PAPER *Special Section on VLSI Design and CAD Algorithms*

A Method to Find Linear Decompositions for Incompletely Specified Index Generation Functions Using Difference Matrix*

Tsutomu SASAO†a), *Member*, Yuta URANO†, *Nonmember*, *and* Yukihiro IGUCHI†, *Member*

SUMMARY This paper shows a method to find a linear transformation that reduces the number of variables to represent a given incompletely specified index generation function. It first generates the difference matrix, and then finds a minimal set of variables using a covering table. Linear transformations are used to modify the covering table to produce a smaller solution. Reduction of the difference matrix is also considered.
key words: *minimal cover, linear transformation, functional decomposition, incompletely specified function, logic minimization*

1. Introduction

Index generation functions [12] are useful in network applications [5] and pattern matching including computer virus scanning engines [4].

In many cases, functions must be updated frequently. Thus, a memory-based architecture is desirable. To reduce the size of the memory to implement index generation functions, a linear decomposition shown in Fig. 1 is quite effective [14]. When a given function is defined for only k input combinations and $k \ll 2^n$, the number of variables for the general function can be often reduced. To find a good decomposition, we use a linear transformation to reduce the number of variables p for the general function. In many cases, by this, the size of the LUT for the general function is drastically reduced.

In this paper, we show a new method to find a linear transformation that reduces the number of variables to represent a given incompletely specified index generation function. The rest of the paper is organized as follows: Sect. 2 defines index generation functions; Sect. 3 shows a method to reduce the number of variables; Sect. 4 introduces a difference matrix to reduce the number of variables; Sect. 5 shows a method to reduce variables using a linear transformations; Sect. 6 shows a heuristic method to find a good linear transformation; Sect. 7 shows a method to reduce the difference matrix; Sect. 8 shows experimental results; and Sect. 9 summarizes the paper.

2. Index Generation Function

Definition 2.1: Consider a set of k different vectors of n bits. These vectors are **registered vectors**. For each registered vector, assign a unique integer (index) from 1 to k. A **registered vector table** shows an **index** for each registered vector. An **incompletely specified index generation function** produces a corresponding index when the input vector matches a registered vector. Otherwise, the value of the function is undefined (d, *don't care*). The incompletely specified index generation function represents a mapping $M \rightarrow \{1, 2, \ldots, k\}$, where $M \subset B^n$ denotes the set of registered vectors. k is the **weight** of the function.

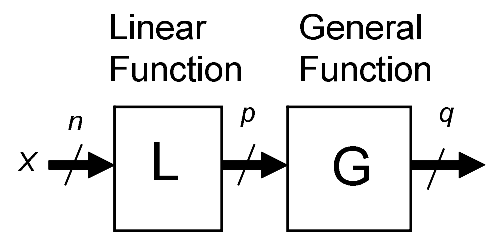

Fig. 1 Linear decomposition.

Table 1 Registered vector table.

Vector					
x_1	x_2	x_3	x_4	x_5	Index
1	0	0	0	0	1
0	1	0	0	0	2
0	0	1	0	0	3
0	0	0	1	0	4
0	0	0	0	1	5

Example 2.1: Consider the registered vectors shown in Table 1. These vectors show an index generation function with weight $k = 5$. ■

3. Number of Variables to Represent Incompletely Specified Functions

In an incompletely specified index generation function f, *don't care* values can be chosen as any value to minimize the number of variables to represent f. This property is useful to realize a function using a smaller memory (look-up table: LUT) [12].

Definition 3.1: Let $f(X)$ be an index generation function, and (X_1, X_2) be a partition of the input variables, where $X_1 = (x_1, x_2, \ldots, x_q)$ and $X_2 = (x_{q+1}, x_{q+2}, \ldots, x_n)$. The **decomposition chart** for f is a two-dimensional matrix with 2^q columns and 2^{n-q} rows, where each column and row is

Manuscript received March 10, 2014.
Manuscript revised May 26, 2014.
†The authors are with the Department of Computer Science, Meiji University, Kawasaki-shi, 214-8571 Japan.
*A preliminary version of this paper was presented at SASIMI-2013 Workshop [16].
a) E-mail: sasao@cs.meiji.ac.jp
DOI: 10.1587/transfun.E97.A.2427

Copyright © 2014 The Institute of Electronics, Information and Communication Engineers

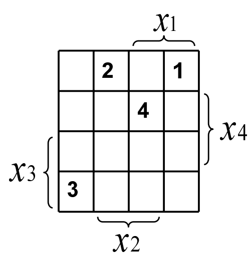

Fig. 2 Index generation function of 4 variables.

labeled by a unique binary code, and each element corresponds to the value of f.

Theorem 3.1: Suppose that an incompletely specified function f is represented by a decomposition chart. If each column has at most one *care* element, then the function can be represented by using only the column variables.

(Proof) In each column, let the values of *don't cares* elements be set to the value of the *care* element in the column, then the function depends only the column variables. □

Example 3.1: Consider the decomposition chart shown in Fig. 2, where x_1 and x_2 specify the columns, and x_3 and x_4 specify the rows, and blank elements denote *don't cares*. Note that in Fig. 2, each column has at most one *care* element. Thus, this function can be represented by only the column variables x_1 and x_2:

$$F = 1 \cdot x_1\bar{x}_2 \vee 2 \cdot \bar{x}_1 x_2 \vee 3 \cdot \bar{x}_1\bar{x}_2 \vee 4 \cdot x_1 x_2.$$

■

Algorithms to minimize the number of variables in incompletely specified functions have been developed [2], [3], [8], [10]. As for the lower bound on the number of variables, we have the following:

Theorem 3.2: [14] To represent any incompletely specified index generation function f with weight k, at least $q = \lceil \log_2 k \rceil$ variables are necessary.

This lower bound is useful for the minimum covering step in Algorithm 4.1[†].

Thus, when the weight k of an n-variable index generation function is greater than 2^{n-1}, we cannot reduce the number of variables.

4. Minimization of the Number of Variables Using Difference Matrix

In this section, we introduce the difference matrix to minimize the number of variables to represent a given incompletely specified index generation function.

Definition 4.1: [15], [17] Let M be the set of binary vectors corresponding to the minterms of f. Let D_f be the matrix where each row is a vector $\vec{a} \oplus \vec{b}$, $\vec{a}, \vec{b} \in M$, and $\vec{a} \neq \vec{b}$. D_f is called a **difference matrix** of M. Note that D_f consists of $\binom{k}{2} = \frac{k(k-1)}{2}$ vectors, where $k = |M|$.

[†]We use a branch-and-bound method to solve minimum covering problems. This bound is often better than other lower bounds for two-level logic minimizations [7].

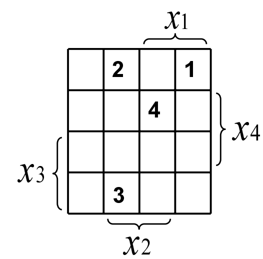

Fig. 3 Index generation function of 4 variables.

Table 2 Registered vectors before linear transformation.

x_1	x_2	x_3	x_4	Index
1	0	0	0	1
0	1	0	0	2
0	1	1	0	3
1	1	0	1	4

Table 3 Difference matrix before linear transformation.

x_1	x_2	x_3	x_4	Tag
1	1	0	0	(1, 2)
1	1	1	0	(1, 3)
0	1	0	1	(1, 4)
0	0	1	0	(2, 3)
1	0	0	1	(2, 4)
1	0	1	1	(3, 4)

Example 4.1: Consider the function shown in Fig. 3. Table 2 shows M, the set of vectors corresponding to the minterms for f. It is also called the **registered vector table**. Table 3 shows the corresponding difference matrix D_f. The last column of Table 3 shows tags specifying the pair of vectors in M. For example, the first vector in D_f has the tag (1, 2), which shows that the first and the second elements in M were used to generate the vector:

$$(1, 0, 0, 0) \oplus (0, 1, 0, 0) = (1, 1, 0, 0).$$

It shows that to distinguish the first and the second vectors in M, either x_1 or x_2 is necessary. ■

From the difference matrix, we can determine the conditions to distinguish all the pairs of vectors in M [8], and is essentially the same as the **covering table** [7]. Thus, we can find the minimal set of variables to represent an incompletely specified index generation function as follows:

Algorithm 4.1: (Minimal Sets of Variables to Represent an Incompletely Specified Index Generation Function)

1. Let M be the set of vectors showing an incompletely specified index generation function.
2. Generate D_f, the difference matrix, from M.
3. Assume that in D_f, each column corresponds to a variable x_j, and each row corresponds to a vector in D_f.
4. The element (i, j) of the covering table is 1 iff the j-th element of the i-th vector in D_f is 1.
5. Derive the minimal set of variables that covers all the rows of D_f.

Example 4.2: Consider the index generation function shown in Fig. 3. Table 2 shows the registered vector table. Note that the number of the columns is $n = 4$, while the number of the rows is $\binom{k}{2} = \frac{k(k-1)}{2} = \frac{4\times 3}{2} = 6$. The first row with the tag (1,2) corresponds to the first element in D_f, which show that to distinguish the 1st and 2nd vectors in M, either x_1 or x_2 is necessary. Also, note that the row with the tag (2,3) has only single one. This row is covered only by the column of x_3. Such a variable is **essential** and, is necessary in all solutions. Minimal sets of variables that cover all the rows are $\{x_1, x_2, x_3\}$, $\{x_1, x_3, x_4\}$, and $\{x_2, x_3, x_4\}$. ■

To find a minimal set of variables, we can use a standard method [7]. Although the method is straightforward, it takes much computation time when n and k are large.

5. Reduction of Variables by Linear Transformations

In the previous section, we showed a method to reduce the number of variables for incompletely specified functions. Unfortunately, the effect of such method is limited. In this section, we show that more variables can be reduced by using linear transformations†.

Example 5.1: The number of 1's in each row of Table 1, is one. Note that, the number of variables to represent the function can be reduced to four: Any one variable can be removed. For example, if we remove x_5, then we have:

$$F = 1 \cdot x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee 2 \cdot \bar{x}_1 x_2\bar{x}_3\bar{x}_4 \vee 3 \cdot \bar{x}_1\bar{x}_2 x_3\bar{x}_4 \vee 4 \cdot \bar{x}_1\bar{x}_2\bar{x}_3 x_4 \vee 5 \cdot \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4.$$

However, we cannot remove two or more variables simultaneously. Thus, at least four variables are necessary to represent this function. ■

Definition 5.1: A **linear transformation** is defined as

$$\begin{aligned} y_1 &= c_{11}x_1 \oplus c_{12}x_2 \oplus c_{13}x_3 \oplus \ldots \oplus c_{1n}x_n, \\ y_2 &= c_{21}x_1 \oplus c_{22}x_2 \oplus c_{23}x_3 \oplus \ldots \oplus c_{2n}x_n, \\ y_3 &= c_{31}x_1 \oplus c_{32}x_2 \oplus c_{33}x_3 \oplus \ldots \oplus c_{3n}x_n, \\ &\vdots \\ y_p &= c_{p1}x_1 \oplus c_{p2}x_2 \oplus c_{p3}x_3 \oplus \ldots \oplus c_{pn}x_n, \end{aligned}$$

where $c_{ij} \in \{0, 1\}$. $t_i = \sum_{j=1}^{n} c_{ij}$ is the **compound degree** of y_i.

Definition 5.2: Given an incompletely specified index generation function, an **optimum linear transformation** is one that minimizes the number of variables p in Fig. 1.

By Theorem 3.2, if the linear transformation reduces the number of variables to $q = \lceil \log_2 k \rceil$ variables, then it is optimum.

†Here, we only consider linear transformations because 1)They are easy to implement by hardware, and 2) They are easy to analyze. However, it is also possible to use non-liner transformations. Currently, we have no design method using non-linear transformations.

Table 4 Registered vectors after linear transformation.

Vector			Index
y_1	y_2	y_3	
1	1	0	1
1	0	0	2
0	1	0	3
0	0	1	4
0	0	0	5

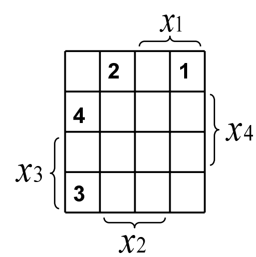

Fig. 4 Index generation function of 4 variables.

Example 5.2: For the function in Table 1, consider the linear transformation:

$$\begin{aligned} y_1 &= x_1 \oplus x_2, \\ y_2 &= x_1 \oplus x_3, \\ y_3 &= x_4. \end{aligned}$$

The transformed registered vectors are shown in Table 4. In this case, all the vectors are distinct, and three variables (y_1, y_2, y_3) distinguish five vectors. Note that this is an optimum transformation. ■

6. A Heuristic Method to Find Linear Transformations

6.1 Strategies to Find a Good Linear Transformation Using Difference Matrix

Since the number of linear transformations to be considered is very large [1], in this section, we present a heuristic method to find a linear transformation that reduces the number of variables.

Assume that x_i is transformed to $y_i \Leftarrow x_i \oplus x_j$ in M. Consider the effect of this linear transformation in D_f. Let $\vec{a}$ and $\vec{b}$ be different row vectors in M. Note that only the i-th part of the vectors is modified. Modified vectors in M are written as:

$$\begin{aligned} \vec{a}' &= (a_1, a_2, \ldots, a_i \oplus a_j, a_{i+1}, \ldots, a_n). \\ \vec{b}' &= (b_1, b_2, \ldots, b_i \oplus b_j, b_{i+1}, \ldots, b_n). \end{aligned}$$

Thus, we have
$\vec{a}' \oplus \vec{b}' = (a_1 \oplus b_1, a_2 \oplus b_2, \ldots, (a_i \oplus b_i) \oplus (a_j \oplus b_j), a_{i+1} \oplus b_{i+1}, \ldots, a_n \oplus b_n) = \vec{a} \oplus \vec{b} \oplus (0, 0, \ldots, 0, a_j \oplus b_j, 0, 0, \ldots, 0)$.
Note that also in D_f, only the i-th part of the vectors is modified. This means that the linear transformation can be also done in D_f.

Example 6.1: Consider the index generation function shown in Fig. 3, and apply the linear transformation: $y_2 \Leftarrow$

Table 5 Registered vectors after transformation.

x_1	$\mathbf{y_2}$	x_3	x_4	Index
1	0	0	0	1
0	1	0	0	2
0	**0**	1	0	3
1	1	0	1	4

Table 6 Difference matrix after transformation.

x_1	$\mathbf{y_2}$	x_3	x_4	Tag
1	1	0	0	(1, 2)
1	**0**	1	0	(1, 3)
0	1	0	1	(1, 4)
0	**1**	1	0	(2, 3)
1	0	0	1	(2, 4)
1	**1**	1	1	(3, 4)

Table 7 Original difference matrix.

x_1	x_2	x_3	x_4	x_5	Tag
1	1	0	0	0	(1, 2)
1	0	1	0	0	(1, 3)
1	0	0	1	0	(1, 4)
1	0	0	0	1	(1, 5)
0	1	1	0	0	(2, 3)
0	1	0	1	0	(2, 4)
0	1	0	0	1	(2, 5)
0	0	1	1	0	(3, 4)
0	0	1	0	1	(3, 5)
0	0	0	1	1	(4, 5)

Table 8 Difference matrix after 1st reduction.

x_1	x_2	x_3	x_4	x_5	Tag
1	1	0	0	0	(1, 2)
0	0	1	1	0	(3, 4)
0	0	1	0	1	(3, 5)
0	0	0	1	1	(4, 5)

$x_2 \oplus x_3$. Table 5 shows M after the transformation, while Table 6 shows D_f after the transformation. Note that, in the transformed D_f, each row has at least two non-zero elements. Also, the total number of 1's in the transformed D_f is increased. In the transformed D_f, variable x_3 is not essential any more. Minimal sets of variables that cover all the rows are $\{x_1, y_2\}, \{x_1, x_3, x_4\}$, and $\{y_2, x_3, x_4\}$. Note that the linear transformation reduced the number of variables to two. ∎

The previous example suggests that D_f with more 1's tends to produce smaller solutions. Let the **merit of a variable** be the number of rows covered by the variable. Our strategy is to **find a linear transformed variable with maximal merit**. Then, we eliminate the rows of D_f covered by this variable. Repeat this process until all the rows of D_f are eliminated.

Example 6.2: Consider the function in Table 1. The difference matrix is shown in Table 7. First, we obtain a linear transformed variable with the maximal merit. Since $y_1 = x_1 \oplus x_2$ is such a variable, we select this transformation. Then, we remove the rows of D_f that are covered by y_1. Since the rows for (1,3),(1,4),(1,5),(2,3),(2,4) and (2,5) are covered by y_1, we remove them from D_f, and have the reduced difference matrix shown in Table 8.

Then, we find the second linear function that maximally covers the remaining rows shown in Table 8. In this case, $y_2 = x_1 \oplus x_3$ covers maximal number of rows, we select this transformation. In this case, rows for (1,2), (3,4), and (3,5) are removed, and only the row (4,5) remains. Since, the row (4,5) can be covered by $y_3 = x_4$, we select this as the third transformed variable. In this way, we can cover all the rows of D_f. The resulting linear transformed variables are exactly the same as ones introduced in Example 5.2. ∎

6.2 An Algorithm to Find Good Linear Transformations

From the previous observation, we have the following:

Algorithm 6.1: (A greedy algorithm to find a set of linear transformations)

1. Let M be the set of vectors showing an incompletely specified index generation function.
2. Generate D_f, the difference matrix, from M. Eliminate duplicated rows using Theorem 7.1.
3. Find a linear transformed variable y_i that covers the maximal number of rows in D_f using Algorithm 6.2.
4. Eliminate the rows in D_f that are covered by the linear transformed variable y_i.
5. Repeat this process until all the rows of D_f are eliminated.

In Step 3, we used the following:

Algorithm 6.2: (A greedy algorithm to find a linear transformed variable)

1. Find a variable x_i that has the maximal merit. Let $y_i \Leftarrow x_i$.
2. Find another variable x_j such that $y_i \oplus x_j$ covers the maximum number of rows in D_f. If the number of covered rows is increased, then $y_i \Leftarrow y_i \oplus x_j$.
3. Repeat above operation while the number of covered rows is increased.

Algorithm 6.1 can be considered as an improvement of [17]. Our method to find linear transformed variables $y_1, y_2, \ldots, y_p$ is more efficient and effective, since we used iterative improvement method recently introduced in [15].

7. Reduction of Difference Matrix

In the previous section, we showed that a good linear transformation can be found using a difference matrix. Note that the difference matrix has $\frac{k(k-1)}{2}$ rows and n columns. Thus, when k is large, the matrix would be too large. In this section, we show a method to reduce the number of rows of the difference matrix. In the difference matrix, row vectors with the same patterns can appear.

Example 7.1: Consider the registered vectors shown in Table 9. The corresponding difference matrix is shown in Table 10. Note that in the difference matrix, the first vector and the last vectors are the same. Also, the second vector and fifth one are the same. Also, the third and fourth vectors are the same. In this case, if the first three rows are covered,

Table 9 Registered vectors.

x_1	x_2	x_3	x_4	Index
1	1	0	0	1
0	0	1	0	2
1	0	0	1	3
0	1	1	1	4

Table 10 Difference matrix with duplicated rows.

x_1	x_2	x_3	x_4	Tag
1	1	1	0	$(1,2)$
0	1	0	1	$(1,3)$
1	0	1	1	$(1,4)$
1	0	1	1	$(2,3)$
0	1	0	1	$(2,4)$
1	1	1	0	$(3,4)$

then the last three rows are also covered. In other words, the linear transformation can be obtained by using only the first three vectors. ■

Theorem 7.1: Consider the difference matrix of an index generation function. An optimal linear transformation can be found by using the reduced difference matrix.

The linear transformation can be performed in the reduced difference matrix. In some cases, the number of rows can be reduced drastically.

For random functions, the reduction of duplicated rows is effective when k is large:

Theorem 7.2: Consider a random index generation function with weight k. Let $R(n,k)$ be the ratio of the numbers of vectors after the reduction of duplicated rows to that of vectors before reduction. Then, we have

$$R(n,k) \simeq \frac{2^{n+1}(1-e^{-\frac{k^2}{2^{n+1}}})}{k^2}.$$

(Proof) To obtain the number of rows in the difference matrix after reduction, consider Table 11, which shows the distribution of patterns. In this table, 1) V_i denotes a vector in the difference matrix; 2) P_j denotes a pattern of the row vector; 3) L denotes the number of rows of the original difference matrix; and 4) $N = 2^n$ denotes the number of all possible bit patterns of n bits. For example, $\surd$ in the second row of Table 11 shows that the pattern of V_1 is P_0. The number of rows in the difference matrix after reduction is equal to the total number of columns with $\surd$ in Table 11. In other words, the expected number of columns that have $\surd$ in Table 11 is equal to the number of rows in the difference matrix after reduction.

Assume that the distribution of 0 and 1 in the difference matrix is random. Each row has exactly one $\surd$. Thus, the probability that a column has a $\surd$ in a row is $\frac{1}{N}$. Thus, the probability that a certain column does not have $\surd$ is $P_r = (1-\frac{1}{N})^L$. Since the value of $\frac{1}{N}$ is sufficiently small, it can be approximated by $P_r \simeq e^{-L/N}$.

Thus, the probability that a certain column has a $\surd$ is $1 - P_r = 1 - e^{-L/N}$. Since, the number of columns is N, the expected number of columns with $\surd$ is $N(1-e^{-L/N})$. Note that $L = \frac{k(k-1)}{2}$. Thus, we have the theorem. □

Table 11 Distribution of patterns in the difference matrix.

	P_0	P_1	......	P_{N-1}
V_1	$\surd$			
V_2		$\surd$		
⋮				
⋮				
V_L				$\surd$

Table 12 Number of variables to represent m-out-of-20 code to index converter.

		# of Variables and CPU time [ms]			
m	k	[14]	CPU	HEUR	CPU
1	20	6	2611	**5**	6
2	190	9	5919	10	273
3	1140	**11**	76939	13	9835
4	4845	16	1110684	16	182448

8. Experimental Results

8.1 Comparison with Existing Methods

We developed a program for Algorithm 6.1. For simplicity, reduction of the difference matrix is not incorporated into the program.

As for the benchmark functions, we used *m-out-of-n* code to index converters [14]. They are index generation functions with weight $k = \binom{n}{m}$. Table 1 shows the case of $n = 5$ and $m = 1$. In this case, the i-th variable has 1 and other variables have 0 in the input if and only if the value of the function is i. The minimum number of variables to represent a *1-out-of-n* code to index converter is $\lceil \log_2 n \rceil$. For up to $n = 256$, our program obtained exact minimum solutions. The CPU time for $n = 256$ was 67.7 sec. These results are much better than previous results [15], [17]. For example, in [17], linear transformed variables were generated randomly, so experimental results for only up to $n = 12$ were reported.

Table 12 shows the results for m-out-of-20 code to index converters. The column headed by [14] shows the results in ASPDAC-2012 [14]. In this case, all the transformed variables with the compound degrees of up to six were considered. Note that this method requires memory proportional to

$$k \times \sum_{i=1}^{t=6} \binom{n}{i},$$

where t denotes the compound degree.

The presented program is faster and requires much less memory than one in [14], although the qualities of solutions are lower. In the experiment, we used a PC using an INTEL Core i5-2450M CPU @2.5 GHz; Windows 7 64-bit operating system; and 8.00 GB RAM. In Table 12, the figures shown in bold face denote optimum solutions.

Table 13 compares the present algorithm with existing ones, where t denotes the compound degree used for linear

Table 13 Comparison with existing methods.

	Exhaustive Method ISMVL 2011	Heuristic Method ASPDAC 2012	Heuristic Method SASIMI 2013
Memoroy size	$O(k2^n)$	$O(kn^t)$	$O(nk^2)$
CPU time	Too Large	Medium	Small
Quality of Solutions	Exact Minimum	Good	Good

Table 14 Numbers of vectors in the difference matrix for m-out-of-n code to index converters.

m	k	$Total$	$Distinct$	$Ratio$
1	20	190	190	1.000000
2	190	17955	5035	0.280423
3	1140	649230	43795	0.067457
4	4845	11734590	169765	0.014467

transformations.

8.2 Reduction of Difference Matrix and Their Effects

m-out-of-n code to index converters

Table 14 shows the numbers of rows in the difference matrix for m-out-of-n code to index converters. The number of registered vectors is $k = \binom{20}{m}$. On the other hand, the number of rows in the difference matrix is $\binom{k}{2} = \frac{k(k-1)}{2}$.

For these functions, when the duplicated rows are removed, the number of rows is drastically reduced. The column headed $Total$ shows $\frac{k(k-1)}{2}$, the number rows in the original difference matrix. The column headed $Distinct$ show the number of distinct rows in the difference matrix, or the number of rows after reduction. The last column shows $Ratio = \frac{Distinct}{Total}$.

1. When $m = 1$, all the rows are distinct.
2. When $m = 2$, the difference matrix consists of rows with weights two and four. Thus, there are $\binom{20}{2} + \binom{20}{4} = 5035$ patterns.
3. When $m = 3$, the difference matrix consists of rows with weights two, four and six. Thus, there are $\binom{20}{2} + \binom{20}{4} + \binom{20}{6} = 43795$ patterns.
4. When $m = 4$, the difference matrix consists of rows with weights two, four, six and eight. Thus, there are $\binom{20}{2} + \binom{20}{4} + \binom{20}{6} + \binom{20}{8} = 169765$ patterns.

For this class of functions, the reduction of the difference matrix is quite effective when k is large.

Random Functions

Table 15 shows the numbers of rows in the difference matrices for random index generation functions with weight k, where $k = 20, 190$ and 1140. The numbers of distinct vectors are average of 100 randomly generated functions. When $k = 4845$, the number of rows in the difference matrix is greater than $2^n = 1048576$. Thus, the values are omitted. These values are quite near to the estimated values obtained by Theorem 7.2.

Table 15 Numbers of vectors in the difference matrix for random functions.

n	k	$Total$	$Distinct$	$Ratio$
20	20	190	190.0	1.00000
20	190	17955	17802.1	0.99149
20	1140	649230	484443.7	0.74619
20	4845	11734590	– – – –	– – – –

Table 16 Reduction of duplicated rows: Estimation and experiment.

		Statistical	Experimental	
n	k	$R(n,k)$	$Average$	SD
10	30	0.8092	0.8320	0.0268
10	50	0.5775	0.5981	0.0140
10	100	0.2032	0.2058	0.0004
15	100	0.9274	0.9306	0.0059
15	200	0.7485	0.7530	0.0038
15	300	0.5438	0.5473	0.0021
15	400	0.3739	0.3757	0.0008
15	500	0.2564	0.2572	0.0002
15	600	0.1813	0.1817	0.0000
15	700	0.1337	0.1339	0.0000
20	200	0.9905	0.9908	0.0012
20	300	0.9788	0.9791	0.0015
20	400	0.9628	0.9632	0.0008
20	500	0.9427	0.9431	0.0009

8.3 An Experiment Supporting Theorem 7.2

To confirm the validity of Theorem 7.2, we produced 100 random index generation functions, derived their difference matrices, and counted the numbers of duplicated rows. Table 16 compares $R(n,k)$ with experimental values (Average and Standard Deviation: SD) for different values of n and k. Table 16 shows that estimated values $R(n,k)$ predict the experimental results fairly well.

9. Conclusion

Major contributions of this paper are:

- Showed an algorithm to derive minimal sets of variables to represent f using a difference matrix.
- Showed a heuristic algorithm to find a good linear transformed variable to cover a difference matrix.
- Developed an efficient computer program which is much faster and requires smaller memory than previous methods.
- Showed that the difference matrix can be reduced when k is large.

Finding a good linear transformation requires a modification of the covering table so that the solution is reduced. We perform this by selecting a transformed variable that covers the maximal number of uncovered rows, step by step.

In our applications [4], [5], values of n are around 20–256, while values of k are up to 10^6. Thus, the proposed method is still too time consuming for large problems. Thus, in large problems, we have to partition the vectors into smaller groups and implement each group separately.

Acknowledgments

This work is partially supported by the Japan Society for the Promotion of Science (JSPS), Grant in Aid for Scientific Research, and by the Adaptable and Seamless Technology Transfer Program through target-driven R&D, JST. Prof. Jon T. Butler improved English presentation.

References

[1] E. Artin, Geometric Algebra, Interscience Publishers, New York, 1957.

[2] C. Halatsis and N. Gaitanis, "Irredundant normal forms and minimal dependence sets of a Boolean function," IEEE Trans. Comput., vol.C-27, no.11, pp.1064–1068, Nov. 1978.

[3] Y. Kambayashi, "Logic design of programmable logic arrays," IEEE Trans. Comput., vol.C-28, no.9, pp.609–617, Sept. l979.

[4] H. Nakahara, T. Sasao, and M. Matsuura, "A low-cost and high-performance virus scanning engine using a binary CAM emulator and an MPU," 8th International Symposium on Applied Reconfigurable Computing, (ARC 2012), Hong-Kong, March 2012. Also, Lect. Notes Comput. Sci., vol.7199, pp.202–214, 2012.

[5] H. Nakahara, T. Sasao, and M. Matsuura, "An architecture for IPv6 lookup using parallel index generation units," The 9th International Symposium on Applied Reconfigurable Computing (ARC2013), Los Angeles, March 2013. Also, Lect. Notes Comput. Sci., vol.7806, pp.59–71, 2013.

[6] E.I. Nechiporuk, "On the synthesis of networks using linear transformations of variables," Dokl. AN SSSR, vol.123, no.4, pp.610–612, Dec. 1958.

[7] T. Sasao, Switching Theory for Logic Synthesis, Kluwer Academic Publishers, 1999.

[8] T. Sasao, "On the number of dependent variables for incompletely specified multiple-valued functions," International Symposium on Multiple-Valued Logic (ISMVL-2000), pp.91–97, Portland, Oregon, U.S.A., May 2000.

[9] T. Sasao, "Design methods for multiple-valued input address generators," (invited paper) International Symposium on Multiple-Valued Logic (ISMVL-2006), pp.1–10, Singapore, May 2006.

[10] T. Sasao, "On the number of variables to represent sparse logic functions," ICCAD-2008, pp.45–51, San Jose, California, USA, Nov. 2008.

[11] T. Sasao, T. Nakamura, and M. Matsuura, "Representation of incompletely specified index generation functions using minimal number of compound variables," 12th EUROMICRO Conference on Digital System Design, Architectures, Methods and Tools (DSD 2009), pp.765–772, Patras, Greece, Aug. 2009.

[12] T. Sasao, Memory-Based Logic Synthesis, Springer, 2011.

[13] T. Sasao, "Index generation functions: Recent developments," (invited paper) International Symposium on Multiple-Valued Logic (ISMVL-2011), pp.1–9, Tuusula, Finland, May 2011.

[14] T. Sasao, "Linear decomposition of index generation functions," 17th Asia and South Pacific Design Automation Conference (ASPDAC-2012), pp.781–788, Sydney, Australia, Jan.- Feb. 2012.

[15] T. Sasao, "An application of autocorrelation functions to find linear decompositions for incompletely specified index generation functions," 43rd International Symposium on Multiple-Valued Logic (ISMVL-2013), pp.96–102, Toyama, Japan, May 2013.

[16] T. Sasao, Y. Urano, and Y. Iguchi, "A heuristic method to find linear decompositions for incompletely specified index generation functions," The 18th workshop on Synthesis and system Integration of Mixed Information Technologies (SASIMI-2013), R3-1, pp.143–148, Sapporo, Japan, Oct. 2013.

[17] D.A. Simovici, M. Zimand, and D. Pletea, "Several remarks on index generation functions," International Symposium on Multiple-Valued Logic (ISMVL-2012), pp.179–184, Victoria, Canada, May 2012.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in electronics engineering from Osaka University, Osaka, Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan; IBM T. J. Watson Research Center, Yorktown Height, NY; the Naval Postgraduate School, Monterey, CA; and Kyushu Institute of Technology, Iizuka, Japan. Now, he is a Professor of Department of Computer Science, Meiji University, Kawasaki, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than nine books on logic design, including *Logic Synthesis and Optimization*, *Representation of Discrete Functions*, *Switching Theory for Logic Synthesis*, *Logic Synthesis and Verification*, and *Memory-Based Logic Synthesis*, in 1993, 1996, 1999, 2001, and 2011, respectively. He has served as Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan, in 1998. He received the NIWA Memorial Award in 1979, Takeda Techno-Entrepreneurship Award in 2001, and Distinctive Contribution Awards from the IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003, 2004 and 2013. He has served as an Associate Editor of the *IEEE Transactions on Computers*. He is a Fellow of the IEEE.

Yuta Urano received the B.E. degree in computer science from Meiji University in 2013. He is now a Mater Student of the same university.

Yukihiro Iguchi received the B.E., M.E., and Ph.D. degrees in electronic engineering from Meiji University, Kanagawa Japan, in 1982, 1984, and 1987, respectively. He is now a professor of Meiji University. His research interest includes logic design, switching theory, and reconfigurable systems. In 1996 and 2006, he spent each year at Kyushu Institute of Technology. He received Takeda Techno-Entrepreneurship Award in 2001.