which ND-algorithm benefits.

2.1 Truncated Sum

Let $\mathcal{X} = \{ x_1, x_2, ..., x_n \}$ be a set of n input variables where

x_i takes on values from $\mathcal{R} = \{0, 1, ..., r-1\}$. An n-variable r-valued function f is a mapping $f : \mathcal{R}^n \longrightarrow \mathcal{R} \cup \{r\}$, where r is the don't care value [10]. The MIN function is defined as $MIN(x_1, x_2) = x_1 x_2 = minimum(x_1, x_2)$[11]. There are two definitions of *literal* in the literature: interval (window or contiguous) [3, 4, 5] and set (or discrete) [4, 5]. Although the interval version is a special case of the set version, we use the literal version since there is a physical realization of the interval literal defined [3, 6]. The interval literal of a variable x is defined [3] as:

$$ {}^{a}x^{b} = \begin{cases} r-1 & \text{if } a \le x \le b \\ 0 & \text{otherwise.} \end{cases} $$

From now on the term "literal" will mean interval literal. The truncated sum (TSUM) operation [3] is

$$ TSUM(x_1, x_2) = x_1 + x_2 = minimum(x_1 \; plus \; x_2, r-1). $$

Note that the notation '+' is the addition truncated to the highest logic value $(r-1)$ while *plus* denotes normal addition. Other than the truncation the TSUM operation is the same as normal addition and TSUM obeys the associative and commutative rules. If $\mathcal{R} = \{0, 1, 2, 3\}$ then examples of TSUM are TSUM(1,2) = 3 and TSUM(2, 2) = 3. The use of TSUM in this paper is inspired by the fact that the CCD implementation supports TSUM naturally [3,6].

Example 1: For example, ${}^{1}x_1^3$ is a literal and takes value of 3 in a 4-valued function but the function $2\,{}^{1}x_1^3$ takes a value of 2 due to the definition of MIN. Similarily, the product term $2\,{}^{3}x_1^3\,{}^{0}x_2^0$ takes a value of 2. The constant or coefficient c, in a product term effectively scale the term. □

A product term p is the MIN of one nonzero constant $c \in \mathcal{R}$, and one or more literal functions. In general, a product term is defined as:

$$ p \equiv c\,{}^{i_1}x_1^{j_1}\,{}^{i_2}x_2^{j_2} \ldots {}^{i_n}x_n^{j_n} \begin{cases} i_k \le j_k \\ i_k, j_k \in \mathcal{R}; \quad 1 \le k \le n. \end{cases} $$

For each variable x_i we say the window size of the literal ${}^{i_k}x_i^{j_k}$ is $j_k - i_k + 1$. We use the terms *product term* and *implicant* interchangeably in this article. A minterm α is of the form $c\,{}^{a_1}x_1^{a_1}\,{}^{a_2}x_2^{a_2}\,{}^{a_3}x_3^{a_3} \ldots {}^{a_n}x_n^{a_n}$ where $a_i \in \mathcal{R}$ and constant $c \in \mathcal{R} - \{\,0\,\}$. We say the coordinate of α is $< a_1, a_2, \ldots a_n >$. We denote the value of minterm α, $g(\alpha)$, as the nonzero constant c. Obviously, the window size of each variable of a minterm α is 1. If the product term is defined as above, the *value* of product term is $g(p) = c$. Given two product terms p_1, p_2 with values c_1, c_2 respectively we say the value of $TSUM(p_1, p_2)$ is equal to $TSUM(c_1, c_2)$. A sum-of-products expression is $p_1 + p_2 + \ldots + p_N$ for some integer N, where p_i is a product term. A product term $p = c\,{}^{i_1}x_1^{j_1}\,{}^{i_2}x_2^{j_2} \ldots {}^{i_n}x_n^{j_n}$ consists of $\prod_{k=1}^{n}(j_k - i_k + 1)$ minterms each with value c.

We say p generates or covers those minterms. Given a product term p, the set of minterms generated from p is denoted by MS_p. In other words, a minterm α is "covered" by an implicant iff $a_k \in [i_k, j_k]$, $i_k, j_k \in \mathcal{R}$; for all $1 \le k \le n$. If the number of elements in MS_{p_1} is greater than that in MS_{p_2}, we say p_1 covers larger area than p_2. Given a minterm α generated from the original function to be minimized if $g(\alpha) = r - 1$ then α is a saturated minterm. Let SAT be the set of all saturated minterms. We mark a saturated minterm with a dot in our figures. Given a function f, the set of minterms generated from its product terms is denoted by MS_f. If more than one product term in f generates the same minterm then that minterm appears in MS_f only once. We can express the value of f by representing values of MS_f as illustrated in the example below.

Example 2: If the input function f to be minimized is expressed as follows,

$$ 3\,{}^{0}x_1^3\,{}^{1}x_2^1 + 2\,{}^{1}x_1^2\,{}^{0}x_2^0 + 3\,{}^{1}x_1^1\,{}^{2}x_2^3 + 2\,{}^{2}x_1^2\,{}^{2}x_2^3 + 1\,{}^{2}x_1^2\,{}^{3}x_2^3 + 1\,{}^{0}x_1^0\,{}^{2}x_2^2, $$

the MS_f can be represented as 11 minterms in Figure 1.

Observation 1 *Given a minterm α the maximum number of implicants that covers α is $O(r^{2n})$.*

Proof: Consider a variable (axis) x_i of α. Any implicant (I_α) that covers α may have window size w such that $1 \le w \le r$. With a window size w we may have w implicants that covers α. In the worst case, the minterm α is at the middle point of an axis, i.e., $\lceil r/2 \rceil$. Consider the window bounded by $[i, j]$ with $i \le \frac{r}{2}$ and $j \ge \frac{r}{2}$ for a given axis, we have $\frac{r}{2} \times \frac{r}{2} = \frac{r^2}{4}$ possible implicants that cover α. Over the entire n-dimensional space, we have $(\frac{r^2}{4})^n = O(r^{2n})$ [1] □.

2.2 Definitions Used in ND-algorithm

Let α and β are minterms with coordinates $< a_1, a_2, \ldots a_n >$ and $< b_1, b_2, \ldots b_n >$ respectively. If for all i we have $a_i = b_i$ except one position j such that $|a_j - b_j| = 1$ we say that α and β are direct neighbors. Given a minterm α, we use $N(\alpha)$ to denote the set of its direct neighbors. To emphasize the requirement of $|a_j - b_j| = 1$ the ND-algorithm uses the term direct neighbor. Two minterms α and β are directional neighbors in the direction x_j if for one $j \in [1, n]$, $a_j \ne b_j$ and $a_i = b_i$ for all other $i \in [1, n]$ $(i \ne j)$. When $b_j > a_j$ we say that β is in the positive direction of α while when $b_j < a_j$ we say that β is in the negative direction of α. Therefore, if β is a direct neighbor of α then β is a directional neighbor of α in the direction of x_i for some $i \in [1, n]$. The maximum number of direct neighbors of a given minterm is $2n$, where n is the number of variables.

The connected minterms to be defined is similar to the *expandable adjacency* in Dueck's thesis [6, p.50]. It is defined recursively and includes SAT and don't cares. This definition is important and reflects the transitive property of the "connected" relationship. It is interesting to note that the recursive definition of two connected minterms is not *commutative*. Given a minterm α and a minterm β, then we say β is a connected minterm of α, if

(1) β is a direct neighbor of α and either $g(\beta) \le g(\alpha)$ or α is don't care.

(2) β is a directional neighbor of α on a direction x_i and its direct neighbor is connected to α *and* either $g(\beta) \le g(\alpha)$ or α is don't care. □

Notice that in the condition $g(\beta) \leq g(\alpha)$ it is possible that $\alpha \in SAT$. The connected minterm count of minterm α, CMC_α, is the number of minterms that are connected to minterm α. The expandable directional count of minterm α, EDC_α, is the number of directions in which α has connected minterms. Because for each x_i have 2 directions (positive and negative) we observe that $0 \leq EDC_\alpha \leq 2n$. The EDC_α is similar to the idea of DM's DEA_α, the number of directions of expandable adjacency of α but they are not the same, since, in the EDC, the positive and negative directions are treated as distinct. Besides, the interpretation of *expandable* and *connected* are not the same. Note also, the extendible minterm (see [5], p.122) is defined differently from connected minterms.

The clustering factor relative to a minterm α is defined as $CF_\alpha = EDC_\alpha(r-1) + CMC_\alpha$. This is a measure of the weight of all connected minterms relative to α. The $(r-1)$ factor is the range, or maximum possible number of minterms, in a direction x_i. The weight of $(r-1)$ on the EDC emphasizes the importance of the directions similar to DCM^w [5]. Conceptually, the clustering factor is the inverse of the isolation factor [4, 5] but its value is not the inverse of the isolation factor in most cases because of the different interpretation of each term.

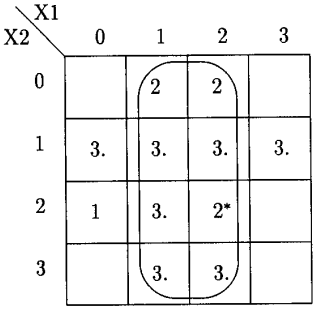

Figure 1: Map for Example 2, 3, 4; Step 1 of Table 3

Example 3: In Figure 1 the minterm $2\ {}^2x_1{}^2\,{}^2x_2{}^2$ (the minterm with * sign) has no connected minterms nor expandable directional neighbors, i.e., its CMC and EDC values are 0. However, in Figure 2 the minterm $\alpha = 3\ {}^0x_1^0\,{}^1x_2^1$ is one of the eight minterms and $CMC_\alpha = 4$, $EDC_\alpha = 2$. The clustering factors of all minterms in Figure 2 is listed in Table 1. □

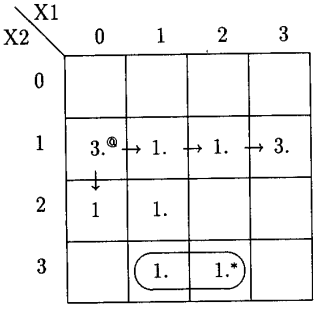

Figure 2: Map for Example 3, 4; Step 2 of Table 3

Minterm	$3\ {}^0x_1{}^0\,{}^1x_2{}^1$	$1\ {}^1x_1{}^1\,{}^1x_2{}^1$	$1\ {}^2x_1{}^2\,{}^1x_2{}^1$	$3\ {}^3x_1{}^3\,{}^1x_2{}^1$
CF	10	14	9	6
Minterm	$1\ {}^0x_1{}^0\,{}^2x_2{}^2$	$1\ {}^1x_1{}^1\,{}^2x_2{}^2$	$1\ {}^1x_1{}^1\,{}^3x_2{}^3$	$1\ {}^2x_1{}^2\,{}^3x_2{}^3$
CF	4	12	9	4

Table 1: CFs for all minterms in Figure 2

3 The ND-Algorithm

As stated in Section 1, the neighborhood decoupling algorithm is an improvement to the Dueck and Miller's method. First, the most isolated minterm is chosen using the algorithm M described below. The most isolated minterms in general are different from Dueck and Miller's (DM) method due to different decision rules. In the DM-algorithm, an "isolated" minterm is selected as follows: *starting from the minimum value*, for each minterm search for expandable adjacency minterms and isolation factors (see [6], p.54, step 1, 2, 3.) The minterm with the maximum isolation factor is selected. In the ND-algorithm, however, it *does not* start from the minterm with minimum value. Instead, for each minterm it looks for its "connected minterms" (differs from adjacency as described in Section 2) and clustering factor (differs from the inverse of isolation factor, see Section 2.) The minterm with the smallest clustering factor is selected. If there is a tie, the ND-algorithm will select the minterm that is evaluated last (the ND-algorithm varies the X_i before X_j when $i < j$.) In the DM-algorithm, ties are broken arbitrarily (see [6], p.54.) For example, in Figure 3, the ND-algorithm selects $2\ {}^3x_1{}^3\,{}^2x_2{}^2$ as most isolated minterm. However, the DM-algorithm will not select this since it starts from the minterm with the smallest value (1).

Second, from all implicants which cover the most isolated minterm we choose the one that is not strongly "coupled" with its neighbors. This decoupling process is based on observations that if we choose that specific implicant then we may minimize the negative impact for future minterm selections as well as implicant selections. We will explain this idea further in Section 3.2. In the algorithm below, f denotes the function to be minimized.

{ /** The ND-Algorithm; SS = Solution Set **/
$SS \leftarrow \phi$;
$WS = MS_f = \{\alpha | \alpha$ is generated by the function f;
if $\alpha \in SAT$ then mark its coordinate $\}$.
While $WS \neq \phi$ **do** {
1. Use algorithm M (see Section 3.1) to select a minterm α from the WS.
2. Use algorithm N (see Section 3.2) to select an implicant I_α that covers α; $SS \leftarrow SS \cup I_\alpha$.
3. $\forall \beta \in I_\alpha$ **do** {
compute $g(\beta) \leftarrow g(\beta) - g(\alpha)$.
if β is originally marked and $g(\beta) = 0$ **then**
$g(\beta) \leftarrow r$. }
4. Update WS. } /* end While */
}

X2 \ X1	0	1	2	3
0				
1	1	1	1	
2	2	2	2	2
3	2	2	1	

Figure 3: $2^3 x_1{}^{32} x_2{}^2$ is the most isolated minterm in ND

3.1 Algorithm M: Minterm Selection

We first compute the clustering factors for all minterms and the clustering factor is computed in the order of coordinates (x_i). For example, the minterm $2\ {}^1x_1^1\,{}^1x_2^1$ is evaluated earlier than the minterm $2\ {}^1x_1^1\,{}^2x_2^2$. The algorithm M is simply described as follows.

/ Algorithm M *************************************
*α: the chosen minterm from algorithm M; **/*

1. $\forall \alpha_k \in WS$ compute the corresponding CF_{α_k}
2. select the minterm α that has the smallest clustering factor. If there is a tie, the last one gets evaluated is chosen.

3.2 Algorithm N: Implicant Selection

The purpose of algorithm N is to choose the most "isolated" implicant (I_α) and update the working set WS. It computes the neighborhood relative count (NRC) for all implicants that cover the minterm α. The implicant with the smallest NRC is chosen. The NRC is a measure of the coupling strength of an implicant with its neighbors. To select an implicant is equivalent to breaking the coupling between that implicant with its neighbors. The candidate implicant should have the smallest coupling strength with its neighbors. Therefore, we choose the most "isolated" implicant, i.e., lowest NRC. If there is a tie in selecting the I_α, we choose the one which covers the largest area. If a tie still exist, i.e., two or more implicants of same size cover the α then we select the last one get examined (evaluated) in HAMLET [12]. Note that, the NRC defined here is similar to RBC (relative break count) [4] or BCR (break count reduction) [5] but is different in three ways: (1) in addition to +1 and -1 counting, the ND-algorithm uses +2 and -2 weightings for penalty (+2) or incentive (-2), i.e., an idea of Besslich [2] is incorporated; (2) in DCM^w the order of BCR evaluation is based on the index of a variable (X_1 varies before X_2) but the ND-algorithm evaluates the NRC in the order of the implicant sizes, (3) instead of choosing the minimum RBC [4] or maximum BRC [5] (the DCM^w often has ties [5, p.134] and the ties are broken by choosing the last implicant that is evaluated) the ND-algorithm selects the implicant with the smallest NRC that covers the largest area. With a refined weighting scheme there are fewer ties with the ND-algorithm. Our experiences indicate that the performance of the ND-algorithm is very sensitive to the weighting factors.

We now explain how we compute the NRC for a given implicant. We first initialize the NRC to zero. We then check all neighboring minterms of the implicant and increment or decrement its NRC by the following rules: if the coupling strength between covered and uncovered area is weak (good for further decoupling) we decrement NRC, otherwise increment NRC.

/* Algorithm N ************************************

α: the chosen minterm from algorithm M;
M: the set of minterms which was covered (generated) by the chosen implicant (I_α).
$N(\beta)$: the set of direct neighbors of minterm β.

*** */

{ $NRC \leftarrow 0$;
$\forall \beta \in M$ **and** $\beta \neq \alpha$ **do** {
  if($g(\beta) - g(\alpha) \leq 0$) **then** $NRC \leftarrow NRC - 1$; }
$\forall \beta \in M$ **and** $\forall \gamma \in N(\beta)$ **do** {
  if ($\gamma \notin M$ **and** $\gamma \neq 0$ **and** ($\gamma \notin SAT$ **or** $\beta \notin SAT$))
  then {
    if ($g(\beta) - g(\alpha) > g(\gamma)$) **then** {
      if ($\gamma \in SAT$) **then** $NRC \leftarrow NRC - 1$;
        else $NRC \leftarrow NRC + 2$;}
    if ($g(\beta) - g(\alpha) < g(\gamma)$) **then** {
      if ($g(\beta) = g(\gamma)$) **then** $NRC \leftarrow NRC + 2$;
      if ($\gamma \in SAT$ **and** $g(\gamma) - g(\beta) < 0$) **then**
        $NRC \leftarrow NRC + 2$;
      else {
      if ($g(\beta) > g(\alpha)$ **and** $g(\beta) \neq g(\gamma)$) **then**
        if($\beta \in SAT$) **then** $NRC \leftarrow NRC - 1$;
        else $NRC \leftarrow NRC + 2$;
          } /* end else */
      } /* end if */
    if ($g(\beta) - g(\alpha) = g(\gamma)$ **and** $\gamma \notin SAT$) **then**
        $NRC \leftarrow NRC - 1$;
    } /* end then */
  } /* end do */
If (M = { α } **and** $NRC = 0$) **then** $NRC \leftarrow 1$;
}

Example 4: It is instructive to consider an example of the steps of the ND-algorithm by using the function f shown in Example 2 (two-variable four-valued). The working set, WS, is initialized to MS_f and is represented in Figure 1. The clustering factors of all minterms in WS are calculated (see example 3 for computation). The smallest CF comes from minterm $2\ {}^2x_1^2\,{}^2x_2^2$ and therefore algorithm M will select $\alpha = 2\ {}^2x_1^2\,{}^2x_2^2$. We compute the NRC_I for each implicant I which covers α using algorithm N. Since implicant $2\ {}^1x_1^2\,{}^0x_2^3$ has the smallest NRC (-3) we select it as the first implicant to be in the solution set (SS). Table 3, together with Figures 2, 4, 5 show the steps of choosing successive implicants. The * sign in each Figure indicates the most isolated minterm while a

circled implicant is the most isolated implicant. Suppose we have chosen two implicants and the working set arrives at the configuration of Figure 4. The minterm $1\ ^0x_1{}^0\ ^2x_2{}^2$ is selected since it has the smallest CF. There are four implicants cover the minterm $1\ ^0x_1{}^0\ ^2x_2{}^2$ and their NRC values are listed in Table 2. The implicant $1\ ^0x_1{}^1\ ^1x_2{}^2$ is chosen since it has the smallest NRC. Having updated the working set and having added $1\ ^0x_1{}^1\ ^1x_2{}^2$ to the solution set we have the new map in Figure 5.

The final minimized result, g, is expressed as:

$$g = 2\ ^1x_1{}^2\ ^0x_2{}^3 + 1\ ^1x_1{}^2\ ^3x_2{}^3 + 1\ ^0x_1{}^1\ ^1x_2{}^2 + 3\ ^0x_1{}^3\ ^1x_2{}^1$$

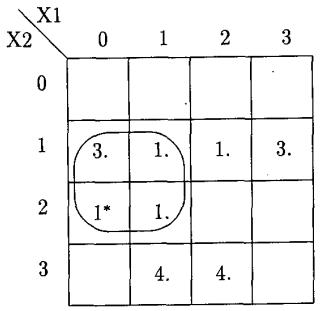

Figure 4: Map for Example 4; Step 3 of Table 3

Implicant	$1\ ^0x_1{}^0\ ^2x_2{}^2$	$1\ ^0x_1{}^0\ ^1x_2{}^2$	$1\ ^0x_1{}^1\ ^2x_2{}^2$	$1\ ^0x_1{}^1\ ^1x_2{}^2$
NRC	2	2	-1	-2

Table 2: The NRC values for $\alpha = 1\ ^0x_1{}^0\ ^2x_2{}^2$

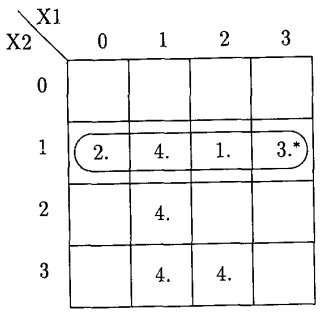

Figure 5: Map for Example 4; Step 4 of Table 3

Step (Figure)	Minimum CF_α	Minterm α	Minimum NRC	Candidate Implicant
1 (1)	0	$2^2x_1{}^2\ ^2x_2{}^2$	-3	$2^1x_1{}^2\ ^0x_2{}^3$
2 (2)	4	$1^2x_1{}^2\ ^3x_2{}^3$	-1	$1^1x_1{}^2\ ^3x_2{}^3$
3 (4)	4	$1^0x_1{}^0\ ^2x_2{}^2$	-2	$1^0x_1{}^1\ ^1x_2{}^2$
4 (5)	6	$3^3x_1{}^3\ ^1x_2{}^1$	-2	$3^0x_1{}^3\ ^1x_2{}^1$

Table 3: Steps of ND algorithm

4 Performance Results

We have run sample functions against three algorithms using VAX 11/785 and ISI workstations. We randomly generated a large set of functions (37,000) and applied each algorithm to minimize the functions similar to Tirumalai and Butler [10], and Yurchak and Butler [12]. We investigated three algorithms: (1) Pomper and Armstrong [8], (2) Dueck and Miller [4, 5, 12], and (3) Neighborhood Decoupling under various settings. The three settings are (1) 2-variable 4-valued with 3 to 16 input product terms, (2) 2-variable 5-valued with 3 to 25 input product terms, and (3) 3-variable 4-valued with 3 to 30 input product terms. In Table 4, we ran 1000 functions for each given number of input product terms, i.e., Table 4 is produced by running 14,000 testing functions. In Table 5 and 6 we ran 500 for each given number of input product terms, i.e., each Table takes 11,500 and 14,000 functions respectively to run.

4.1 Output Product Terms

For each setting, we compute the average number of output product terms (see Table 4, 5, 6). For example, in Table 4, we can see that for functions with 7 input product terms, on the average of 1000, these algorithms will minimize it to 4.203, 4.0, and 3.957 product terms (implicants) respectively. For all the cases, Neighborhood Decoupling algorithm outperforms the other two algorithms.

We plot the testing results in Tables 4, 5, and 6 in Figures 6, 7, and 8. In these bell-shaped figures we can see two important features: the neighborhood decoupling algorithm outperforms the other two algorithms and when the number of the input product terms reaches a certain point the number of output implicants decreases. This is due to the fact that the more the input product terms in a function the higher the tendency of generating truncated minterms. In most cases, a single implicant can cover a cluster of truncated minterms.

Number of Input Terms	Pomper and Armstrong	Dueck and Miller	Neighborhood Decoupling
3	2.838	2.715	2.692
4	3.483	3.280	3.262
5	3.916	3.690	3.665
6	4.178	3.952	3.925
7	4.203	4.000	3.957
8	4.201	3.982	3.949
9	4.072	3.915	3.875
10	3.913	3.749	3.711
11	3.717	3.563	3.526
12	3.573	3.432	3.396
13	3.362	3.249	3.213
14	3.178	3.077	3.044
15	2.991	2.903	2.881
16	2.759	2.691	2.666

Table 4: Two-Variable Four-Valued Average Output Product Terms

4.2 Performance Ratios

We can take another measure, the performance ratio, to demonstrate the performance of each algorithm. First, for each algorithm, we count the number of functions for which that specific algorithm is the "best" of the three, i.e., the number of instances where that specific algorithm uses the

Number of Input Terms	Pomper and Armstrong	Dueck and Miller	Neighborhood Decoupling
3	3.098	2.840	2.810
4	3.966	3.626	3.600
5	4.652	4.278	4.234
6	5.298	4.900	4.830
7	5.580	5.200	5.106
8	5.774	5.378	5.256
9	5.820	5.454	5.378
10	5.788	5.450	5.282
11	5.772	5.448	5.298
12	5.602	5.256	5.102
13	5.488	5.186	5.016
14	5.264	4.978	4.844
15	5.088	4.904	4.736
16	4.784	4.558	4.414
17	4.660	4.500	4.332
18	4.328	4.200	4.042
19	4.174	4.048	3.876
20	4.022	3.924	3.760
21	3.648	3.562	3.404
22	3.430	3.394	3.288
23	3.204	3.178	3.054
24	3.148	3.132	3.002
25	3.080	3.048	2.920

Table 5: Two-Variable Five-Valued Average Output Product Terms

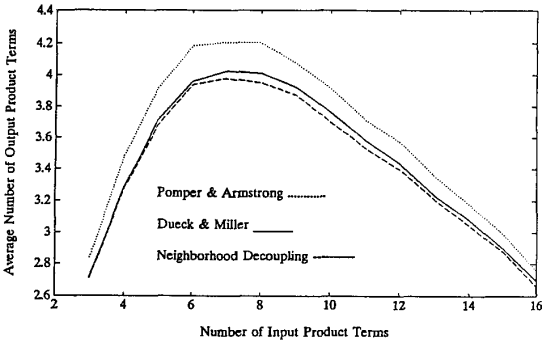

Figure 6: Two-variable Four-valued average product term

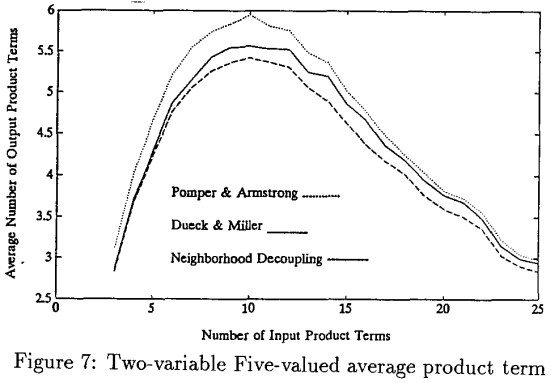

Figure 7: Two-variable Five-valued average product term

Number of Input Terms	Pomper and Armstrong	Dueck and Miller	Neighborhood Decoupling
3	3.026	2.934	2.928
4	4.110	3.892	3.916
5	5.164	4.822	4.810
6	6.124	5.658	5.622
7	7.082	6.468	6.456
8	7.918	7.178	7.220
9	8.646	7.744	7.802
10	9.240	8.394	8.332
11	9.670	8.822	8.762
12	10.028	9.228	9.130
13	10.342	9.626	9.524
14	10.534	9.906	9.774
15	10.844	10.218	10.044
16	10.796	10.234	10.032
17	11.044	10.442	10.206
18	11.002	10.488	10.244
19	10.854	10.300	10.000
20	10.602	10.122	9.842
21	10.478	10.034	9.672
22	10.356	9.898	9.558
23	10.162	9.822	9.390
24	9.978	9.514	9.210
25	9.600	9.260	8.934
26	9.478	9.122	8.738
27	9.210	8.882	8.526
28	9.042	8.690	8.350
29	8.738	8.490	8.120
30	8.538	8.306	7.874

Table 6: Three-Variable Four-Valued Average Output Product Terms

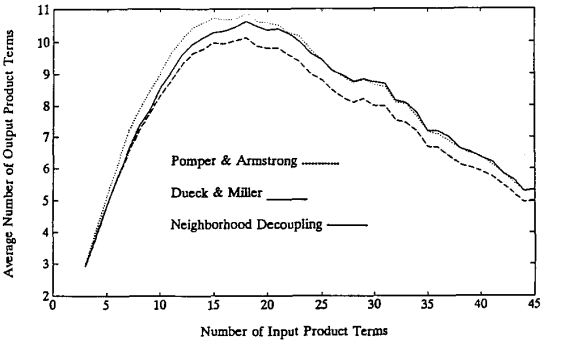

Figure 8: Three-variable Four-valued average product term

Performance	Pomper and Armstrong	Dueck and Miller	Neighborhood Decoupling
best	62	133	327
better	332	2059	2251
equal	11157	11157	11157
total	11551	13349	13735
ratio	0.8251	0.9535	0.9811

Table 7: Testing Results of 14000 2-Variable 4-Valued Sample Functions

Performance	Pomper and Armstrong	Dueck and Miller	Neighborhood Decoupling
best	124	310	1097
better	657	2156	2575
equal	7275	7275	7275
total	8056	9741	10947
ratio	0.7005	0.8470	0.9519

Table 8: Testing Results of 11500 2-Variable 5-Valued Sample Functions

Performance	Pomper and Armstrong	Dueck and Miller	Neighborhood Decoupling
best	622	1475	2613
better	1414	3765	4173
equal	4614	4614	4614
total	6650	9854	11400
ratio	0.4750	0.7039	0.8143

Table 9: Testing Results of 14000 3-Variable 4-Valued Sample Functions

minimum number of implicants (output product terms). If two algorithms use an equal number of implicants and less than the other one, we say they are "better" than the third one. When all three algorithms use equal number of implicants to minimize a function we say they are "equal". The performance ratio is defined as

$$\eta = (N_{best} + N_{better} + N_{equal}) \div N_{total}$$

where N_{best}, N_{better}, and N_{equal} are the number of instances that specific algorithm performs "best", "better", and "equal" respectively. This is the fraction of times when that algorithm gets as good or better solutions than the other two. The total number of functions tested, N_{total} in our case is either 11,500 or 14,000. Table 7, 8, and 9 show the performance ratios for each setting. For example, in Table 7, with 14,000 functions tested, we counted the cases which Neighborhood Decoupling algorithm performs no worse than the others as 13,735. That is, $\eta_{ND} = 13735/14000 = 0.9811$. Table 7, 8, 9 show that the performance is degraded when n or r is increased. However, the neighborhood decoupling algorithm consistently outperforms the others.

5 Analysis and Discussion

Notice that in the comparisons did not include Bessilich's algorithm and the absolute minimum solutions. We now justify the reasons. The current HAMLET does not include the Bessilich's algorithm. In addition, we know that the Dueck and Miller is a satisfactory [10, 12] heuristic that approximates well to the absolute minimum solutions.

One interesting question is: why is the Pomper and Armstrong algorithm a good choice in some cases? We explain our observations as follows. Let the x_0 indicates the minterm optimal solution will pick. Since the optimal solution is not unique [10] the exact x_0 may not be unique. We define the *hit ratio* as the probability that the α is the same as x_0. When there are only a few sparse minterms the hit ratio is higher so that the Pomper and Armstrong algorithm performs well.

We now explain why the neighborhood decoupling algorithm is superior both in using less implicants and in running faster than other algorithms. Like other algorithms, the theoretical computation complexity of the ND-algorithm is $O(r^{2n})$ since the most expensive computation is the algorithm N (see Section 3.2). This complexity is a direct consequence of Lemma 4. It may appear that the ND-algorithm has complex decision rules and may take longer time to do the selection. Neverthless, in numerous testings we found that the ND-algorithm, on the average, stops earlier than others, i.e., uses less implicants. In other words, the depth of the ND-algorithm in the HAMLET recursive computations is smaller than others. Therefore, in reality the ND-algorithm runs faster than the other two algorithms. This time efficiency is because the decision rules employed in the ND-algorithm take advantage of the special property of truncated sum operations. In other words, the input product terms have a tendency to produce saturated minterms. In the decoupling process (algorithm N), a minterm in SAT will always qualify to combine with its neighbors to form an implicant. Although the ND-algorithm conceptually is similar to the Dueck and Miller's algorithm, it uses saturated minterms in an effective way and the decoupling computation (NRC) is more finely tuned. For example, when we update (deduct) saturated minterm from the expression, the minterm will be updated to a "don't care minterm" (see Section 3). As in binary logic minimization, a "don't care minterm" [7] can simplify the minimization process.

Recall that, in the algorithm N, we compute the NRC values for a given minterm α by examing the relationships of I_α and its immediate neighbors, i.e., one step look-ahead. It is natural to believe that with more steps of look-ahead we might make a better choice of the implicant and therefore provide a better solution. The exponential growth of the number of possible implicants restricts the practical use of k-lookahead for $k \neq 1$.

6 Conclusion

The truncated sum MVL logic minimization can be done by the neighborhood decoupling algorithm which selects the most isolated minterms as well as implicants. Truncated sum operations may produce saturated minterms by its definition. In the development of the ND-algorithm we reduce a saturated minterm to a don't care minterm in the minimization process and use the don't care as much as we can. The ND-algorithm outperforms most heuristic methods and does not lose its run time efficiency because the algorithm finds the solution and stops earlier than others.

Acknowledgment

This work has benefited immensely from the work of HAMLET [12] by Yurchak and Butler. Jon Butler introduced the authors to the challenging world of the MVL and helped the authors in revising the paper. Our program development is based on HAMLET's C program and we add our algorithm as one independent option of the HAMLET's program. The program structure is discussed in [12]. Constructive suggestions of all referees are incorporated and authors are indebted to these referees for their efforts in making the paper more readable.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Reading, Mass., Addison-Wesley, 1974.

2. P. W. Besslich, "Heuristic minimization of MVL functions: a direct cover approach," *IEEE Trans. Comp.*, Vol C-35, Feb. 1986, pp.134-144.

3. J. Butler and H. G. Kerkhoff, "Multiple-Valued CCD Circuits", IEEE Computer, April, 1988, pp. 58-69.

4. G. W. Dueck and D. M. Miller, "A direct cover MVL minimization using the truncated sum" Proc. of 17th Intl. Symp. on MVL 1987.

5. G. W. Dueck, "Algorithms for The Minimization of Binary and Multiple-Valued Logic Functions", Ph. D. Dissertation, Department of Computer Science, University of Manitoba, Winnipeg, MB, 1988.

6. H. G. Kerkhoff, "Theory and design of multiple-valued logic CCD's," in Computer Science and Multiple-Valued Logic (ed. D. C. Rine), North Holland, New York, 1984, pp. 502-537.

7. E. J. McCluskey, *Logic Design Principles*, Englewood Cliffs, NJ, Prentice-Hall, 1986.

8. G. Pomper and J. A. Armstrong, "Representation of multivalued functions using the direct cover method," *IEEE Trans. Comp.* Sept. 1981, pp. 674-679

9. V. T. Rhyne, P. S. Noe, M. H. Mckinney, and U. W. Pooch, "A new technique for the fast minimization of switching functions", *IEEE Trans. Comp.*, August 1977, pp. 757 - 764.

10. P. Tirumalai and J. T. Butler, "Analysis of Minimization Algorithms for Multiple-valued PLA", Proc. of 18th Intl. Symp. on MVL 1988, pp. 226-236.

11. P. Tirumalai and J. T. Butler, "Prime and Non-Prime Implicants in the Minimization of Multiple-valued Logic Functions", Proc. of 19th Intl. Symp. on MVL 1989, pp. 272-279.

12. J. Yurchak and J. T. Butler, "HAMLET - An Expression Compiler/Optimizer for the Implementation of Heuristics to Minimize Multiple-Valued Programmable Logic Arrays", Proc. of 20th Intl. Symp. on MVL 1990.