A Heuristic Decomposition of Index Generation Functions with Many Variables

Tsutomu Sasao, Kyu Matsuura, and Yukihiro Iguchi

Department of Computer Science, Meiji University,
Kawasaki, 214-8571, Japan

Abstract— This paper shows a heuristic method to decompose index generation functions with many variables. Three different measures are used to select the bound variables. Experimental results shows that this method finds fairly good decompositions in a short time. Comparison with Monte Carlo method is presented.

I. Introduction

Index generation functions [9] are used for access control lists, routers, and virus scanning for the internet, etc.. They represent functions of Content Addressable Memories (CAMs) used for pattern matching applicatoins.

Functional decomposition [1, 3] is a technique to decompose a circuit into two parts with a lower cost than the original circuit. Various techniques to find functional decompositions have been presented [2, 6, 7]. They are routinely used in logic synthesis. Index generation functions often have effective decompositions.

An index generation function can be efficiently implemented by an LUT or an **IGU** (Index Generation Unit), which are programmable [9]. An IGU implementation of an index generation function dissipates much lower power and are less expensive than a CAM implementation [10]. Suppose that LSIs for IGUs with n inputs and weight k are available. For a function with larger k, we can partition the set of vectors into several groups, and implement each group by an independent LSI. The outputs of the LSIs can be combined by an OR gate to produce the final output [11]. This is a **parallel decomposition**. For a function with larger n, we can partition the set of input variables into two sets X_1 and X_2, and implement the function by a **serial decomposition** as shown in Fig. 1.1. The rest of the paper is organized as follows: Section II defines the basic

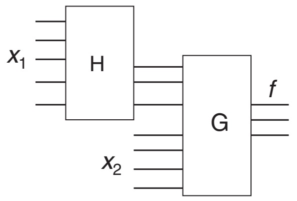

Fig. 1.1. Realization of a logic function by decomposition.

		0	0	1	1	x_1
		0	1	0	1	x_2
0	0	0	0	0	1	
0	1	1	1	0	0	
1	0	0	1	0	0	
1	1	0	0	0	0	
x_3	x_4					

Fig. 2.1. Decomposition chart of an logic function.

words. Section III introduces the properties of index generation functions. Section IV shows a heuristic method to find a good decomposition. Section V shows the experimental results. Section VI concludes the paper.

II. Definitions

In this part, we introduce basic concepts.

Definition 2.1 *[1] Let $f(X)$ be a function, and (X_1, X_2) be a partition of the input variables, where $X_1 = (x_1, x_2, \ldots, x_s)$ and $X_2 = (x_{s+1}, x_{s+2}, \ldots, x_n)$. The* **decomposition chart** *for f is a two-dimensional matrix with 2^s columns and 2^{n-s} rows, where each column and row is labeled by a unique binary assignment of values to the variables. Each assignment maps under f to $\{0, 1, \ldots, k\}$. The function represented by a column is a* **column function** *and is dependent on X_2. Variables in X_1 are* **bound variables***, while variables in X_2 are* **free variables***. In the decomposition chart, the* **column multiplicity***, denoted by $\mu(f : X_1)$, is the number of different column functions. The set of bound variables is the* **bound set***.*

Example 2.1 *Fig. 2.1 shows a decomposition chart of a 4-variable switching function. $X_1 = (x_1, x_2)$ denotes the bound variables, and $X_2 = (x_3, x_4)$ denotes the free variables. Since all the column patterns are different and there are four of them, the column multiplicity is $\mu(f : X_1) = 4$.* ■

Theorem 2.1 *[3] For a given function f, let X_1 be the bound variables, let X_2 be the free variables, and let $\mu(f : X_1)$ be the column multiplicity of the decomposition chart. Then, the*

TABLE 3.1
INDEX GENERATION FUNCTION

x_1	x_2	x_3	x_4	x_5	f
0	0	0	0	0	1
0	1	0	1	0	2
0	1	1	1	0	3
1	1	1	0	0	4
1	0	0	1	1	5
1	0	1	1	1	6
1	1	1	0	1	7

	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	x_1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	x_2
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	x_3
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	x_4
0	1	0	0	0	0	2	0	3	0	0	0	0	0	0	4	0	
1	0	0	0	0	0	0	0	0	0	5	0	6	0	0	7	0	
x_5																	

Fig. 3.1. Decomposition chart for f.

function f can be represented as $f(X_1, X_2) = g(h(X_1), X_2)$, and is realized with the network shown in Fig. 1.1. The number of signal lines connecting blocks H and G is $r = \lceil \log_2 \mu(f : X_1) \rceil$, where H and G realize h and g, respectively.

The logic functions for H and G can be realized by memories, and the complexities for G and H can be measured by the number of bits in the memories.

Definition 2.2 *The signal lines connecting H and G are called* **rails**. *When the number of rails r is smaller than the number of input variables in X_1, then function has a* **support-reducing decomposition** *[5].*

When the function has a support-reducing decomposition, the total amount of memory can often be reduced by realizing the logic in Fig. 1.1 [9].

III. Index Generation Functions and Their Properties

Definition 3.1 *Consider a set of k different binary 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, for each registered vector, its* **index**. *An* **index generation function** *f produces the corresponding index if the input matches a registered vector, and produces 0 otherwise. k is the* **weight** *of the index generation function. An index generation function represents a mapping: $f : B^n \rightarrow \{0, 1, 2, \ldots, k\}$, where $B = \{0, 1\}$.*

Typically, k is much smaller than 2^n, the total number of input combinations.

Example 3.1 *Consider the registered vector table shown in Table 3.1. It shows a 5-variable index generation function $f(X)$*

TABLE 3.2
TRUTH TABLE FOR h.

x_1	x_2	x_3	x_4	y_1	y_2	y_3
0	0	0	0	0	0	1
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	1	0	1
1	0	1	0	0	0	0
1	0	1	1	1	1	0
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	0

	0	0	0	0	1	1	1	1	y_1
	0	0	1	1	0	0	1	1	y_2
	0	1	0	1	0	1	0	1	y_3
0	0	1	2	3	4	0	0	0	
1	0	0	0	0	7	5	6	0	
x_5									

Fig. 3.2. Decomposition chart for g.

with weight 7. Consider the decomposition chart for f shown in Fig. 3.1. $X_1 = (x_1, x_2, x_3, x_4)$ denotes the bound variables, and $X_2 = (x_5)$ denotes the free variable. Note that the column multiplicity of this decomposition chart is 7. ■

Lemma 3.1 *Let $\mu(f : X_1)$ be the column multiplicity of a decomposition chart of an index generation function $f(x_1, X_2)$. Let k be the weight of f, and s be the number of variables in X_1. Then,*

$$\mu(f : X_1) \leq \min\{2^s, k + 1\}.$$

Lemma 3.2 *Let f be an index generation function with weight k. Then, there exists a functional decomposition $f(X_1, X_2) = g(h(X_1), X_2)$, where g and h are index generation functions, the weight of g is k, and the weight of h is at most k.*

Example 3.2 *Consider the decomposition chart in Fig. 3.1. Let the function $f(X)$ be decomposed as $f(X_1, X_2) = g(h(X_1), X_2)$, where $X_1 = (x_1, x_2, x_3, x_4)$, and $X_2 = (x_5)$. Table 3.2 shows the function h. It can be considered as a 4-variable index generation function with weight 6. The decomposition chart for the function g is shown in Fig. 3.2. As shown in this example, the functions obtained by decomposing the index generation function f are also index generation functions, and the weights of f and g are both 7.* ■

Property 3.1 *Assume that an index generation function f with weight k is implemented by the circuit in Fig. 1.1. When the number of the bound variables is $s = \left\lceil \frac{n+q}{2} \right\rceil$, the amount of memory is minimized, in many cases, where $q = \lceil \log_2(k+1) \rceil$.*

(Explanation) Consider the decomposition $f(X_1, X_2) = g(h(X_1), X_2)$. Assume that the column multiplicity μ satisfies the relation $2^{q-1} < \mu \leq 2^q$, then, the LUT for H requires $q \cdot 2^s$ bits. The LUT for G requires $q \cdot 2^{q+(n-s)}$ bits. Thus, the total size is $q \cdot (2^s + 2^{q+n-s})$. This value takes its minimum when $s = q + n - s$. From this, we have $s = \left\lceil \frac{n+q}{2} \right\rceil$. □

Theorem 3.1 *Consider an n variable index generation function f with weight k. When $k \leq 2^{n-4} - 1$, f has a support-reducing decomposition, and its amount of memory is a half of the single-LUT realization.*

In index generation functions, in most applications, the conditions of $k \leq 2^{n-4} - 4$ is satisfied. Thus, we can assume that index generation functions have support reducing decompositions. **This is a salient property that cannot be found in ordinary logic functions**.

IV. Heuristic Method to Select Bound Variables

In a functional decomposition, we can often select any s variables for the bound variables. In this case, the problem is to select a bound set having a small column multiplicity. There are $\binom{n}{s}$ ways to select bound sets. In many cases, to find an optimum bound set by the exhaustive method is infeasible. In this section, we show a heuristic method to find a good bound set using three different measures: column multiplicity, imbalance measure, and ambiguity measure.

Example 4.1 *Consider the index generation function shown in Table 4.1. Assume that $s = 3$, i.e., the number of bound variables is three. When (x_1, x_2, x_3) is the bound set, the column multiplicity is five. On the other hand, when (y_1, y_2, y_3) is the set, the column multiplicity is eight. Thus, (x_1, x_2, x_3) produce a smaller column multiplicity than (y_1, y_2, y_3).*

To reduce the column multiplicity, variables with less information [4] should be selected for bound variables. Fig. 4.1 shows the decision tree for the function when $(x_1, x_2, \ldots, x_7)$ is used for the bound set. On the other hand, Fig. 4.2 shows the decision tree for the same function when (y_1, y_2, y_3) is used for the bound set. To distinguish 7 vectors, the tree in Fig. 4.1 requires 6 variables, while the tree in Fig. 4.2 requires only three variables. In other words, a variable in $\{x_1, x_2, \ldots, x_7\}$ has less information than one in $\{y_1, y_2, y_3\}$. ■

To select a good bound set, a measure showing the degree of imbalance for the tree is introduced.

Definition 4.1 *In the registered vector table for a function f, 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 f with respect to the variable x_i is*

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

TABLE 4.1
Registered Vector Table

Vector										Index
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	0	1	0	2
0	0	1	0	0	0	0	1	1	0	3
0	0	0	1	0	0	0	0	0	1	4
0	0	0	0	1	0	0	1	0	1	5
0	0	0	0	0	1	0	0	1	1	6
0	0	0	0	0	0	1	1	1	1	7

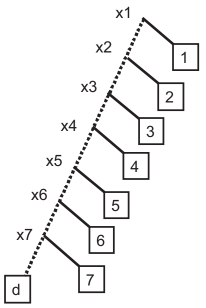

Fig. 4.1. Unbalanced Decision Tree.

In the variable x_i, when the numbers of occurrences of 0's and 1's are the same, $\omega(f : x_i)$ takes its minimum. The larger the difference of the occurrences of 0's and 1's, the larger the imbalance measure. Let k be the number of registered vectors. Then, $\nu(x_i, 0) + \nu(x_i, 1) = k$.

Example 4.2 *In Table 4.1, since $\nu(x_i, 0) = 6$ and $\nu(x_i, 1) = 1$ for $i = 1, 2, \ldots, 7$, we have*

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

However, since $\nu(y_j, 0) = 3$ and $\nu(y_j, 1) = 4$ for $j = 1, 2, 3$, we have

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

The imbalance measure for f with respect to x_i ,$(i = 1, 2, \ldots, 7)$ is larger than that for y_j, $(j = 1, 2, 3)$. Thus, x_i is more suitable for bound variables than y_j. ■

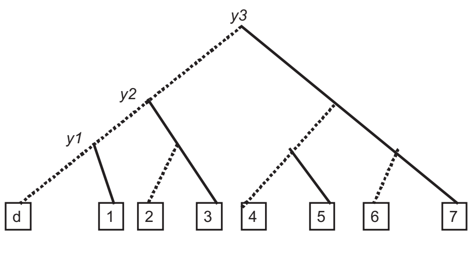

Fig. 4.2. Balanced Decision Tree.

The imbalance measure is useful for finding a single bound variable, but not so effective for a set of variables. Thus, we introduce another measure.

Definition 4.2 *Let $f(x_1, x_2, \ldots, x_n)$ be an index generation function with weight $|f|$. Let X_1 be a proper subset of $\{x_1, x_2, \ldots, x_n\}$. Assume that X_1 be an ordered set. Then, X_1 is a* **partial vector** *of $\{x_1, x_2, \ldots, x_n\}$. Suppose that the values of X_1 are fixed at $\vec{a}$. Let $N(f, X_1, \vec{a})$ be the number of registered vectors of f that take non-zero values, when the values of X_1 are set to $\vec{a} = (a_1, a_2, \ldots, a_s)$, $a_i \in \{0,1\}$. The* **ambiguity measure** *of f with respect to X_1 is*

$$AMB(f : X_1) = -|f| + \sum_{\vec{a} \in B^s} N(f, X_1, \vec{a})^2.$$

Example 4.3 *Consider the index generation function f in Table 3.1. Assume that the values of (x_1, x_2, x_3) are changed as $(0,0,0)$,$(0,0,1)$,$(0,1,0)$, $(0,1,1)$,$(1,0,0)$,$(1,0,1)$, $(1,1,0)$,$(1,1,1)$, in this order. Then, the values of f change as follows:*

$$[1], [0], [2], [3], [5], [6], [0], [4, 7].$$

In this case, the AMB measure with respect to (x_1, x_2, x_3) is

$$\begin{aligned} & AMB(f : x_1, x_2, x_3) \\ = \quad & -7 + (1^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 2^2) = 2. \end{aligned}$$

When $(x_1, x_2, x_3) = (1,1,1)$, the value of f is **ambiguous**, *since f can be either 4 or 7.*

Next, let the variable set be (x_1, x_2, x_4). Similarly, the values of f change as follows:

$$[1], [0], [0], [2, 3], [0], [5, 6], [4, 7], [0].$$

In this case, the AMB measure with respect to (x_1, x_2, x_4) is

$$\begin{aligned} & AMB(f : x_1, x_2, x_4) \\ = \quad & -7 + (1^2 + 0^2 + 0^2 + 2^2 + 0^2 + 2^2 + 2^2 + 0^2) = 6. \end{aligned}$$

When $(x_1, x_2, x_4) = (0,1,1)$, $(1,0,1)$ and $(1,1,0)$, the values of f are ambiguous.

Finally, let the variable set be (x_3, x_4, x_5). Similarly, the values of f change as follows:

$$[1], [0], [2], [5], [4], [7], [3], [6].$$

In this case, the AMB measure with respect to (x_3, x_4, x_5) is

$$\begin{aligned} & AMB(f : x_3, x_4, x_5) \\ = \quad & -7 + (1^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2) = 0. \end{aligned}$$

Note that f can be represented with only (x_3, x_4, x_5). ■

By using three measures, we have a heuristic algorithm to select a bound set that produces a small column multiplicity. The first bound variable is chosen to maximize the imbalance measure. Then, other bound variables are selected using the ambiguity measure $AMB(f : X_1)$ and the column multiplicity $\mu(f : X_1)$.

TABLE 4.2
Original Table.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Index
0	1	1	0	0	0	0	1	1
0	1	0	1	1	1	1	1	2
1	1	1	1	0	1	0	1	3
0	0	0	1	1	1	1	0	4
0	0	1	1	1	1	0	0	5
0	1	1	1	0	0	1	0	6
0	0	1	0	0	0	1	1	7
1	1	1	1	1	1	1	1	8
1	1	1	0	1	1	1	0	9
1	0	1	0	0	0	0	1	10
52	52	68	52	50	52	52	52	

Algorithm 4.1 *(Heuristic Method to Select a Bound Set)*

1. *Let $T = \{x_1, x_2, \ldots, x_n\}$ be the set of the input variables. Let $s \geq 2$ be the number of bound variables. Let k be the number of registered vectors. Let $s_{max} = \lceil \frac{n+q}{2} \rceil$ be the maximum number of bound variables, where $q = \log_2(k+1)$. Let Limit be the maximum column multiplicity to find. Usually, Limit is set to k.*

2. *For each variable x_i, count the number of 0's and 1's in the registered vector table, and compute the imbalance measure.*

3. *Let y_1 be the variable with the largest imbalance measure. Let $X_1 \leftarrow (y_1)$, $T \leftarrow T - y_1$. Let $s \leftarrow 1$.*

4. *While $\mu(f : X_1) <$ Limit and $s \leq s_{max}$, find the variable y_j in T that maximize the value of $AMB(f : X_1, y_j)$ and satisfy the condition $\mu(f : X_1, y_j) \leq$ Limit. Let $X_1 \leftarrow (X_1, y_j)$, $T \leftarrow T - y_j$. Let $s \leftarrow s + 1$*

5. *X_1 is the bound set. Stop.*

As will be shown in the experimental results, Algorithm 4.1 produces fairly good solutions in a short time.

Example 4.4 *Consider the index generation function shown in Table 4.2. The maximum number of the bound variables is*

$$\left\lceil \frac{n + \log_2(k+1)}{2} \right\rceil = \left\lceil \frac{8 + \log_2 11}{2} \right\rceil = 6.$$

Find a bound set having the column multiplicity at most Limit $= 8$. The last row of Table 4.2 shows the imbalance measure for each variable. Select a variable with the largest imbalance measure. In this case, we select x_3, since 68 is the maximum. Thus, $X_1 = (x_3)$ and $s = 1$. With this variable, the set of registered vectors are partitioned into two. Thus, the column multiplicity is $\mu(f : X_1) = 2$.

Next, select the second bound variable. Since $AMB(f : x_3, x_5) = 38$ gives the maximum value, x_5 is selected for the second bound variable. Thus, $X_1 = (x_3, x_5)$ and $s = 2$. In this case, the column multiplicity is $\mu(f : X_1) = 3 + 1 = 4$. Since $s \leq 6$ and $\mu \leq 8$, we try to add more bound variable.

TABLE 4.3
Distribution of Column Multiplicity.

μ	# of bound sets
8	2
9	9
10	27
11	18
Total	56

Then, we select the third bound variable. Since $AMB(f : x_3, x_5, x_6) = 30$ gives the maximum value, x_6 is selected as the third bound variable. Thus, $X_1 = (x_3, x_5, x_6)$ and $s = 3$. In this case, the column multiplicity is $\mu(f : X_1) = 4 + 1 = 5$. Since $s \leq 6$ and $\mu \leq 8$, we try to add more bound variable.

Then, we select the fourth bound variable. Since $AMB(f : x_3, x_5, x_6, x_1) = 20$ gives the maximum value, x_1 is selected as the fourth bound variable. Thus, $X_1 = (x_3, x_5, x_6, x_1)$ and $s = 4$. In this case, the column multiplicity is $\mu(f : X_1) = 6+1 = 7$. Since $s \leq 6$ and $\mu \leq 8$, we try to add more bound variable.

Then, we select the fifth bound variable. Since $AMB(f : x_3, x_5, x_6, x_1, x_7) = 16$ gives the maximum value, x_7 is selected as the fifth bound variable. Thus, $X_1 = (x_3, x_5, x_6, x_1, x_7)$ and $s = 5$. In this case, the column multiplicity is $\mu(f : X_1) = 7 + 1 = 8$. Since $s \leq 6$ and $\mu \leq 8$, we try to add more bound variable.

Then, we try to add the sixth bound variable. However, since no variable y in T satisfies the condition $\mu(f : X_1, y) \leq 8$, we terminate the procedure. The selected bound set is $X_1 = (x_3, x_5, x_6, x_1, x_7)$.

For this function, we can obtain exact solutions by an exhaustive search. Since the number of bound variables is five, there are $\binom{8}{5} = 56$ ways to select bound sets. Table 4.3 shows the results of the exhaustive search. The first column shows the multiplicity μ, while the second column shows the number of bound sets having μ. Only two bound sets produce the minimum column multiplicity $\mu = 8$. Algorithm 4.1 obtained a bound set having the minimum column multiplicity. ■

V. Experimental Results

We implemented Algorithm 4.1 for a Windows PC using 2.6 GHz Core i5 and 8 Giga byte of memory, on the Windows 7 (64-bit) operating system.

A. IP Address Tables

We collected distinct IP addresses of computers that accessed our web site over a period of one month. We considered four lists of different values of k. The original numbers of variables are 32 for all the functions. Table 5.1 shows the results. The first column shows the name of the function. The second column shows the number of the variables: n. The third column shows the number of registered vectors: k. The middle three columns shows the results obtained by Algorithm 4.1.

TABLE 5.1
Decomposition of IP Address Table.

			Alg 4.1			Monte		
Name	n	k	s	μ	r	*Min*	*Ave*	*Max*
*Ip*1670	32	1670	22	1311	11	1381	1571.8	1671
*Ip*3288	32	3288	22	2689	12	2799	3136.8	3289
*Ip*4591	32	4591	23	3901	12	4048	4444.8	4591
*Ip*7903	32	7903	23	5285	13	5523	6614.1	7902

TABLE 5.2
Decomposition of English Word Lists

			Alg 4.1			Monte		
Name	n	k	s	μ	r	*Min*	*Ave*	*Max*
*Dic*1730	40	1730	26	1058	11	1161	1562.3	1701
*Dic*3366	40	3366	26	1497	11	1778	2779.7	3271
*Dic*4705	40	4705	27	2947	12	4077	4152.5	4586

The fourth column shows the number of the bound variables: s. The fifth column shows the column multiplicity: μ. The sixth column shows the number of rails : r. The last three columns show the column multiplicities obtained by the Monte Carlo method [12] using 10000 random bound sets [1]. *Min*, *Ave*, and *Max* show the minimum, the average, and the maximum of the column multiplicities, respectively. As shown in Table 5.1, Algorithm 4.1 obtained better solutions than the Monte Carlo method. Note that for each function, the number of rails r is smaller than the number of bound variables s. Thus, the functions have efficient support-reducing decompositions.

B. Lists of English Words

To compress English text, we can use a list of frequently used words. We made three lists of English words: *Dic1730*, *Dic3366*, and *Dic4705*. 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 characters, we append blanks to make the length of words 8. We represent each alphabetic character by 5 bits. So, in the lists, all the words are represented by 40 bits. The numbers 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. Table 5.2 shows the results of decomposition. For *Dic1730* and *Dic3366*, $r < \lceil \log_2(k+1) \rceil$, i.e., the column multiplicities were greatly reduced.

Again, these functions have efficient support-reducing decompositions.

C. Lists of English Words Having the Same Word Length

Five additional lists of English words were derived in a similar way. However, in this case, each list consist of words with

[1]To the best of the authors knowledge, the serial decomposition of index generation functions was first considered in [12]. Thus, [12] only shows results for index generation functions.

TABLE 5.3
Decomposition of English Word Lists

			Alg 4.1			Monte		
Name	n	k	s	μ	r	Min	Ave	Max
$Char4$	20	768	15	378	9	450	635.1	729
$Char5$	25	820	18	644	10	625	760.4	808
$Char6$	30	809	20	739	10	677	782.3	805
$Char7$	35	701	23	619	10	644	691.7	703
$Char8$	40	548	25	496	9	517	543.5	549

the same length. Thus, the different list have different number of characters (variables). Also, no blank character(s) are contained in the lists.

Table 5.3 shows the results. For example, *Char4* shows the list of English words consisting of exactly four characters. Thus, it has $n = 5 \times 4 = 20$ input variables. When $s = 16$, the number of ways to select bound variables is $\binom{20}{15} = 15504$, which is small enough to do an exhaustive search to find the optimum decomposition. The exact minimum is $\mu = 378$, and Algorithm 4.1 obtained this solution. For $Char5$ and $Char6$, the Monte Carlo method obtained better solutions than Algorithm 4.1. However, the solutions obtained by Algorithm 4.1 still produce circuits with the minimum rails. Again, these functions have efficient support-reducing decompositions.

D. Random Functions

We generated nine random index generation functions that have the same value of (n, k), as the previous experiments. Table 5.4 shows the results.

Unlike non-random functions, column multiplicities of random functions were hard to reduce, and $\mu \simeq k + 1$. In fact, for all the random functions, $r = \lceil \log_2(k + 1) \rceil$. Thus, Property 3.1 holds for random functions. The differences of Min and Max values are smaller than the corresponding non-random functions having the same values of (n, k). Again, in the case of random functions, we obtained support-reducing decompositions.

TABLE 5.4
Decomposition of Random Index Generation Functions

			Alg 4.1			Monte		
Name	n	k	s	μ	r	Min	Ave	Max
$32i1670k$	32	1670	22	1669	11	1667	1670.7	1670
$32i3288k$	32	3288	22	3284	12	3283	3287.8	3289
$32i4591k$	32	4591	23	4587	13	4584	4590.8	4592
$32i7903k$	32	7903	23	7889	13	7890	7899.9	7904
$20i0768k$	20	768	15	752	10	745	759.3	768
$25i0820k$	25	820	18	817	10	813	819.6	821
$30i0809k$	30	809	20	808	10	806	809.7	810
$35i0701k$	35	701	23	699	10	700	702.0	742
$40i0548k$	40	548	25	548	10	548	549.0	549

E. Computation Time

The most time-consuming function was $IP7903$, which took 283 ms and 28.0 second by Algorithm 4.1 and the Monte Carlo method, respectively.

VI. Conclusion and Comments

In this paper, we present a heuristic method to find a decomposition of index generation functions. Three different measures are introduced to find a set of bound variables. Comparison with the Monte Carlo method shows that our heuristic produces good solutions in a short time. Also, we showed that index generation functions have support-reducing decompositions.

This algorithm is also useful for multiple-output n-variable logic functions where k, the number of input combinations that produce non-zero outputs, satisfies the condition $k \ll 2^n$.

Acknowledgments

This research is partly supported by the Japan Society for the Promotion of Science (JSPS) Grant in Aid for Scientific Research.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *Inter. Symp. on the Theory of Switching*, pp. 74-116, April 1957.

[2] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions,"*ICCAD-97*, pp. 78-82, Nov. 1997.

[3] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[4] S. Ganapathy and V. Rajaraman, "Information theory applied to the conversion of decision table to computer programs," *Communications of the ACM*,Vol. 16 Issue 9, Sept. 1973, pp. 532-539.

[5] V. Kravets and K. Sakallah, "Constructive library-aware synthesis using symmetries,"*Proc. DATE-2000*, pp. 208-213.

[6] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition,"*SASIMI'98*, pp. 44-50, Oct. 1998.

[7] T. Sasao, "FPGA design by generalized functional decomposition,"In *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publisher, pp. 233-258, 1993.

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

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

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

[11] T. Sasao, "A realization of index generation functions using multiple IGUs," *Inter. Symp. on Multiple-Valued Logic*, (ISMVL-2016), Sapporo, Japan, May 17-19, 2016. pp. 113-116.

[12] T. Sasao and J. T. Butler, "Decomposition of index generation functions using a Monte Carlo method," *Inter. Symp. on Logic and Synthesis*, (IWLS-2016), Austin, TX, U.S.A, June 10-11, 2016.