A Method to Minimize Variables for Incompletely Specified Index Generation Functions Using a SAT Solver

Tsutomu Sasao, Ichidou Fumishi, and Yukihiro Iguchi
Department of Computer Science, Meiji University
Kanagawa 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 further be 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 shows a method to minimize the number of variables by using a SAT solver. With this technique, we obtained better solutions than existing methods.**

***Keywords*-Incompletely specified function, Index generation function, Linear transformation, Variable minimization, SAT solver, Symmetric function**

I. Introduction

Index generation functions [6] are useful for computer virus scanning engines and packet filters in the internet. An index generation function can be efficiently implemented by the **index generation unit (IGU)** [6] shown in Fig. 1.1.

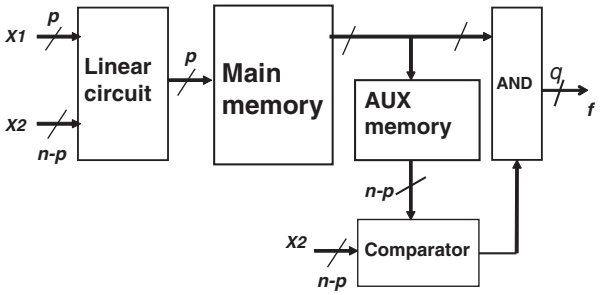

Figure 1.1. Index Generation Unit (IGU).

In Fig. 1.1, the **linear circuit** realizes linear functions, while the **main memory** realizes an index. When the given function is defined for only k input combinations, and $k \ll 2^n$, p, the number of variables for the main memory often can be reduced, where n denotes the total number of inputs to the IGU. When we use the linear transformation that minimizes p, the size of the main memory can be reduced drastically. For the fixed linear transformation, the minimization of variables to the main memory can be performed by a minimum covering [2], [3]. However, when the linear transformation can be chosen freely, the problem becomes very difficult, since the number of the linear transformations to consider is very large. In this paper, we show a method to find an optimum linear transformation using a SAT solver.

Table 2.1
Registered Vector Table

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

The rest of this paper is organized as follows: Section 2 defines index generation functions; Section 3 introduces a method to reduce the number of variables for an incompletely specified function; Section 4 presents a method to reduce the number of variables using a linear decomposition; Section 5 formulate the problem using a SAT solver; Section 6 shows a method to reduce the search space for general functions; Section 7 shows a method to reduce the search space for symmetric functions; Section 8 shows experimental results; and finally, Section 9 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 is undefined (*don't care*). An incompletely specified index generation function shows a mapping $M \rightarrow \{1, 2, \ldots, k\}$, where $M \subset B^n$ represent a set of registered vectors. k is the **weight** of the function.

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

III. Number of Variables to Represent an Incompletely Specified Index Generation Function

An incompletely specified function f can often be represented with fewer variables than the original function, when *don't care* values are assigned property to 0 or 1.

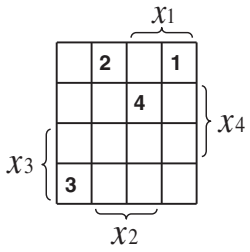

Figure 3.1. 4-variable index generation function.

This property is useful to realize functions by look-up tables (LUTs).

Theorem 3.1: Assume that an incompletely specified function f is represented by a decomposition chart [4]. If each column of the decomposition chart has at most one care element, then the function can be represented by only column variables.

(Proof) If a column has a *care* element, then set the values of the *don't cares* to the same value to that of the *care* element. In this case, the function depends only on column variables. □

Example 3.1: Consider the decomposition chart in Fig. 3.1. x_1 and x_2 specify columns, while x_3 and x_4 specify rows. Also, blank cells denote *don't cares*. In Fig. 3.1, each column has at most one care element. Thus, this function can be represented with only the column variables x_1 and x_2:

$$F = 1 \cdot x_1\bar{x}_2 \vee 2 \cdot \bar{x}_1x_2 \vee 3 \cdot \bar{x}_1\bar{x}_2 \vee 4 \cdot x_1x_2.$$

■

A lower bound on the number of variables to represent a function is obtained as follows:

Theorem 3.2: [7] To represent an index generation function f with weight k, at least $LB = \lceil \log_2(k+1) \rceil$ variables are necessary.

In the above theorem, we assume that zero-output is used to denote non-registered vector.

IV. Linear Decomposition

A **linear decomposition** 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 and multiplexers [6], 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 ordinary integer addition, and c_i is treated as an integer. A **primitive variable** is a variable with compound degree one.

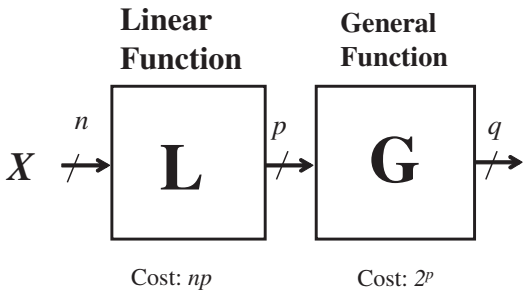

Figure 4.1. Linear Decomposition

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**.

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 3.2. As a method to obtain an optimum transformation, one might consider the following: First, generate all the compound variables. Next, minimize the variables using the method in [2], [3]. Unfortunately, this method requires too much computation time and excessive memory space.

V. SAT-based Formulation

Assume that an IGU can implement arbitrary linear transformations. Then, the size of the main memory can be minimized by using the linear transformation that minimizes p, the number of the compound variables. To find such a linear transformation, we need to check if the function can be represented or not, for all possible linear transformations. This would be very time consuming. Here, we use the following approach: first obtain a good solution by a heuristic algorithm [7], and then prove its minimality by a SAT solver [1].

Theorem 5.1: 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.

Example 5.1: Consider the 4-variable index generation function shown in Table 5.1. This function requires at least 4 variables even if any linear transformation is used. To prove this, assume that this function can be represented with only

three compound variables:

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

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

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

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 using a SAT solver, we can show that no assignment satisfies these conditions at the same time. ∎

To check the existence of the solutions by using Theorem 5.1, 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. Thus, we need other constraints to reduce the search space.

In this paper, we use m-out-of-n code to index converters for benchmark functions, since

1) Various functions can be generated for different values of m and n, where $m \leq n$.
2) The necessary number of variables reduces with the increase of the compound degree.
3) They have mathematical property that can be analyzed easily.

Definition 5.1: The **m-out-of-n code** consists of $k = \binom{n}{m}$ binary codes with weight m. The **m-out-of-n code to index 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. The m-out-of-n code is produced in ascending order. That is, the minimum index corresponds to $(0, 0, \ldots, 0, 1, 1, \ldots, 1)$, while the maximum index corresponds to $(1, 1, \ldots 1, 0, 0, \ldots, 0)$.

Table 5.1
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

Table 5.2
2-OUT-OF-6 CODE TO INDEX CONVERTER.

2-out-of-6 code						Index	After Linear Transform			
x_6	x_5	x_4	x_3	x_2	x_1		y_4	y_3	y_2	y_1
0	0	0	0	1	1	1	0	0	0	1
0	0	0	1	0	1	2	0	0	1	1
0	0	0	1	1	0	3	0	0	1	0
0	0	1	0	0	1	4	0	1	1	0
0	0	1	0	1	0	5	0	1	1	1
0	0	1	1	0	0	6	0	1	0	1
0	1	0	0	0	1	7	1	1	0	0
0	1	0	0	1	0	8	1	1	0	1
0	1	0	1	0	0	9	1	1	1	1
0	1	1	0	0	0	10	1	0	1	0
1	0	0	0	0	1	11	1	0	0	0
1	0	0	0	1	0	12	1	0	0	1
1	0	0	1	0	0	13	1	0	1	1
1	0	1	0	0	0	14	1	1	1	0
1	1	0	0	0	0	15	0	1	0	0

Example 5.2: Consider the 2-out-of-6 code to index converter shown in Table 5.2. We can reduce the number of variables by using the following linear transformation:

$$\begin{aligned} y_4 &= x_6 \oplus x_5, \\ y_3 &= x_5 \oplus x_4, \\ y_2 &= x_4 \oplus x_3, \\ y_1 &= x_3 \oplus x_2. \end{aligned}$$

The right four columns in Table 5.2 show the values of the compound variables y_i. All the possible 4-bit non-zero patterns appear in the right columns, and all the patterns are distinct. By Theorem 5.1, (y_4, y_3, y_2, y_1) represents the index generation function. Also, by Theorem 3.2, this function requires at least four compound variables. Thus, this is an optimum linear transformation. ∎

Example 5.3: Consider the 2-out-of-8 code to index converter. A heuristic algorithm [7] obtained a solution with 6 compound variables. The weight of this function is $k = \binom{8}{2} = 28$. Theorem 3.2 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 algorithm in [7] is optimum. Assume that this function can 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. ∎

Example 5.4: Consider the design of the 2-out-of-20 code to index converter. The weight of this function is $k = \binom{20}{2} = 190$. A heuristic program [7] obtained a solution with 9 compound variables. Theorem 3.2 shows that this function requires at least 8 variables to represent it. If there is no assignment that satisfies the constraints of Theorem 5.1, then the solution obtained by [7] is optimum.

Assume that this function can 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) $(a_{1,1} \oplus a_{1,3}, a_{2,1} \oplus a_{2,3}, \ldots, a_{7,1} \oplus a_{7,3}, a_{8,1} \oplus a_{8,3})$
...
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 would be too long. Thus, we need some methods to reduce the search space. ■

VI. Reduction of Search Space for General Functions

In Section V, we showed a method to decide if a given n-variable incompletely specified index generation function can be represented by p compound variables or not. This method uses np unknown coefficients. When the value of np is large, the computation time would be very large. This section shows some methods to reduce the search space.

Theorem 6.1: Consider the problem to check if an index generation function f can be represented with p compound variables in Theorem 5.1. Then, we need only to check the combinations such that $A_1 > A_2 > \ldots > A_p$, where $A_i = \sum_{j=1}^{n} 2^{n-j} a_{i,j}$.

(Proof) Assume that an index generation function can be represented by p compound variables y_i $(i = 1, 2, \ldots, p)$. It is clear that any permutation of compound variable y_i does not change the number of variables necessary to represent the function. Thus, when $(a_{i,1}, a_{i,2} \ldots a_{i,p})$ represent an integer, we can assume that the compound variables are in the descending order of their values. □

When we check if the function can be represented with p compound variables or not, Theorem 6.1 reduces the search space by $p!$.

Example 6.1: Consider the problem in Example 5.1. We need only to consider the case

$$A_1 > A_2 > A_3,$$

where

$$\begin{aligned} A_1 &= 8a_{1,1} + 4a_{1,2} + 2a_{1,3} + a_{1,4} \\ A_2 &= 8a_{2,1} + 4a_{2,2} + 2a_{2,3} + a_{2,4} \\ A_3 &= 8a_{3,1} + 4a_{3,2} + 2a_{3,3} + a_{3,4}. \end{aligned}$$

■

Theorem 6.2: Let $\{y_1, y_2, \ldots y_p\}$ be a minimal set of compound variables to represent an incompletely specified index generation function f. Then, $y_1, y_2, \ldots y_p$ is linearly independent.

Theorem 6.3: Assume that p compound variables $y_1, y_2, \ldots, y_p$ in Theorem 5.1 are linearly independent. Then, the number of different tuples $(y_1, y_2, \ldots, y_p)$ is

$$\lambda(n, p) = \prod_{i=0}^{p-1} (2^n - 2^i).$$

Example 6.2: Consider the case of $n = 4$ and $p = 3$. The number of unknown coefficients is $np = 12$. Thus, there are $2^{12} = 4096$ combinations to consider. With the linear independence, by Theorem 6.3 the number of combinations to consider is reduced to

$$(16 - 2^0)(16 - 2^1)(16 - 2^2) = 15 \times 14 \times 12 = 2880.$$

Furthermore, by Theorem 6.1, the number of combinations to consider is reduced by $p!$. Thus, we need to consider for only

$$\frac{2880}{3!} = 480$$

combinations. ■

To generate the constraints for the linear independence is not easy. However, we can easily generate the conditions for:

1) All the compound variables are distinct.
2) Among any triple of compound variables, there is no dependence.

VII. Reduction of Search Space for Symmetric Functions

Definition 7.1: Consider an incompletely specified index generation function of n variables:

$$F : M \to \{1, 2, \ldots, k\}, M \subset \{0, 1\}^n.$$

The corresponding **characteristic logic function** is

$$\chi : \{0, 1\}^n \to \{0, 1\},$$

where

$$\chi(\vec{x}) = \begin{cases} 1 & (\vec{x} \in M) \\ 0 & (\text{Otherwise}). \end{cases}$$

A **symmetric index generation function** is an index generation function whose characteristic function is symmetric.

Note that m-out-of-n to index converters represent symmetric index generation functions. In a symmetric index generation function, any permutation of the primitive variables x_i does not change the number of variables necessary to represent the function.

Theorem 7.1: Consider the problem to represent a symmetric index generation function f with p compound variables. Let S_n be the set of all the permutations of n elements.

Then, S_n is a **symmetric group with degree** n, and has $n!$ permutations. Assume that i-th permutation of S_n maps the element j into $\pi(i,j)$, where $i \in \{1,2,\ldots,n!\}$ and $j \in \{1,2,\ldots,n\}$. To check if f can be represented by p compound variables, we need only consider the case

$$C(1) > C(2) > \cdots > C(n!),$$

where

$$C(i) = \sum_{j=1}^{n} 2^{p(n-j)} B(i,j),$$

$$B(i,j) = 2^{p-1} a_{1,\pi(i,j)} + 2^{p-2} a_{2,\pi(i,j)} + \ldots + 2^0 a_{p,\pi(i,j)},$$

and $a_{i,j}$ are unknown coefficients defined in Theorem 6.1

Example 7.1: Consider the case of $n = p = 3$. Assume that the compound variables

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

gives a minimum solution. In this case, the coefficient matrix of the linear transformation is

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

When the index generation function is symmetric, the matrices whose columns are permutated

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

also produce minimum solutions. In Theorem 6.1, when the $n^2 = 9$ bit vector

$$(a_{1,1}, a_{2,1}, a_{3,1}, a_{1,2}, a_{2,2}, a_{3,2}, a_{1,3}, a_{2,3}, a_{3,3})$$

is considered as an integer, we need only to consider the matrix that has the maximal value (*i.e.*, the last matrix). ■

For an n variable symmetric index generation function, the search space is reduced by $n!$ using Theorem 6.1.

Definition 7.2: [9] Let $f(X)$ be an incompletely specified index generation function, where $X = \{x_1, x_2, \ldots, x_n\}$ denotes the set of variables for f. Let X_1 be a proper subset of X, and let $\vec{X}_1$ be an ordered set of X_1. In this case, $\vec{X}_1$ is a **partial vector** X. When the value of $\vec{X}_1$ is set to $\vec{a} = (a_1, a_2, \ldots, a_t)$, $a_i \in B$, the number of registered vectors that make the value of f non-zero is denoted by $N(f, \vec{X}_1, \vec{a})$, where $B = \{0,1\}$. Let $\vec{X}_1$ be a partial vector of X. Then,

$$CD(f : X_1) = \max_{\vec{a} \in B^t} \left\{ N(f : \vec{X}_1, \vec{a}) \right\}$$

is the **collision degree**, where t denotes the number of variables in X_1.

Example 7.2: Consider the index generation function f shown in Table 2.1.

$$\begin{aligned} CD(f : \{x_1, x_2\}) &= \max\{|\phi|, |\{2,3\}|, |\{1\}|, |\{4\}|\} = 2, \\ CD(f : \{x_3, x_4\}) &= \max\{|\{1,2\}|, |\{4\}|, |\{3\}|, |\phi|\} = 2, \\ CD(f : x_1) &= \max\{|\{2,3\}|, |\{1,4\}|\} = 2, \\ CD(f : x_2) &= \max\{|\{1\}|, |\{2,3,4\}|\} = 3, \\ CD(f : x_3) &= \max\{|\{1,2,4\}|, |\{3\}|\} = 3, \\ CD(f : x_4) &= \max\{|\{1,2,3\}|, |\{4\}|\} = 3. \end{aligned}$$

■

Theorem 7.2: [9] Let $f(X)$ be an incompletely specified index generation function, and let X_1 be a proper subset of X. Then, to represent f, at least $\lceil \log_2 CD(f : X_1) \rceil$ (compound) variables are necessary to represent f, in addition to the variables in X_1.

Corollary 7.1: Let $f(X)$ be an incompletely specified index generation function, and let x_i be a variable in X. Then, to represent f, at least $\lceil \log_2 CD(f : x_i) \rceil$ (compound) variables are necessary in addition to the variable x_i.

Example 7.3: To represent the 2-out-of-8 code to index converter f with five variables, the compound degree of each variable must be at least two. On the contrary, assume that the minimum solution contained a variable x_1 with the compound degree one. In this case, $CD(f : x_1) = 21$. Thus, to represent f, at least $1 + \lceil \log_2 21 \rceil = 1 + 5 = 6$ variables are necessary to represent the function by Corollary 7.1. ■

Example 7.4: In order to represent the 2-out-of-20 code to index converter f by 8 variables, the compound degrees of each variable must be at least four. Let the variable with the compound degree 1 be $y_1 = x_1$. Let the variable with the compound degree 2 be $y_2 = x_1 \oplus x_2$. Let the variable with the compound degree 3 be $y_3 = x_1 \oplus x_2 \oplus x_3$.

1) When the solution contains y_1, $CD(f : y_1) = 171$.
2) When the solution contains y_2, $CD(f : y_2) = 153$.
3) When the solution contains y_3, $CD(f : y_3) = 136$.

Assume that the solution contained either y_1, y_2, or y_3. Then, Corollary 7.1 shows that at least $1 + \lceil \log_2 CD(f : y_i) \rceil = 1 + 8 = 9$ variables are necessary to represent f. Thus, to represent the function with 8 variables, the degrees of the compound variables must be greater than three. ■

Theorem 7.3: To represent an m-out-of-n code to index converter, we need only to consider the compound variables of degrees with at most $\lceil \frac{n}{2} \rceil$.

Example 7.5: To represent the 2-out-of-8 code to index converter using 8 compound variables, we need only to consider the variables of compound degree with at most four. Also, from Example 7.3, the compound degrees of the variables are at least two. Thus, we can reduce the search

Table 8.1
NUMBER OF VARIABLES TO REPRESENT RANDOMLY GENERATED FUNCTIONS BY ASPDAC2012 METHOD[7] AND SAT-BASED METHOD.

	Heuristic			SAT-Based		
k	$n=16$	$n=20$	$n=24$	$n=16$	$n=20$	$n=24$
7	$*3.00$	$*3.00$	$*3.00$	$*3.00$	$*3.00$	$*3.00$
15	4.66	4.53	4.33	$*4.00$	$*4.00$	$*4.00$
31	6.08	6.00	6.01	6.00	5.99	6.00
63	8.06	8.02	7.90	7.94	7.94	7.78
127	10.13	9.89	9.77	9.88	9.79	9.70
255	12.15	11.86	11.71	11.99	11.83	11.70
511	14.52	14.02	13.60	14.00	13.96	13.60

space by the constraints

$$2 \leq \sum_{j=1}^{8} a_{i,j} \leq 4.$$

■

Example 7.6: To represent the 2-out-of-20 code to index converter, we need only to consider the variables of the compound degrees with at most 10. Also, by Example 7.4, the degrees of the variables are at least four. Thus, we can use the constraint

$$4 \leq \sum_{j=1}^{20} a_{i,j} \leq 10$$

to reduce the search space. ■

VIII. EXPERIMENTAL RESULTS

A. Minimization System

A minimization system for index generation functions was developed on the top of *SUGAR* [10] (a SAT-based constraint solver), *MiniSat* [1] and *GlueMiniSat* [11] (SAT solvers). A tool to convert an index generation function into the data for SUGAR was developed by ourselves.

We used a workstation with CPU:E5-2698 v3 (2.3GHz 16 cores)×2, 128 GB memory, 1TB HDD, and CentOS 6.5.

When the result is SAT, the CPU time is short (less than 5 minutes), while when the result is UNSAT, the CPU time is very long. So, we aborted the computation after a fixed CPU time (10 minutes). For small problems, we can prove the exact minimality of the solutions.

B. Randomly Generated Functions

The numbers of variables to represent randomly generated n-variable functions with weight k were investigated for different values of n and k, when $t = 2$, *i.e.*, when we used variables with compound degree two.

Table 8.1 compares results by a heuristic method [7] with the SAT-based method. They are average of 100 randomly generated functions. The SAT-based method obtained exact minimum solutions for $k = 7$ and $k = 15$ (* marks). For $k \geq 31$, the SAT-based method could not finish the computation for UNSAT within the allocated time, so the results may not be exact minimum. As for the SAT solver, used MiniSat.

Table 8.2
NUMBER OF VARIABLES TO REPRESENT m-OUT-OF-16 CODE TO INDEX CONVERTERS OBTAINED BY ASPDAC2012 METHOD[7].

Function		Compound degree					
m	k	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
1	16	15	11	8	6	5	5
2	120	15	12	9	8	8	8
3	560	15	14	11	10	10	10
4	1820	15	14	13	13	13	13

Table 8.3
NUMBER OF VARIABLES TO REPRESENT m-OUT-OF-16 CODE TO INDEX CONVERTERS OBTAINED BY SAT-BASED METHOD.

Function		Compound degree					
m	k	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
1	16	$*15$	**10**	8	6	$*5$	$*5$
2	120	$*15$	**11**	9	$*8$	8	8
3	560	$*15$	**13**	11	$*10$	$*10$	$*10$
4	1820	$*15$	**13**	13	13	13	13

C. m-out-of-n Code to Index Converters

The numbers of variables to represent m-out-of-16 code to index converters were investigated for different values of t and m, where t denotes the maximum compound degree. The number of registered vectors is $k = \binom{16}{m}$. Thus, by Theorem 3.2, the function requires at least $q = \lceil \log_2(k+1) \rceil$ variables. Table 8.2 shows the results by a heuristic method [7], while Table 8.3 shows the results by the SAT-based method. In Table 8.3, the figures shown in bold face denote better solutions than the heuristic method, and entries with * marks denote exact minimum: For $t = 1$, they are proved by the property of one-hot codes; for $m = 1$ and $m = 3$, they are proved by Theorem 3.2; and for $m = 2$, it is proved by Theorem 7.2. As for the SAT solver, used GlueMiniSat.

IX. CONCLUSION AND COMMENTS

In this paper, we

- Showed an exact method to minimize the number of variables for incompletely specified index generation functions, when the linear transformations can be freely chosen.
- Showed methods to reduce search space for general index generation functions and symmetric index generation functions.
- Derived solutions for m-out-of-16 code to index converters. For some cases, we found better solutions than a heuristic program.

Although the presented exact minimizer is quite time and memory consuming and is applicable to only small problems, it can evaluate the quality of heuristic minimization programs such as [7].

In the program [7], if we limit the compound degree t to two or three, then the computation time and memory requirement would be much smaller. In this case, the cost of the linear circuit is also reduced.

The best strategy is first use a heuristic minimizer to find an initial solution, and then use the SAT-based minimizer to

find a better solution. In many cases, the heuristic minimizer obtains exact minimum solutions. Proving the minimality (UNSAT) takes longer time than finding a solution (SAT). So, we can abort the computation after a fixed CPU time.

For the applications where frequent changes are necessary, a heuristic algorithm rather than the presented one should be used.

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).

REFERENCES

[1] 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.

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

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

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

[5] T. Sasao, "On the number of variables to represent sparse logic functions,"*ICCAD-2008*, San Jose, California, USA, Nov. 10-13, 2008, pp. 45-51.

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

[7] T. Sasao, "Linear decomposition of index generation functions," *17th Asia and South Pacific Design Automation Conference* (ASPDAC-2012), Jan. 30- Feb. 2, 2012, Sydney, Australia, pp. 781-788.

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

[9] T. Sasao, "A reduction method for the number of variables to represent index generation functions: s-Min method," *45th International Symposium on Multiple-Valued Logic* (ISMVL-2015), May 18-20, 2015, Waterloo, Canada, (to appear).

[10] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, "Compiling finite linear CSP into SAT,"*Constraints*, Vol. 14, No. 2, pp. 254-272, June, 2009.

[11] https://sites.google.com/a/nabelab.org/glueminisat/