Index Generation Functions: Minimization Methods

Tsutomu Sasao
Meiji University, Kawasaki 214-8571, Japan

Abstract—**Incompletely specified index generation functions can often be represented with fewer variables than original functions by appropriately assigning values to don't cares. The number of variables can be further reduced by using a linear transformation to the input variables. Minimization of variables under such conditions was considered to be a very hard problem. This paper surveys minimization methods for index generation functions. Major topics include 1) An upper bound on the number of variables to represent index generation functions; 2) A heuristic minimization method using an ambiguity measure; 3) A heuristic minimization method using remainders of a polynomial on GF(2); 4) An exact minimization method using a SAT solver; and 5) Comparison of minimization methods.**

Keywords-**Incompletely specified function, Index generation function, Linear transformation, Variable minimization, SAT solver, Logic minimization, Galois field.**

I. Introduction

One of the basic operations in information processing is to search desired data from a large data set. For example, consider a network router, where IP addresses are represented by 128 bits. Assume that a network router stores 40,000 of the 2^{128} possible combinations of the inputs, and checks if an input pattern matches a stored pattern. A content addressable memory (CAM) [17] is a device that performs this operation directly. Unfortunately, CAMs dissipate high power and are very expensive.

An **index generation function** [28] describes the operation of a CAM. For example, an index generation function can be represented by a **registered vector table** shown in Table 1.1.

An efficient method to implement an index generation function is the **IGU** (Index Generation Unit) shown in Fig. 1.1 [20], [28]. Since an IGU uses ordinary memory and a small amount of logic, cost and power dissipation are much lower than with a typical CAM. In the IGU, the **linear circuit** reduces the number of inputs n to p, and the **main memory** produces a tentative index. The **AUX memory** stores other inputs of the registered vector. The **comparator** checks if the tentative index is correct or not. If it is correct, then the **AND** gate produces the index. Otherwise the AND gate produces zero, which shows that the input vector is not registered. In this case, the main memory can realize any value for the non-registered inputs. In other words, in the design of the main memory, we can consider an **incompletely specified function** instead of a completely specified function. For such a function, the number of inputs to the main memory p is likely to be smaller than n, the original number of inputs. In this way, we can reduce the cost of the IGU. Thus, the minimization of variables for the main memory is the key issue of the design. This paper surveys various minimization methods of variables for the main memory.

TABLE 1.1
REGISTERED VECTOR TABLE

x_1	x_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

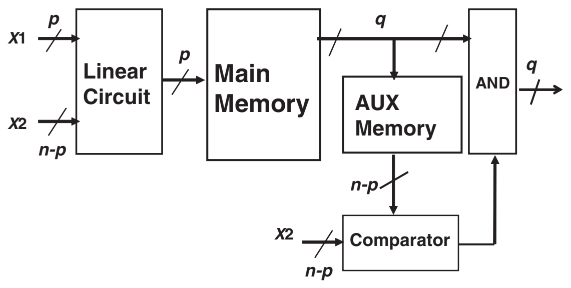

Fig. 1.1. Index Generation Unit (IGU).

The rest of the paper is organized as follows: Section II introduces the index generation function; Section III shows a method to minimize the number of variables for incompletely specified index generation functions; Section IV introduces a linear decomposition to reduce the number of input variables. It also considers upper and lower bounds on the number of variables to represent index generation functions; Section V shows a heuristic method using imbalance measure and ambiguity measure; Section VI introduces a polynomial-based method to reduce the number of variables; Section VII shows an exact minimization method of variables by using a SAT solver; Section VIII compares various minimization methods by experiments; Section IX surveys other works on index generation functions; and Section X concludes the paper.

II. Index Generation Functions

Definition 2.1: Consider a set of k distinct vectors of n bits. These vectors are **registered vectors**. For each registered vector, assign a unique integer from 1 to k. A **registered vector table** shows the **index** for each registered vector. An **incompletely specified index generation function** produces the corresponding index when the input vector equals to a registered vector. Otherwise, the value of the function

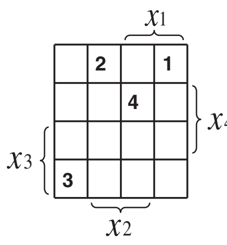

Fig. 2.1. 4-variable index generation function.

is undefined (*don't care*). An incompletely specified index generation function shows a mapping $M \to \{1, 2, \ldots, k\}$, where $M \subset B^n$ represent a set of registered vectors. k is the **weight** of the function.

Example 2.1: Table 1.1 shows a registered vector table, that represents an index generation function with weight $k = 4$. ■

An incompletely specified function f can often be converted into a completely specified functions with fewer variables by appropriate assignment of 0's and 1's into *don't care* values. This property is useful to realize the functions by look-up tables (LUTs).

Theorem 2.1: Assume that an incompletely specified function f is represented by a decomposition chart [18]. If each column of the decomposition chart has at most one care element, then the function can be represented by only column variables.

Example 2.2: Consider the decomposition chart in Fig. 2.1. x_1 and x_2 specify columns, while x_3 and x_4 specify rows. Also, blank cells denote *don't cares*. In Fig. 2.1, each column has at most one care element. Thus, this function can be represented with only the column variables x_1 and x_2:

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

■

III. Exact Minimization of Variables using Difference Matrix

Algorithms to minimize variables for incompletely specified logic functions have been developed [5], [6], [19], [24]. In this section, we introduce difference matrices to minimize the number of variables for incompletely specified index generation functions.

Definition 3.1: [37], [52] Let M be the set of binary vectors corresponding to the minterms of f. The **difference matrix** D_f of M is the set of vectors $\vec{a} \oplus \vec{b}$, where $\vec{a}, \vec{b} \in M$, and $\vec{a} \neq \vec{b}$.

Note that D_f consists of at most $\binom{k}{2} = \frac{k(k-1)}{2}$ binary vectors, where k is the number of vectors in M.

Example 3.1: Consider the function shown in Fig. 2.1. Table 1.1 is the registered vector table, and shows the set M that corresponds to the minterms of the function. Table 3.1 is the corresponding difference matrix D_f. The last column of Table 3.1 represents tags showing pairs of vectors in M. For example, the tag of the first vector in D_f is $(1, 2)$. This tag

TABLE 3.1
Difference Matrix D_f

x_1	x_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	$(3,4)$

shows that the first and the second vectors in M were used to generate the vector:

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

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

The difference matrix shows the condition to distinguish all the vectors in M, and is similar to the **covering table** [18] used in the minimization of sum-of-products expressions. Thus, we have the following algorithm to minimize the variables for an incompletely specified index generation function [41].

Algorithm 3.1: (Exact minimization of primitive variables)

1) Let M be the set of registered vectors for the given incompletely specified index generation function.
2) Generate the difference matrix D_f from M. The j-th column of D_f corresponds to the variable x_j. The j-th element of the i-th vector in D_f is 1 if and only if the value of the (i, j)-th element of the covering table is 1.
3) Derive the minimum set of columns that covers all the rows of D_f.

Example 3.2: Consider the index generation function shown in Fig. 2.1. Table 1.1 is the registered vector table. Table 3.1 is the difference matrix, where the number of columns is $n = 4$, while the number of rows is $\binom{k}{2} = \frac{k(k-1)}{2} = \frac{4\times 3}{2} = 6$. To obtain a minimum cover, we use the **covering function**. In the difference matrix shown in Table 3.1, to satisfy the condition in the first row (with the tag (1,2)), we need either x_1 or x_2. To satisfy the condition of the second row, we need either x_1 or x_3. To satisfy the condition of the third row, we need either x_2 or x_4. To satisfy the condition of the fourth row, we need either x_2 or x_3. To satisfy the condition of the fifth row, we need either x_1 or x_4. To satisfy the condition of the sixth row, we need either x_1 or x_2 or x_3 or x_4.

From these, we have the following covering function:

$$\begin{aligned} P &= (x_1 \vee x_2)(x_1 \vee x_3)(x_2 \vee x_4)(x_2 \vee x_3) \\ &\quad (x_1 \vee x_4)(x_1 \vee x_2 \vee x_3 \vee x_4). \end{aligned}$$

By applying the distributive law and the absorption law, we have

$$\begin{aligned} P &= (x_1 \vee x_2x_3)(x_2 \vee x_3x_4)(x_1 \vee x_4) \\ &= (x_1 \vee x_2x_3x_4)(x_2 \vee x_3x_4) \\ &= x_1x_2 \vee x_2x_3x_4 \vee x_1x_3x_4. \end{aligned}$$

From this, we have three minimal sets of variables that cover all the rows: $\{x_1, x_2\}, \{x_1, x_3, x_4\}$, and $\{x_2, x_3, x_4\}$. ■

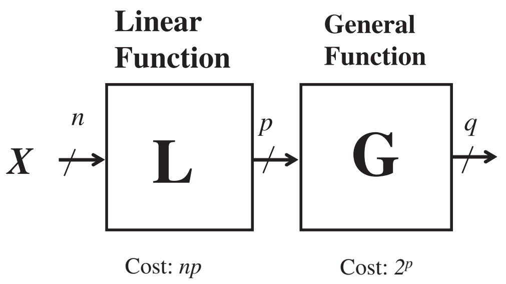

Fig. 4.1. Linear Decomposition

Algorithm 3.1 shows just the idea of minimization. In a practical implementation [24], various techniques are used to reduce computation time and memory requirement.

IV. Linear Decomposition

A **linear decomposition** [7], [10], [53], [54] can often further reduce the number of variables to represent incompletely specified index generation functions. In the linear decomposition shown in Fig. 4.1, L realizes linear functions, while G realizes indices. The cost for L is $O(np)$, while the cost for G is $O(q2^p)$, where $q \leq p \leq n$ and $q = \lceil \log_2(k+1) \rceil$. We assume that L is implemented by EXOR gates, multiplexers and registers [28], while G is implemented by a memory.

Definition 4.1: A **compound variables** has a form $y = c_1x_1 \oplus c_2x_2 \oplus \cdots \oplus c_nx_n$, where $c_i \in \{0,1\}$. The **compound degree** of the variable y is $\sum_{i=1}^{n} c_i$, where $\sum$ denotes an integer addition. A **primitive variable** is a variable with compound degree one.

Definition 4.2: For an incompletely specified index generation function f, the linear transformation that minimizes the number of variables to represent f is an **optimum transformation**.

The number of variables to represent a given function strongly influences the cost of the hardware. As for a lower bound on the number of variables, we have:

Theorem 4.1: [32] To represent an index generation function f with weight k, at least $LB = \lceil \log_2(k+1) \rceil$ compound variables are necessary.

In the above theorem, we assume that zero-output is used to denote non-registered vector. If an index generation function with weight k can be represented with $q = \lceil \log_2(k+1) \rceil$ compound variables, then the transformation is an optimum by Theorem 4.1.

In this part, we use m-out-of-n code to binary converters for benchmark functions.

Definition 4.3: The **m-out-of-n code** consists of $k = \binom{n}{m}$ binary codes with m 1's and $(n-m)$ 0's. The **m-out-of-n code to binary converter** realizes an index generation function with weight $k = \binom{n}{m}$, and has n inputs and $\lceil \log_2 \binom{n}{m} + 1 \rceil$ outputs. When the number of 1's in the input is not m, the circuit generates the code with all 0's.

Example 4.1: Consider the 1-out-of-7 code to binary converter shown in Table 4.1. We can reduce the number of variables by using the following linear transformation:

TABLE 4.1
1-out-of-7 Code to Binary Converter.

1-out-of-7 code							Index	Transformed Code		
x_1	x_2	x_3	x_4	x_5	x_6	x_7		y_1	y_2	y_3
1	0	0	0	0	0	0	1	1	0	1
0	1	0	0	0	0	0	2	0	0	0
0	0	1	0	0	0	0	3	0	1	1
0	0	0	1	0	0	0	4	0	1	0
0	0	0	0	1	0	0	5	0	0	1
0	0	0	0	0	1	0	6	1	0	0
0	0	0	0	0	0	1	7	1	1	0

$$\begin{aligned} y_1 &= x_1 \oplus x_6 \oplus x_7, \\ y_2 &= x_3 \oplus x_4 \oplus x_7, \\ y_3 &= x_1 \oplus x_3 \oplus x_5. \end{aligned}$$

The right three columns in Table 4.1 show the values of the compound variables y_i. In these columns, all the bit patterns are distinct. Thus, (y_3, y_2, y_1) represents the index generation function. Also, by Theorem 4.1, this function requires at least three compound variables. Thus, this is an optimum linear transformation. ■

As for an upper bound on the number of variables, we have

Conjecture 4.1: [28] When the number of the variables n is sufficiently large, most incompletely specified index generation functions with weight k (≥ 7) can be represented by

$$UB_{sasao} = 2\lceil \log_2(k+1) \rceil - 3$$

(primitive) variables.

The above result was obtained by experiments using many randomly generated functions, and statistical analysis for randomly generated functions. Unfortunately, it does not cover all the functions.

Recently, Jaakko Astola's group obtained an upper bound on the number of compound variables for all the index generation functions.

Theorem 4.2: [1] To represent an n-variable index generation function f with weight k,

$$UB_{astola} = 2\lceil \log_2(k+1) \rceil - 1 + \lceil \log_2(n-1) \rceil$$

compound variables are sufficient.

To derive the upper bound, a novel method was developed to reduce the number of compound variables [1]. This will be shown in Section VI.

V. Ambiguity-based Method

As shown in Example 4.1, with a linear transformation, we can often reduce the number of variables to represent the function. To distinguish 7 vectors, in the original domain, 7 variables are used, while in the transformed domain, only three variables are used. Note that, in the original domain, for each variable, 0's appear 6 times, while 1's appear just once. However, in the transformed domain, for each variable, 0's appear four times, while 1's appear three times. This suggests that variables with a balanced number of 0's and 1's tend to require fewer variables to distinguish vectors. Thus, we have the following [32]:

Definition 5.1: In the registered vector table, let $\nu(x_i, 0)$ be the number of vectors with $x_i = 0$, and let $\nu(x_i, 1)$ be the number of vectors with $x_i = 1$. The **imbalance measure** of the function with respect to x_i is defined as

$$\omega(x_i) = \nu(x_i, 0)^2 + \nu(x_i, 1)^2.$$

When the numbers of occurrences of 0's and 1's are the same in x_i, $\omega(x_i)$ is minimum. The larger the difference of the occurrences of 0's and 1's, the larger the imbalance measure.

Example 5.1: In Table 4.1, in the original domain, since for all x_i, $\nu(x_i, 0) = 6$ and $\nu(x_i, 1) = 1$, we have

$$\omega(x_i) = \nu(x_i, 0)^2 + \nu(x_i, 1)^2 = 6^2 + 1^2 = 37.$$

However, in the transformed domain, since $\nu(y_i, 0) = 4$ and $\nu(y_i, 1) = 3$, we have

$$\omega(x_i) = \nu(x_i, 0)^2 + \nu(x_i, 1)^2 = 4^2 + 3^2 = 25.$$

In other words, the linear transformation in Example 4.1 reduces the imbalance measure. ■

Variables with smaller imbalance measures tend to require fewer variables to distinguish the vectors. Thus, our strategy is to reduce the imbalance measure by a linear transformation. To find a minimal set of compound variables to represent the function, we use the following:

Definition 5.2: Let $f(x_1, x_2, \ldots, x_n)$ be an incompletely specified index generation function with weight k. Let $\vec{x} = (x_{i_1}, x_{i_2}, \ldots, x_{i_t})$ be a vector consisting of a subset of the variables $\{x_1, x_2, \ldots, x_n\}$. Let $N(f, \vec{x}, \vec{a})$ be the number of registered vectors of f that such that $\vec{x} = \vec{a} = (a_1, a_2, \ldots, a_t)$, $a_i \in \{0, 1\}$. The **ambiguity measure** of f with respect to $\vec{x}$ is defined as

$$AMB(\vec{x}) = -k + \sum_{\vec{a} \in B^t} N(f, \vec{x}, \vec{a})^2.$$

Theorem 5.1: $AMB(\vec{x}) = 0$ iff $\vec{x}$ can represent f.

Example 5.2: Consider the index generation function f shown in Table 1.1. In this case, $k = 4$. Let $\vec{x} = (x_1, x_2)$. Then,

$$\begin{aligned} N(f, \vec{x}, (0,0)) &= 1, \\ N(f, \vec{x}, (0,1)) &= 1, \\ N(f, \vec{x}, (1,0)) &= 1, \\ N(f, \vec{x}, (1,1)) &= 1. \end{aligned}$$

Thus, the ambiguity measure with respect to (x_1, x_2) is

$$AMB(x_1, x_2) = -4 + (1^2 + 1^2 + 1^2 + 1^2) = 0.$$

This shows that (x_1, x_2) can represent f.
Next, let $\vec{x} = (x_3, x_4)$. In this case,

$$\begin{aligned} N(f, \vec{x}, (0,0)) &= 2, \\ N(f, \vec{x}, (0,1)) &= 1, \\ N(f, \vec{x}, (1,0)) &= 1, \\ N(f, \vec{x}, (1,1)) &= 0. \end{aligned}$$

Thus, the ambiguity measure with respect to (x_3, x_4) is

$$AMB(x_3, x_4) = -4 + (2^2 + 1^2 + 1^2 + 0^2) = 2.$$

This shows that (x_3, x_4) cannot represent f. ■

The **ambiguity-based method** is a heuristic method using imbalance measure and ambiguity measure:

Algorithm 5.1: [32] (Heuristic minimization using imbalance and ambiguity measures.)

1) Let the input variables be $x_1, x_2, \ldots, x_n$. Let $t \geq 2$ be the maximal compound degree.
2) Generate the compound variables y_i whose compound degrees are t or less than t. The number of such compound variables is $\sum_{i=1}^{t} \binom{n}{i}$. Let T be the set of compound variables.
3) Let y_1 be the variable with the smallest imbalance measure. Let $\vec{Y} \leftarrow (y_1)$, $T \leftarrow T - y_1$.
4) While $AMB(\vec{Y}) > 0$, find the variable y_j in T that minimizes the value of $AMB(\vec{Y}, y_j)$. Let $\vec{Y} \leftarrow (\vec{Y}, y_j)$, $T \leftarrow T - y_j$.
5) Stop.

The memory requirement of Algorithm 5.1 is $O(n^t k)$. So, it works only when t is small. The current version of the program can accommodate $t \leq 6$.

VI. Polynomial-Based Method

Jaakko Astola et al. [1] developed **polynomial-based method** to reduce the number of compound variables.

For each registered vector $(x_1, x_2, \ldots, x_n)$, assign a polynomial on $GF(2)$:

$$X(s) = x_1 + x_2 s + x_3 s^2 + \ldots + x_n s^{n-1}.$$

Let

$$g(s) = g_1 + g_2 s + g_3 s^2 \ldots + g_p s^{p-1} + s^p$$

be a polynomial with degree p, where $g_i \in \{0, 1\}$, $(i = 1, 2, \ldots, p)$. For all $X(s)$ that correspond to the registered vectors, if all the remainders

$$R(s) = X(s) \bmod g(s)$$

are distinct, then the mapping

$$\Phi : X(s) \rightarrow R(s)$$

is injective (one-to-one). Since, Φ is linear, we have a desired linear decomposition.

Lemma 6.1: [1] Given an n-variable index generation function with weight k, there exists a polynomial $g(s)$ with the degree up to

$$p < 2\log_2 k + \log_2(n-1) - 1,$$

such that all the remainders $R(s)$ are distinct for the registered vectors.

From this, we have Theorem 4.2 and

Algorithm 6.1: (Polynomial-based method to reduce the number of compound variables)

1) Let $p \leftarrow \lceil \log_2(k+1) \rceil$.

2) For all possible polynomial $g(s)$ with degree p, do steps 3) and 4).
3) For every $X(s)$ that corresponds to a registered vector, compute the remainder

$$R(s) = X(s) \bmod g(s).$$

If there is no such polynomial $g(s)$, then increment p, and repeat step 2).

4) If all the remainders are different, then stop. [The given function is represented with p compound variables.]

The polynomial-based method is simple and fast. The time complexity of Algorithm 6.1 is $O(kn2^p)$. The performance of the algorithm will be shown in Section VIII.

Example 6.1: Consider the 7-variable index generation function shown in Table 6.1, which is the same function as Example 4.1. First, check if the polynomial-based method can represent this function with three variables. The column headed by $X(s)$ denotes the polynomial representations of the registered vectors. For each polynomial $g(s)$ of degree three, compute the remainders of registered vectors. When $g(s) = s^3 + s + 1$, all the remainders are distinct. The column headed by $R(s)$ denotes the remainders. This produces vectors shown in the right columns of Table 6.2. Since all the vector are distinct, three compound variables (y_1, y_2, y_3) represent this function. The linear transformation is

$$\begin{aligned} y_1 &= x_3 \oplus x_5 \oplus x_6 \oplus x_7, \\ y_2 &= x_2 \oplus x_4 \oplus x_5 \oplus x_6, \\ y_3 &= x_1 \oplus x_4 \oplus x_6 \oplus x_7. \end{aligned}$$

Since, $q = \lceil \log_2(k+1) \rceil = 3$, this is an optimal transformation. However, compared with Example 4.1, the compound degrees are larger. ■

TABLE 6.1
REGISTERED VECTOR, ITS POLYNOMIAL $X(s)$, AND ITS RESIDUE $R(s)$.

Original vectors							Index	$X(s)$	$R(s)$
x_1	x_2	x_3	x_4	x_5	x_6	x_7			
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	2	s	s
0	0	1	0	0	0	0	3	s^2	s^2
0	0	0	1	0	0	0	4	s^3	$s+1$
0	0	0	0	1	0	0	5	s^4	s^2+s
0	0	0	0	0	1	0	6	s^5	s^2+s+1
0	0	0	0	0	0	1	7	s^6	s^2+1

TABLE 6.2
REGISTERED VECTOR TABLE OF 7 VARIABLES, AND THEIR LINEAR TRANSFORMATION.

Original vectors							Index	Transformed		
x_1	x_2	x_3	x_4	x_5	x_6	x_7		y_1	y_2	y_3
1	0	0	0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	2	0	1	0
0	0	1	0	0	0	0	3	1	0	0
0	0	0	1	0	0	0	4	0	1	1
0	0	0	0	1	0	0	5	1	1	0
0	0	0	0	0	1	0	6	1	1	1
0	0	0	0	0	0	1	7	1	0	1

Example 6.2: Consider the 6-variable index generation function shown in Table 6.3. First, check if the polynomial-based method can represent this function with four variables. For any polynomial $g(s) = g_0 + g_1 s + g_2 s^2 + g_3 s^3 + s^4$ with degree four, there exist a pair of vectors that produce the same remainders. Thus, in the polynomial-based method, four variables are not sufficient, and five variables are necessary.

On the other hand, the exact method [24] (Algorithm 3.1) produced a solution with four primitive variables: x_1, x_3, x_5, x_6. As shown in the right columns of Table 6.3, all the 4-bit vectors are distinct. This illustrates the limitation of the polynomial-based method. ■

TABLE 6.3
REGISTERED VECTOR TABLE OF 6 VARIABLES.

x_1	x_2	x_3	x_4	x_5	x_6	Index	x_1	x_3	x_5	x_6
0	0	1	0	0	0	1	0	1	0	0
1	0	1	1	0	0	2	1	1	0	0
0	0	0	0	1	1	3	0	0	1	1
1	0	1	1	1	0	4	1	1	1	0
0	1	1	1	1	1	5	0	1	1	1
1	1	0	0	1	1	6	1	0	1	1
0	0	0	1	0	1	7	0	0	0	1
1	0	1	1	1	1	8	1	1	1	1
1	1	0	1	0	0	9	1	0	0	0
1	0	1	1	0	1	10	1	1	0	1

VII. SAT-BASED METHOD

The previous sections showed two reduction methods for compound variables. These method produce linear transformations quickly, but their optimality are not guaranteed.

In this part, we show an exact method to obtain an optimal linear transformation [44]. Since the method is very time and memory consuming, it is applicable only to small problems.

This method is based on:

Theorem 7.1: An incompletely specified index generation function is represented by p compound variables:

$$\begin{aligned} y_1 &= a_{1,1}x_1 \oplus a_{1,2}x_2 \oplus \cdots \oplus a_{1,n}x_n, \\ y_2 &= a_{2,1}x_1 \oplus a_{2,2}x_2 \oplus \cdots \oplus a_{2,n}x_n, \\ &\cdots \\ y_p &= a_{p,1}x_1 \oplus a_{p,2}x_2 \oplus \cdots \oplus a_{p,n}x_n. \end{aligned}$$

if and only if the values of $(y_1, y_2, \ldots, y_p)$ are all distinct for all registered vectors.

The **SAT-based method** uses the following approach: First obtain a good solution by a heuristic method [32], and then prove its minimality by a SAT solver [4].

Algorithm 7.1: (SAT-based method to minimize the number of compound variables)

1) Use a heuristic minimization algorithm to obtain a near minimum solution $S(p)$ with p compound variables.
2) Decrement p.
3) Use Theorem 7.1 to check if there is solution for the equations. If there is no solution, then stop. [UNSAT: $S(p+1)$ is the minimum solution.]
Otherwise, Go to step 2). [SAT: A better solution S(p) is found].

Example 7.1: Consider the 4-variable index generation function shown in Table 7.1. This function requires at least four variables even if any linear transformation is used. To

TABLE 7.1
REGISTERED VECTOR TABLE

x_1	x_2	x_3	x_4	index
1	0	0	0	1
0	1	0	0	2
0	0	1	0	3
0	0	0	1	4
1	0	0	1	5
0	1	1	0	6

prove this, assume that this function could be represented with only three compound variables:

$$\begin{aligned} y_1 &= a_{1,1}x_1 \oplus a_{1,2}x_2 \oplus a_{1,3}x_3 \oplus a_{1,4}x_4, \\ y_2 &= a_{2,1}x_1 \oplus a_{2,2}x_2 \oplus a_{2,3}x_3 \oplus a_{2,4}x_4, \\ y_3 &= a_{3,1}x_1 \oplus a_{3,2}x_2 \oplus a_{3,3}x_3 \oplus a_{3,4}x_4, \end{aligned}$$

where $a_{i,j} \in \{0,1\}$. No combination of $a_{i,j}$ makes (y_1, y_2, y_3) distinct for all the registered vectors. To prove this we use the following: The values of (y_1, y_2, y_3) for registered vectors are

1) $(a_{1,1}, a_{2,1}, a_{3,1})$
2) $(a_{1,2}, a_{2,2}, a_{3,2})$
3) $(a_{1,3}, a_{2,3}, a_{3,3})$
4) $(a_{1,4}, a_{2,4}, a_{3,4})$
5) $(a_{1,1} \oplus a_{1,4}, a_{2,1} \oplus a_{2,4}, a_{3,1} \oplus a_{3,4})$
6) $(a_{1,2} \oplus a_{1,3}, a_{2,2} \oplus a_{2,3}, a_{3,2} \oplus a_{3,3})$

Next, we need to check if there exist an assignment for $a_{i,j}$ that makes the values of these vectors all distinct. The condition that two vectors (a_1, a_2, a_3) and (b_1, b_2, b_3) are different is represented by the constraint:

$$(a_1 \oplus b_1) \vee (a_2 \oplus b_2) \vee (a_3 \oplus b_3) = 1.$$

There are $k = 6$ registered vectors. So the number of constraints is $\binom{6}{2} = 15$. The number of unknown coefficients is $np = 12$. By enumerating all $2^{12} = 4096$ assignments of values to $a_{i,j}$, we can show that no assignment satisfies these conditions at the same time. Thus, we can prove that at least four variables are necessary to represent the function. ■

To check the existence of the solutions by using Theorem 7.1, we have to search 2^{np} space. Especially, when the result is UNSAT (*i.e.*, there is no assignment that satisfies the constraints), the computation time would be very long. In such a case, we have to abort the computation. This method sometimes finds better solutions than heuristic algorithms, without showing the minimality.

Example 7.2: Consider the 2-out-of-8 code to binary converter. A ambiguity-based method [32] obtained a solution with 6 compound variables. The weight of this function is $k = \binom{8}{2} = 28$. Theorem 4.1 shows that to represent this function, at least five variables are necessary. Thus, if we can show that there is no solution with five compound variables, then the solution obtained by the ambiguity-based method [32] is optimum. Assume that this function could be represented with five compound variables $(y_1, y_2, y_3, y_4, y_5)$. In this case, we have $n = 8, p = 5$, and the number of unknown coefficients is $np = 40$. The condition that the values for $(y_1, y_2, y_3, y_4, y_5)$ are all distinct for all registered vectors, produces $\binom{28}{2} = 378$ constraints. The SAT solver shows that there is no solution (UNSAT). Thus, a 6-variable solution is an optimal. ■

Example 7.3: Consider the design of the 2-out-of-20 code to binary converter. The weight of this function is $k = \binom{20}{2} = 190$. The ambiguity-based method [32] obtained a solution with 9 compound variables. Theorem 4.1 shows that this function requires at least 8 variables to represent it. If there is no assignment that satisfies the constraints of Theorem 7.1, then the solution obtained by [32] is optimum.

Assume that this function could be represented with 8 compound variables:

$$\begin{aligned} y_1 &= a_{1,1}x_1 \oplus a_{1,2}x_2 \oplus \cdots \oplus a_{1,19}x_{19} \oplus a_{1,20}x_{20} \\ y_2 &= a_{2,1}x_1 \oplus a_{2,2}x_2 \oplus \cdots \oplus a_{2,19}x_{19} \oplus a_{2,20}x_{20} \\ &\cdots\cdots \\ y_8 &= a_{8,1}x_1 \oplus a_{8,2}x_2 \oplus \cdots \oplus a_{8,19}x_{19} \oplus a_{8,20}x_{20}. \end{aligned}$$

The values of vectors $(y_1, y_2, \ldots, y_8)$ for the registered vectors are

1) $(a_{1,1} \oplus a_{1,2}, a_{2,1} \oplus a_{2,2}, \ldots, a_{7,1} \oplus a_{7,2}, a_{8,1} \oplus a_{8,2})$
2) $(a_{1,1} \oplus a_{1,3}, a_{2,1} \oplus a_{2,3}, \ldots, a_{7,1} \oplus a_{7,3}, a_{8,1} \oplus a_{8,3})$
$\cdots$
190) $(a_{1,19} \oplus a_{1,20}, a_{2,19} \oplus a_{2,20}, \ldots, a_{7,19} \oplus a_{7,20}, a_{8,19} \oplus a_{8,20})$

We need to check if there is an assignment for $a_{i,j}$ that makes the values of these vectors distinct. Since the number of registered vectors is $k = \binom{20}{2} = 190$, the number of constraint is $\binom{190}{2} = 17955$. The total number of unknown coefficient is $np = 20 \times 8 = 160$, and the computation time is very long. To show that there is no solutions(UNSAT), it took nearly one full day. Thus, the 2-out-of-20 code to binary converter requires 9 variables. ■

Various methods can be used to reduce the search space. For example, if we restrict the maximal compound degree t, then the search space can be greatly reduced. Such a constraint can be written as

$$t \geq \sum_{j=1}^{n} a_{i,j},$$

for some appropriate choice for t. Also, the columns of the coefficient matrix $[a_{i,j}]$ must be linearly independent. Such restrictions can be represented by a set of equations.

VIII. COMPARISON OF MINIMIZATION METHODS

This part compares four different minimization methods.

1) Algorithm 3.1: The exact minimization method [24] to reduce primitive variables ($t = 1$). In many cases, this method finds a near minimum solution quickly, but takes a considerable time to prove its minimality. We abort the minimal cover in a time proportional to the time to construct the covering table.
2) Algorithm 5.1: The ambiguity-based method [32] to reduce compound variables with degree $2 \leq t \leq 6$.
3) Algorithm 6.1: The polynomial-based method [1] implemented by Sasao: It works only for two-valued input variables, but is much faster than the Astola's code.
4) Algorithm 7.1: The SAT-based method [44] that can reduce the number of compound variables and prove

TABLE 8.1
NUMBER OF VARIABLES TO REPRESENT RANDOMLY GENERATED FUNCTIONS.

n	k	$Exact$	$Polyno$	
		$t=1$	p	t
20	20	5.53	6.15	7.78
20	40	7.30	7.86	7.17
20	60	8.09	8.90	6.85
20	80	9.01	9.46	6.71
20	100	9.45	10.01	7.39
40	20	5.08	6.11	16.11
40	40	6.99	7.89	14.81
40	60	7.98	8.85	15.66
40	80	8.88	9.52	15.83
40	100	9.36	9.99	17.11
60	20	5.01	6.16	23.89
60	40	6.95	7.86	24.29
60	60	7.96	8.87	24.79
60	80	8.49	9.49	26.05
60	100	9.03	10.00	26.38

the minimality of the solutions. It works only for small problems.

To compare the performance of the algorithms, we used existing benchmark index generation functions [32].

A. Random Index Generation Functions

For different values of n and k, we generated 100 sample functions, and minimized the variables. Table 8.1 shows the results. n denotes the number of variables, and k denotes weight of the function. Since the sample functions are different from [1], the results are different. However, the differences are small. The column headed by *Exact* was obtained by the exact algorithm [24] using only primitive variables (i.e., $t = 1$).

The columns headed by *Polyno* denotes the results obtained by the polynomial-based method. The column headed by p denotes the average number of compound variables to represent the functions, while the column headed by t denotes the average compound degrees of the linear transformations.

The polynomial-based method required more variables with larger compound degrees than the exact method using only primitive variables. This is somewhat disappointing result. For this class of functions, the exact method [24] is more suitable than the polynomial-based method.

B. Constant-Weight Code to Binary Converters

The number of variables to represent an m-out-of-20 code to binary number converter is investigated for different values of compound degrees t, and for different values of m. The number of registered vectors is $k = \binom{20}{m}$, and by Theorem 4.1, the function requires at least $q = \lceil \log_2(k + 1) \rceil$ variables. Table 8.2 compares the results. When the compound degree is one ($t = 1$), all the converters required 19 variables. In this case, an exact method [24] was used to obtain the solutions. For the columns with $t \geq 2$, solutions were obtained by the ambiguity-based method [32]. For $m = 1$, $m = 2$ and $m = 3$, with the increase of the compound degree t, the necessary number of variables decreased. The entries with * marks denote optimum solutions proved by Theorem 4.1. The last two columns show results obtained by the polynomial-based method. The column headed by p shows the number of compound variables obtained by the polynomial-based method. For these functions, the polynomial-based method obtained very

TABLE 8.2
NUMBER OF VARIABLES TO REPRESENT m-OUT-OF-20 CODE TO BINARY CONVERTER: AMBIGUITY AND POLYNOMIAL-BASED METHODS.

		$Exact$	$Ambiguity - based$[32]					$Polyno$	
m	k	$t=1$	2	3	4	5	6	p	t
1	20	19	14	10	8	7	6	$*5$	10
2	190	19	15	12	10	9	9	9	8
3	1140	19	17	14	12	$*11$	$*11$	$*11$	7
4	4845	19	17	16	16	16	16	15	4

* denotes optimal solution proved by Theorem 4.1.

TABLE 8.3
NUMBER OF VARIABLES TO REPRESENT m-OUT-OF-20 CODE TO BINARY CONVERTER: SAT-BASED METHOD.

$Function$		$Exact$	$SAT - based$[44]						
m	k	$t=1$	2	3	4	5	6	7	8
1	20	#19	#**13**	#10	#8	#7	#6	#6	$*5$
2	190	#19	#**14**	#12	#10	#9	#9		
3	1140	#19	#**16**	**13**	12	12	$*11$		
4	4845	#19	#**16**	15	15	15	15		

* denotes optimal solution proved by Theorem 4.1.
# denotes optimal solution proved by the SAT-based method.

good solutions quickly. The last column headed by t shows the compound degrees for the solutions. The polynomial-based method obtained solutions with larger compound degrees than necessary.

Table 8.3 shows the result obtained by SAT-based method [44]. Entries in boldface denote solutions improved by the SAT-based method. Entries with # marks denote optimum solutions proved by the SAT-based method.

C. IP Address Tables

Distinct IP addresses of computers that accessed our web site over a period of a month were used. We considered four lists with different values of k. Table 8.4 shows the results. The original number of variables is $n = 32$. The first column shows the number of registered vectors: k. The second column shows the number of variables to represent the function, when only the primitive variables are used (i.e., $t = 1$). For this column, an exact method [24] was used to obtain solutions. The third column ($t = 2$) shows the number of variables to represent the function, when the variables with compound degrees up to two are used. Other columns show the numbers of variables for different values of t. The last two columns show the results obtained by the polynomial-based method. As for the quality of solutions, the polynomial-based method obtained solutions comparable to $t = 2$ or $t = 3$ of the ambiguity-based method [32]. However, the compound degrees are larger, which is a disadvantage of the polynomial-based method.

D. Lists of English Words

We used three lists of English words: *List A*, *List B*, and *List C*. The maximum number of characters in the word lists is 13, but we only consider the first 8 characters. For English words consisting of fewer than 8 letters, we append blanks to make

TABLE 8.4
NUMBER OF VARIABLES TO REPRESENT IP ADDRESS TABLE.

	$Exact$	$Ambiguity - based$[32]					$Polyno$	
k	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	p	t
1670	18	17	16	16	15	15	17	12
3288	20	19	18	17	17	17	19	10
4591	21	20	19	18	18	18	20	8
7903	23	21	20	20	20	20	22	7

TABLE 8.5
NUMBER OF VARIABLES TO REPRESENT LIST OF ENGLISH WORDS

		$Exact$	$Ambiguity-based$[32]				$Polyno$	
$Name$	k	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	p	t
$ListA$	1730	31	19	17	16	15	18	12
$ListB$	3366	31	21	19	17	17	19	15
$ListC$	4705	37	24	20	19	18	20	16

the length of words 8. We represent each alphabetic character by 5 bits. So, in the lists, all the words are represented by $n = 40$ bits. The number of words in the lists are 1730, 3366, and 4705, respectively. Within each word list, each English word has a unique index, an integer from 1 to k, where $k = 1730$ or 3360 or 4705. The numbers of bits for the indices are 11, 12, and 13, respectively. Table 8.5 shows the number of variables to represent the lists. For these data, we could reduce many variables by using compound variables with large t. The last two columns show the results with the polynomial-based method. In this case, the polynomial-based method obtained solutions comparable to $t = 3$ of the ambiguity-based method [32]. Again, the compound degrees obtained by the polynomial-based method are very large.

E. Computation Time

The ambiguity-based method [32] took 14.9 seconds to derive a 16-variable solution[1] for the minimization of the 4-out-of-20 code to binary converter with $t = 4$.

As for the polynomial-based method, the author developed a code that works only for two-valued input functions. To minimize the variables for the 4-out-of-20 code to binary converter by the polynomial-based method, this code took 249 milliseconds, while Astola's original code [1] took 167 seconds. Thus, for this function, this code is about 670 times faster than Astola's code.

One of the most time-consuming problems among the benchmark functions was the IP address table with $k = 7903$. The exact method to minimize primitive variables [24] took 13.5 seconds to derive a 23-variable solution; the polynomial-based method (this code) took 546 seconds to derive a 22-variable solution; and the ambiguity-based method [32] took 2.7, 26.6, 199, 1153, and 10939 seconds, for $t = 2$, $t = 3$, $t = 4$, $t = 5$ and $t = 6$, respectively. For the SAT-based method, to prove that 2-out-of-8 code to binary converter requires at least 6 variables (i.e., to show UNSAT) took 4004 seconds.

For the first three methods, we used a PC with INTEL Core i5 (2.6 GHz), and 8GB RAM, on Windows 10 Professional 64-bit Operating System. For the SAT-based method [44], we used a workstation with CPU:E5-2698 v3 (2.3GHz 16 cores) $\times 2$, 256 GB memory, 1TB HDD, with CentOS 6.5 operating system.

IX. OTHER WORKS

This part lists other works on index generation functions.

[1]We rewrote the code used in [32] to make it faster. So, the results here are different from the results in [32].

A. Minimization of variables

Logic function (pioneering works) [5], [6], [19].
Heuristic method [25], [41].
Iterative improvement [30], [42].
Fast algorithm [11].
Using autocorrelations [37], [49].
Using entropy functions [51].
Using non-linear transformations [2].
Using adder [33], [36].
Extension to multi-valued inputs [34]. [35].

B. Realization with multiple IGUs

Architecture [26], [45], [50].
Optimization using graph theory [8], [9].

C. Functional decomposition

Fast algorithms [46] [47], [48].

D. Analysis

[3], [19], [27], [39], [40], [43], [52].

E. Architecture

[12], [21], [22], [23], [50].

F. Applications

Virus scanning engine [13], [15].
IP look up [14], [16].

G. Survey

[20], [28], [29], [31], [38].

X. CONCLUSION AND COMMENTS

In this paper, we surveyed minimization methods for the number of variables in incompletely specified index generation functions.

For the minimization of primitive variables only, the exact minimization method [24] is fast. Its memory requirement is $O(nk^2)$ for an n variable index generation function with weight k.

For the minimization of compound variables with degrees up to t, where $2 \leq t \leq 6$, the ambiguity-based method [32] is practical. Its memory requirement is $O(n^t k)$, where t denotes the degree of compound variables. When the compound degree t is at most two or three, the computation time and memory requirement are moderate, and the cost for the linear circuit is also small.

The polynomial-based method [1] is quite different from existing methods. It quickly obtains a solution with a small number of variables, but the compound degrees tend to be very large. Its memory requirement is $O(nk)$. For m-out-of-n code to binary converters, the polynomial-based method obtains very good solutions. When k is large, it also obtains solutions with fewer variables than the ambiguity-based method[2][32].

The SAT-based method [44] is quite time and memory consuming to reduce compound variables, and can solve only

[2]Due to the space limitation, experimental results are omitted.

small problems. First, we used the ambiguity-based method to obtain a near minimum solution. Proving its minimality by a SAT solver (i.e., to show UNSAT) takes much longer time than finding solutions (i.e., to show SAT). In many cases, we have to abort the computation after a fixed CPU time. In such a case, we cannot guarantee the minimality of the solutions. The SAT-based method can be used to evaluate the quality of heuristic minimization programs.

Acknowledgments

This research is supported in part by the Grant in Aid for Scientific Research of the Japan Society for the Promotion of Science (JSPS). Many people were involved in this project: J. T. Butler, Y. Iguchi, M. Matsuura, H. Nakahara, S. Nagayama, I. Inoue, H. Iwamoto, and students of the Kyushu Institute of Technology and Meiji University. Reviewers' comments on my papers improved presentation. Finally, the author thanks R. Stankovic who provided encouragement to write this paper.

References

[1] J. Astola, P. Astola, R. Stankovic and I. Tabus, "An algebraic approach to reducing the number of variables of incompletely defined discrete functions," *International Symposium on Multiple-Valued Logic*, (ISMVL-2016), Sapporo, Japan, May 17-19, 2016, pp. 107-112.

[2] H. Astola, R. S. Stankovic, and J. T. Astola, "Index generation functions based on linear and polynomial transformations," *International Symposium on Multiple-Valued Logic*, (ISMVL-2016), Sapporo, Japan, May 17-19, 2016, pp. 102-106.

[3] J. T. Butler and T. Sasao, "Analysis of the number of variables to represent index generation functions,"*International Workshop on Boolean Problems*, Sept. 22, 2016, Freiberg, Germany, pp. 35-42.

[4] N. Een and N. Sorensson,"An extensible SAT-solver," *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing* (SAT-2003),pp. 502-518, 2003.

[5] C. Halatsis and N. Gaitanis, "Irredundant normal forms and minimal dependence sets of a Boolean function," *IEEE Trans. on Computers*, Vol. C-27, No. 11, pp. 1064-1068, November, 1978.

[6] Y. Kambayashi, "Logic design of programmable logic arrays," *IEEE Trans. on Computers*, Vol. C-28, No. 9, pp. 609-617, September 1979.

[7] R. J. Lechner,"Harmonic analysis of switching functions," in A. Mukhopadhyay (ed.), *Recent Developments in Switching Theory*, Academic Press, New York, 1971.

[8] Y. Matsunaga, "Synthesis algorithm of parallel index generation units," *Design, Automation & Test in Europe*, (DATE 2014), Dresden, March 2014, pp. 297.

[9] Y. Matsunaga, "Synthesis algorithm for parallel index generator," *IEICE Trans. Fund. Electronics, Communications and Computer Sciences*, Vol. E97-A, No. 12, pp. 2451-2458, Dec. 2014.

[10] 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.

[11] S. Nagayama, T. Sasao, and J. T. Butler, "An efficient heuristic algorithm for linear decomposition of index generation functions," *International Symposium on Multiple-valued Logic*, (ISMVL-2016), May 2016, pp. 96-101.

[12] H. Nakahara, T. Sasao and M. Matsuura, "A CAM emulator using look-up table cascades," *14th Reconfigurable Architectures Workshop*, RAW 2007, March 2007, Long Beach California, USA. CD-ROM RAW-9-paper-2.

[13] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "A parallel sieve method for a virus scanning engine," *12th EUROMICRO Conference on Digital System Design, Architectures, Methods and Tools*, Patras, Greece (*DSD-2009*), Aug. 2009, pp. 809-816.

[14] 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), March 25-27, 2013. Los Angeles.Also, in *Lecture Notes in Computer Science*, Vol. 7806, 2013, pp. 59-71.

[15] H. Nakahara, T. Sasao, and M. Matsuura, "A virus scanning engine using an MPU and an IGU based on row-shift decomposition," *IEICE Trans. Inf. and Sys.*, Vol. E96-D, No. 8, Aug. 2013, pp. 1667-1675.

[16] H. Nakahara,T. Sasao, M. Matsuura, H. Iwamoto, and Y. Terao," A memory-based IPv6 lookup architecture using parallel index generation units," *IEICE Trans. Inf. and Syst.* Vol. E98-D, No. 2, pp. 262-271, Feb. 2015.

[17] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, Vol. 41, No. 3, pp. 712-727, March 2006.

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

[19] T. Sasao, "On the number of dependent variables for incompletely specified multiple-valued functions," *International Symposium on Multiple-Valued Logic*, pp. 91-97, Portland, Oregon, U.S.A., May 23-25, 2000.

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

[21] T. Sasao, "A design method of address generators using hash memories," *International Workshop on Logic Synthesis* (IWLS-2006), Vail, Colorado, U.S.A, June 7-9, 2006, pp. 102-109.

[22] T. Sasao and M. Matsuura, "An implementation of an address generator using hash memories," *DSD-2007*, Aug. 27 - 31, 2007, Lubeck, Germany, pp. 69-76.

[23] T. Sasao and H. Nakahara, "Implementations of reconfigurable logic arrays on FPGAs," *International Conference on Field-Programmable Technology 2007* (ICFPT'07), Dec. 12-14, 2007, Kitakyushu, Japan, pp. 217-223.

[24] T. Sasao, "On the number of variables to represent sparse logic functions," *IEEE/ACM International Conference on Computer-Aided Design*, (ICCAD-2008),San Jose, California, USA, Nov.10-13, 2008, pp. 45-51.

[25] 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), Aug. 2009, pp. 765-772.

[26] T. Sasao, M. Matsuura, and H. Nakahara, "A realization of index generation functions using modules of uniform sizes," *International Workshop on Logic and Synthesis* (IWLS-2010), Irvine, California, June 11-13, 2010, pp. 201-208.

[27] T. Sasao, "On the numbers of variables to represent multi-valued incompletely specified functions," *13th EUROMICRO Conference on Digital System Design, Architectures, Methods and Tools*, (DSD-2010), Lille, France, Sept. 2010, pp. 420-423.

[28] T. Sasao, *Memory Based Logic Synthesis*, Springer, 2011.

[29] T. Sasao,"Index generation functions: Recent developments," (invited paper), *International Symposium on Multiple-Valued Logic* (ISMVL-2011), Tuusula, Finland, May 23-25, 2011, pp. 1-9.

[30] T. Sasao,"Linear Transformations for Variable Reduction,"*Reed-Muller Workshop* (RM-2011), Tuusula, Finland, May 25, 2011.

[31] T. Sasao, "Linear decomposition of logic functions: Theory and applications,"*International Workshop on Logic and Synthesis*, (IWLS-2011), San Diego, June 3 - 5, 2011.

[32] T. Sasao, "Linear decomposition of index generation functions," *Asia and South Pacific Design Automation Conference* (ASPDAC-2012), Jan. 30–Feb. 2, 2012, Sydney, Australia, pp. 781-788.

[33] T. Sasao, "Row-shift decompositions for index generation functions," *Design, Automation and Test in Europe*, (DATE-2012), March 12-16, 2012, Dresden, Germany, pp. 1585-1590.

[34] T. Sasao, "Multi-valued input index generation functions: Optimization by linear transformation," *International Symposium on Multiple-Valued Logic*, (ISMVL-2012), May 14-16, Victoria, BC, Canada, pp. 185-190.

[35] T. Sasao, "Multiple-valued index generation functions: Reduction of variables by linear transformation,"*Journal of Multiple-Valued Logic and Soft Computing*, Vol. 21, No. 5-6, pp. 541-559, 2013.

[36] T. Sasao, "Cyclic row-shift decompositions for incompletely specified index generation functions," *International Workshop on Logic and Synthesis*, (IWLS-2013), Austin, Texas, June 2013, pp. 131-137.

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

[38] T. Sasao, “Index generation functions: Tutorial,”*Journal of Multiple-Valued Logic and Soft Computing*, Vol. 23, No. 3-4, pp. 235-263, 2014.

[39] T. Sasao, Y. Urano and Y. Iguchi, “A lower bound on the number of variables to represent incompletely specified index generation functions,” *International Symposium on Multiple-Valued Logic*, (ISMVL-2014), May 19, 2014, Bremen, Germany, pp. 7-12.

[40] T. Sasao, “On the average number of variables to represent incompletely specified index generation functions,” *International Workshop on Logic and Synthesis*, (IWLS-2014), San Francisco, May 30-June 1, 2014, pp. 1-7, (Paper 1-4).

[41] T. Sasao, Y. Urano, and Y. Iguchi, “A method to find linear decompositions for incompletely specified index generation functions using difference matrix,” *IEICE Trans. on Fundamentals of Electronics, Communication and Computer Sciences*, Vol. E97-A, No. 12, pp. 2427-2433, Dec. 2014.

[42] T. Sasao, “A reduction method for the number of variables to represent index generation functions: s-Min method,” *International Symposium on Multiple-Valued Logic*, (ISMVL-2015), May 18-20, 2015, Waterloo, Canada, pp. 164-169.

[43] T. Sasao, “On the sizes of reduced covering tables for incompletely specified index generation functions,” *Reed-Muller Workshop 2015*, May 21, 2015, Waterloo, Ontario, Canada.

[44] T. Sasao, I. Fumishi, and Y. Iguchi, “A method to minimize variables for incompletely specified index generation functions using a SAT solver,” *International Workshop on Logic and Synthesis*, (IWLS-2015), Mountain View, June 12-13, 2015, pp. 161-167.

[45] T. Sasao, “A realization of index generation functions using multiple IGUs,” *International Symposium on Multiple-Valued Logic*, (ISMVL-2016), Sapporo, Japan, May 17-19, 2016, pp.113-118.

[46] T. Sasao and J. T. Butler, “Decomposition of index generation functions using a Monte Carlo method,”*International Workshop on Logic and Synthesis*, (IWLS-2016), June 10-11, Austin, Texas, USA, pp. 171-176.

[47] T. Sasao, K. Matsuura, Y. Iguchi,“A heuristic decomposition of index generation functions with many variables,”*Workshop on Synthesis And System Integration of Mixed Information Technologies* (SASIMI-2016), Kyoto, Oct. 24, 2016.

[48] T. Sasao, K. Matsuura, and Y. Iguchi, “An algorithm to find optimum support-reducing decompositions for index generation functions,” *Design Automation and Test in Europe*, (DATE-2017), March 27-31,2017, Lausanne, Switzerland.

[49] T. Sasao, “A linear decomposition of index generation functions: Optimization using autocorrelation functions,”*Journal of Multiple-Valued Logic and Soft Computing*, Vol. 28, No. 1, pp.105-127, 2017.

[50] T. Sasao, “A fast updatable implementation of index generation functions using multiple IGUs,” *IEICE Trans. Inf. and Syst.* Vol.Exxx, No. xx, pp. xxx, xxx., 2017.(to appear).

[51] D. A. Simovici, D. Pletea, and R. Vetro, “Information-theoretical mining of determining sets for partially defined functions,” *International Symposium on Multiple-Valued Logic*, (ISMVL-2010), May 2010, pp. 294-299.

[52] D. A. Simovici, M. Zimand, and D. Pletea, “Several remarks on index generation functions,” *International Symposium on Multiple-Valued Logic*, (ISMVL-2012), May 2012, pp. 179-184.

[53] R. S. Stankovic and J. Astola (eds.) E.I. Nechiporuk, ”Network synthesis by using linear transformation of variables,” in *Reprints from the Early Days of Information Sciences*, Tampere International Center for Signal Processing, Tampere 2007.

[54] D. Varma and E. Trachtenberg, “Design automation tools for efficient implementation of logic functions by decomposition,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8, No. 8, pp. 901-916, 1989.