Improvement in the Quality of Solutions of a Heuristic Linear Decomposer for Index Generation Functions

Shinobu Nagayama* Tsutomu Sasao† Jon T. Butler‡

*Dept. of Computer and Network Eng., Hiroshima City University, Hiroshima, JAPAN
†Dept. of Computer Science, Meiji University, Kawasaki, JAPAN
‡Dept. of Electr. and Comp. Eng., Naval Postgraduate School, Monterey, CA USA

***Abstract*—This paper presents a method to improve a heuristic linear decomposer for index generation functions. Besides being fast and scalable, it also improves the solutions. This is done through the use of an efficient evaluation method that can find better solutions and can predict the quality of the solution. Experimental results show that the improved heuristic quickly finds exact optimum solutions that are not found by existing heuristics.**

***Keywords*-Linear decomposition; index generation functions; functional decomposition; balanced decision tree.**

I. Introduction

Functional decomposition [1], [4] is a way to compactly realize discrete functions with smaller sub-functions. Various methods for functional decomposition and their optimization algorithms [3], [5]–[7], [9], [17], [23] have been proposed to minimize circuit size. Among them, we focus on *linear decomposition* [6], [17] in this paper.

Linear decomposition realizes a function f with two parts: L and G, as shown in Fig. 1, where L realizes linear functions y_i $(i = 1, 2, \ldots, p)$, and G stores function values of f. The first part L produces y_i from inputs $x_1, x_2, \ldots, x_n$ of f, and the second part G outputs a function value of f from y_i. By using linear decomposition, *index generation functions* [19], [20] can be realized efficiently with a memory-based architecture [19].

In the memory-based architecture, L is implemented by EXOR gates, registers, and multiplexers, and G is implemented by a $(2^p \times q)$-bit memory. Since memory size of G strongly depends on the number of linear functions p in a linear decomposition, minimization of p has been required in a wide range of applications of index generation functions,

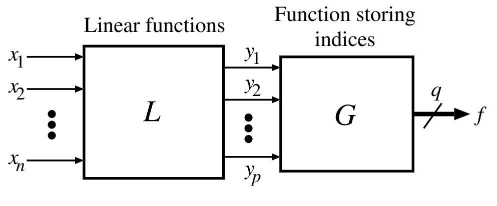


Figure 1. Linear decomposition of index generation functions [22].

such as IP address lookup tables, terminal access controllers, URL whitelists, computer virus scanning circuits, memory patch circuits, and code converters [29]. Thus, many minimization methods [2], [10]–[16], [21], [22], [25]–[28], [30]–[32] have been proposed.

We have also proposed a heuristic method [10]. Although the heuristic is fast and scalable for large index generation functions, it still has room for improvement in the quality of solutions. Since the heuristic greedily searches for a solution, it produces a local optimum solution. To improve the quality of solutions, predicting the quality of solutions and choosing a better candidate during the search are important. Thus, we propose such an evaluation method for candidates of solutions. The evaluation method is inspired by our previous study [16] on an analytic approach to exact optimization. Since the computational complexity of the proposed evaluation method is low, the new heuristic with the proposed evaluation method can find a better solution with the same time complexity as the previous heuristic.

The rest of this paper is organized as follows: Section II defines index generation functions and linear decomposition. Section III formulates the minimization problem of the number of linear functions, and outlines the previous heuristic to solve it. Section IV presents a three-step evaluation method to improve the quality of solutions of the heuristic. Section V shows experimental results from practical examples, and Section VI concludes the paper.

II. Preliminaries

This section shows brief definitions of index generation functions [19], [20] and linear decomposition [6], [17], [22].

A. Index Generation Functions

Definition 1: An **incompletely specified index generation function**, or simply **index generation function**, $f(X)$ is a multiple-valued function, where X is a tuple of n binary variables $(x_1, x_2, \ldots, x_n)$, and k assignments of values to binary variables $x_1, x_2, \ldots, x_n$ map to a set of **indices** $K = \{1, 2, \ldots, k\}$. That is, the variables of f are binary-valued, while f is k-valued. Further, there is a one-to-one relationship between the k assignments of values to

Table I
EXAMPLE OF INDEX GENERATION FUNCTION [11].

Registered vectors				indices
x_1	x_2	x_3	x_4	f
0	0	0	1	1
0	0	1	0	2
0	1	0	0	3
1	1	0	1	4

$x_1, x_2, \ldots, x_n$ and k indices. The k assignments of values to $x_1, x_2, \ldots, x_n$ are called the **registered vectors**. Other assignments are left unspecified. $k = |K|$ is called the **weight** of the index generation function f.

Example 1: Table I shows an example of a 4-variable index generation function with weight four. Note that, in this function, input values other than 0001, 0010, 0100, and 1101 are NOT assigned to any function values. $\square$

Definition 2: Let $K = \{1, 2, \ldots, k\}$ be a set of indices of an index generation function. If $K = I_1 \cup I_2 \cup \ldots \cup I_u$, each $I_i \neq \emptyset$, and $I_i \cap I_j = \emptyset$ $(i \neq j)$, then $\mathcal{P} = \{I_1, I_2, \ldots, I_u\}$ is a **partition** of the set of indices K.

B. Linear Decomposition

Definition 3: A **linear decomposition** of an index generation function $f(x_1, x_2, \ldots, x_n)$ realizes f using a function $g(y_1, y_2, \ldots, y_p)$ where y_i is a linear combination of $\{x_1, x_2, \ldots, x_n\}$:

$$y_i(x_1, x_2, \ldots, x_n) = a_{i1}x_1 \oplus a_{i2}x_2 \oplus \ldots \oplus a_{in}x_n,$$

$i \in \{1, 2, \ldots, p\}$, $a_{ij} \in \{0, 1\}$ $(j \in \{1, 2, \ldots, n\})$, and, for all registered vectors of the index generation function, the following holds:

$$f(x_1, x_2, \ldots, x_n) = g(y_1, y_2, \ldots, y_p).$$

Each y_i is called a **compound variable**. For each y_i, $\sum_{j=1}^{n} a_{ij}$ is called a **compound degree** of y_i, denoted by $deg(y_i)$, where a_{ij} is viewed as an integer, and $\sum$ as an integer sum.

Definition 4: An inverse function of an index storing function $z = g(y_1, y_2, \ldots, y_p)$ in a linear decomposition is a mapping from $K = \{1, 2, \ldots, k\}$ to a set of p-bit vectors $\{0, 1\}^p$, denoted by $g^{-1}(z)$. In this inverse function $g^{-1}(z)$, a mapping obtained by focusing only on the i-th bit of the p-bit vectors: $K \to \{0, 1\}$ is called an **inverse function to a compound variable** y_i, denoted by $(g^{-1})_i(z)$.

Definition 5: Let $ON(y_i) = \{z \mid z \in K, (g^{-1})_i(z) = 1\}$, where $K = \{1, 2, \ldots, k\}$ and $(g^{-1})_i(z)$ is an inverse function of $g(y_1, y_2, \ldots, y_n)$ to y_i. $|ON(y_i)|$ is called the **cardinality of** y_i or informally the **number of 1's included in** y_i.

Example 2: The index generation function f in Example 1 can be decomposed into two linear functions: $y_1 = x_2$ and $y_2 = x_1 \oplus x_3$, and a function $g(y_1, y_2)$ shown in Table II. All four function values of f are distinguished by just y_1 and y_2. In this case, $deg(y_1) = 1$ and $deg(y_2) = 2$, respectively. The original index generation function f can be realized by the architecture in Fig. 1 with a $(2^2 \times 3 = 12)$-bit memory for G, while a $(2^4 \times 3 = 48)$-bit memory is needed to directly realize f without linear decomposition.

Table II
FUNCTION g STORING INDICES IN LINEAR DECOMPOSITION OF f.

y_1	y_2	g
0	0	1
0	1	2
1	0	3
1	1	4

For $g(y_1, y_2)$ in Table II, its inverse functions to y_1 and y_2 are $(g^{-1})_1(z)$ and $(g^{-1})_2(z)$, respectively. We have $(g^{-1})_1(1) = 0$, $(g^{-1})_1(2) = 0$, $(g^{-1})_1(3) = 1$, and $(g^{-1})_1(4) = 1$. Similarly, $(g^{-1})_2(1) = 0$, $(g^{-1})_2(2) = 1$, $(g^{-1})_2(3) = 0$, and $(g^{-1})_2(4) = 1$. The cardinalities of both y_1 and y_2 are 2. $\square$

In this way, linear decomposition can significantly reduce memory size needed to realize an index generation function.

III. OPTIMIZATION OF LINEAR DECOMPOSITION

This section formulates the optimization problem for linear decomposition of index generation functions, and outlines an existing heuristic method based on balanced decision trees [10].

A. Formulation of Optimization Problem

We formulate the optimization problem for linear decomposition of index generation functions as follows:

Problem 1: Given an index generation function f and the maximum compound degree t, find a linear decomposition of f such that the number of linear functions p is minimum, and all compound degrees are at most t.

As shown in the previous sections, the memory size of G using linear decomposition exponentially increases with the number of linear functions p. Thus, minimization of p is important. In general, we can reduce the number of linear functions by using large compound degree [10], [22] However, large compound degree makes the circuit size of L large (L is implemented with EXOR gates, registers, and multiplexers). To obtain an optimum design considering balance of both sizes of L and G, a careful choice of t is important as well. In this paper, however, we focus on the minimization of p under a given t.

Example 3: The linear decomposition of f shown in Example 2 is optimum when $t = 2$. This is because at least 2 variables are needed to distinguish 4 indices, and the compound degrees of y_1 and y_2 are at most 2. $\square$

B. Outline of Balanced Tree Based Heuristic Method

Problem 1 can be reduced to the minimization of tree height of an ordered binary decision tree that recursively divides sets of indices by compound variables until only singletons remain [10].

Example 4: Fig. 2 shows an ordered binary decision tree representing g in Table II with the smallest height. The tree

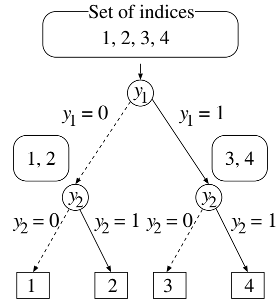


Figure 2. Binary decision tree for g of Table II [16].

Heuristic 1: Overview of heuristic for a compound variable

```
Input: a partition of indices $\mathcal{P}$, an index generation
function, and a compound degree $t$
Output: a compound variable $y_{opt}$
1. Let $y$ be 0 (the constant zero function).
2. Evaluate $x_1, x_2, \ldots,$ and $x_n$ using (1) and (2).
3. Choose the best $x_i$ among them.
4. Replace $y$ with $y \oplus x_i$ and $deg(y)$ with $deg(y)+1$.
5. If the candidate $y$ is better than the best one so
   far, then $y_{opt} = y$.
6. Iterate Steps 2 to 5 until $deg(y) = t$.
```

divides the set of 4 indices into singletons by compound variables y_1 and y_2. The tree height corresponds to the number of compound variables. □

Since an ordered binary decision tree with the smallest height is a *balanced decision tree*, a heuristic to construct a balanced decision tree using compound variables has been proposed [10]. The heuristic recursively divides a set of indices into two subsets by a compound variable so that the divided subsets have balanced sizes. In the heuristic, such a compound variable is heuristically chosen by using the following cost function: [10]

$$\mathrm{cost}_1(\mathcal{P}, y_i) = \sqrt{\sum_{I \in \mathcal{P}} \left(\frac{|I|}{2} - |I \cap ON(y_i)| \right)^2}, \qquad (1)$$

where $\mathcal{P}$ is a partition of a set of indices with already chosen compound variables. A smaller value of the cost function (1) means that a partition obtained by y_i is closer to the ideal one, where all subsets in $\mathcal{P}$ are divided into halves. When values of the cost function are equal, the heuristic chooses a compound variable using the following as the second cost function for breaking a tie: [10]

$$\mathrm{cost}_2(\mathcal{P}, y_i) = \max_{I \in \mathcal{P}}(\max(|I \cap ON(y_i)|, |I \setminus ON(y_i)|)). \qquad (2)$$

By choosing a compound variable y_i with a smaller value of this cost function, the size of the largest subset among divided subsets becomes smaller.

Heuristic 1 is a heuristic to find a compound variable using the cost functions. Since it chooses promising original variables x_i using the cost functions, and compounds only

Heuristic 2: Overview of heuristic for linear decomposition

```
Input: an index generation function and a compound
degree $t$
Output: a set of compound variables
1. Let $\mathcal{P} = \{K\}$ and $i = 1$.
2. Find a compound variable $y_i$ by Heuristic 1.
3. Divide each $I \in \mathcal{P}$ with $y_i$.
4. Update $\mathcal{P}$ with the divided subsets.
5. $i = i + 1$.
6. Iterate Steps 2 to 5 until $|\mathcal{P}| = k$.
```

those variables, a good compound variable can be found with small time and space complexities. In Steps 2 and 5, the cost functions are used to evaluate candidates of solutions.

Heuristic 2 is a heuristic to find a good linear decomposition using Heuristic 1 iteratively. Heuristic 2 divides a set of indices recursively using compound variables selected by Heuristic 1, and it terminates when a set of indices is divided into singletons.

IV. Evaluation Method of Solutions

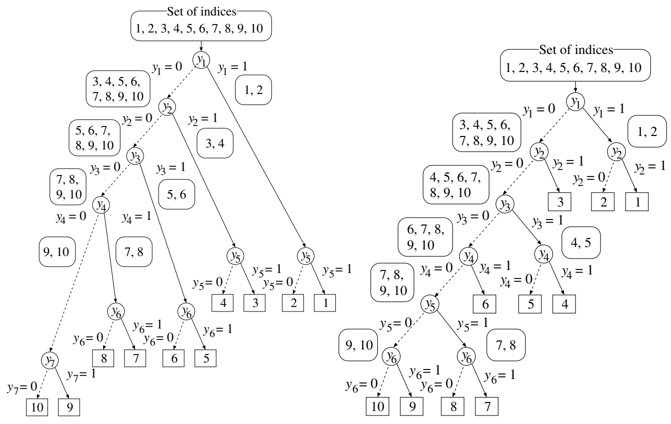
As shown in the previous section, the existing heuristic [10] evaluates the quality of solutions by only the cost functions. Although in the best-case scenario, the heuristic can find a compound variable producing a partition close to the ideal one by using the cost functions, a given index generation function does not always have such a compound variable. This is because the number of 1's included in y_i is not always balanced (i.e., $k/2$). Specifically, when the number of 1's in y_i is small, an *unbalanced tree* is necessarily constructed, regardless of which compound variable is chosen. In that case, another evaluation criterion should be used, instead of the criteria with the goal of constructing a balanced tree.

The cost functions (1) and (2) are criteria for mainly evaluating whether a *current* partition of indices is closer to the ideal one than others, or not. They *do not look ahead* to a whole solution (i.e., total tree height). In unbalanced trees, however, considering reduction of tree height directly is more important than considering balance of the partition on an ad hoc basis. This is because it is hard to keep balance in a partition in unbalanced trees. Unpromising partitions can be chosen by the cost functions (1) and (2). In addition, when tree height is not reduced even if the largest subset in a partition is divided, dividing other subsets as well as the largest subset is more desirable than making the largest subset smaller according to the cost function (2). That is, in that case, a compound variable dividing more subsets is more promising than a compound variable making the largest subset smaller.

Example 5: As an example where unbalanced trees are necessarily constructed, let us consider an index generation function defined by Table III and its linear decomposition with a compound degree $t = 2$. Since the number of 1's in

Table III
AN INDEX GENERATION FUNCTION (1-OUT-OF-10 CODE).

Registered vectors										indices
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	f
1	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	2
0	0	1	0	0	0	0	0	0	0	3
0	0	0	1	0	0	0	0	0	0	4
0	0	0	0	1	0	0	0	0	0	5
0	0	0	0	0	1	0	0	0	0	6
0	0	0	0	0	0	1	0	0	0	7
0	0	0	0	0	0	0	1	0	0	8
0	0	0	0	0	0	0	0	1	0	9
0	0	0	0	0	0	0	0	0	1	10



(a) Non-optimal tree. (b) Optimum tree.

Figure 3. Binary decision trees for 1-out-of-10 code with $t = 2$ [12].

a compound variable is at most two, up to two indices can be separated from subsets of indices by using a compound variable. The heuristic using the cost functions (1) and (2) preferentially divides the largest subset without dividing smaller subsets, and thus, it produces a binary decision tree of the type shown in Fig. 3(a). In this case, 7 compound variables are needed to isolate each index, resulting in a non-optimal solution. This is because unpromising partitions are chosen by the cost functions (1) and (2).

On the other hand, by dividing more subsets at each level of a binary decision tree as shown in Fig. 3(b), we have the optimum solution. It uses 6 compound variables. This is because in this example, tree height is not reduced even if the largest subset is preferentially divided. □

To avoid choosing such unpromising partitions, we propose an evaluation method of sub-solutions consisting of the following three criteria:

1) estimate of tree height,
2) the number of divided subsets, and
3) balance of partition (i.e., the cost function (1)).

We begin with estimating tree height to evaluate a sub-solution. Since a subset of indices I is recursively divided into two, a lower bound on tree height of I is

$$\lceil \log_2(|I|) \rceil .$$

Using this lower bound, we estimate tree height after subsets in a partition $\mathcal{P}$ are divided by a compound variable y_i as follows:

$$\text{tree\_height}(\mathcal{P}, y_i) = \max_{I \in \mathcal{P}}(\max(\lceil \log_2(|I \cap ON(y_i)|) \rceil, \lceil \log_2(|I \setminus ON(y_i)|) \rceil)).$$

We choose a compound variable y_i producing smaller tree height by using this estimate. When estimates of tree height are equal, the number of divided subsets by y_i is evaluated at the second step. At the second step, we choose y_i dividing more subsets. When the numbers of divided subsets are also equal, we use the cost function (1) at the third evaluation.

This improvement just changes the cost functions in Heuristics 1 and 2 into the above three-step evaluation. Since the computational complexity of the three-step evaluation is the same as the computational complexity of the cost functions, the improved heuristic also has the same complexity as Heuristic 2 that is [10]

$$O(nk \log(k)),$$

where n is the number of original variables, and k is the number of indices for an index generation function. Thus, the proposed evaluation method can improve the quality of solutions of the heuristic without sacrificing scalability.

V. EXPERIMENTAL RESULTS

The proposed three-step evaluation method is incorporated into the heuristic [10] implemented in the C language, and run on the following computer environment: CPU: Intel Core2 Quad Q6600 2.4GHz, memory: 4GB, OS: CentOS 5.7 Linux, and C-compiler: gcc -O2 (version 4.1.2).

A. *For Symmetric Index Generation Functions*

To evaluate the effectiveness of the proposed method, we begin with computer experiments for *symmetric* index generation functions [14], [16], [32]. This is because in symmetric index generation functions, all compound variables have the same number of 1's [16], and thus, it is easier to see the effectiveness of the proposed method, excluding other influences. In addition, the exact smallest numbers of compound variables have been obtained even for large symmetric index generation functions [14], [16], [32].

We compare the proposed method with existing heuristic methods [10], [12], in terms of the number of compound variables. Table IV shows the number of compound variables obtained by each method for symmetric index generation functions. The column labeled "opt" in Table IV shows the smallest number of compound variables found by the exact optimization method [16]. Boldfaced numbers denote the exact optimum solutions found by heuristic methods. Computation time is not shown in Table IV, since the heuristic methods are fast, and their computation time for any of the functions is shorter than 0.01 msec.

Table IV
NUMBER OF COMPOUND VARIABLES FOR SYMMETRIC FUNCTIONS.

Symmetric functions	t	opt	Existing [10]	Existing [12]	Proposed
1-out-of-10 code ($n = k = 10$)	1	9	**9**	**9**	**9**
	2	6	7	**6**	**6**
	3	5	**5**	**5**	**5**
	4	4	**4**	**4**	**4**
	5	4	**4**	**4**	**4**
1-out-of-20 code ($n = k = 20$)	1	19	**19**	**19**	**19**
	2	13	14	**13**	**13**
	3	10	**10**	**10**	**10**
	4	8	**8**	**8**	**8**
	5	7	**7**	**7**	**7**
1-out-of-30 code ($n = k = 30$)	1	29	**29**	**29**	**29**
	2	20	22	**20**	**20**
	3	15	16	**15**	**15**
	4	12	**12**	**12**	**12**
	5	10	11	11	**10**
1-out-of-40 code ($n = k = 40$)	1	39	**39**	**39**	**39**
	2	26	29	**26**	**26**
	3	20	22	**20**	**20**
	4	16	17	17	**16**
	5	13	14	14	14
1-out-of-50 code ($n = k = 50$)	1	49	**49**	**49**	**49**
	2	33	37	**33**	**33**
	3	25	27	**25**	**25**
	4	20	21	21	**20**
	5	17	**17**	18	**17**
1-out-of-60 code ($n = k = 60$)	1	59	**59**	**59**	**59**
	2	40	44	**40**	**40**
	3	30	33	**30**	**30**
	4	24	26	25	**24**
	5	20	21	21	**20**
1-out-of-70 code ($n = k = 70$)	1	69	**69**	**69**	**69**
	2	46	52	**46**	**46**
	3	35	38	**35**	**35**
	4	28	30	29	**28**
	5	23	25	25	24
1-out-of-80 code ($n = k = 80$)	1	79	**79**	**79**	**79**
	2	53	59	**53**	**53**
	3	40	44	**40**	**40**
	4	32	34	33	**32**
	5	27	29	28	**27**

Boldface numbers are optimum.

From Table IV, we can see that the heuristic [10] tends to produce worse solutions as the number of indices k increases. This is because as mentioned in Section IV, partitions of indices are necessarily unbalanced. The degree of unbalance becomes larger as k increases when the compound degrees t is relatively small. Thus, unpromising partitions are chosen by the cost functions (1) and (2), resulting in non-optimal solutions. Although in [12], the heuristic method is modified by introducing a constraint to improve the quality of solutions, it still tends to choose unpromising partitions when k is large since the same cost functions are used. On the other hand, the proposed method produces the exact optimum solutions for almost all cases. Indeed, when the proposed method does not yield the optimum results, it is only off by 1.

B. *For Large General Index Generation Functions*

Table V shows the number of compound variables and computation time of the existing method [12] and the proposed method for larger benchmark index generation functions presented in [10]: random *social security and tax numbers* (SST numbers) in Japan, the *bible*, and the *US constitution* including amendments.

In these benchmark functions, the number of 1's included in an original variable x_i is relatively large [10], [12]. In that case, balanced decision trees tend to be constructed, and thus, the existing heuristic, with the goal of keeping balance in partitions, works well. Specifically, for "Bible" with $t = 1$, it works well. Since the proposed method gives preference to "the number of divided subsets" over "balance of partition" and it is a greedy search without backtracking, a worse local optimum solution is produced. However, the proposed method improves the quality of solutions for some cases. We need further analysis to know which "the number of divided subsets" or "balance of partition" should be prioritized for each function. That is one of our future works.

The computation time of the proposed method is comparable to the computation time of the existing one. From this result, we can see that the computational overhead of the three-step evaluation is small. Therefore, the proposed method can improve the quality of solutions without sacrificing scalability.

VI. CONCLUSION AND COMMENTS

This paper proposes a three-step evaluation method to improve the quality of solutions of a heuristic linear decomposer for index generation functions. Since the proposed evaluation method chooses a candidate while predicting the quality of the solution, the heuristic incorporating the proposed three-step evaluation method can find a better solution than the previous heuristics. Experimental results show that the proposed method can improve the quality of solutions with the same computational overhead.

Since the proposed evaluation method is inspired by the study on analytic approach to exact optimization for symmetric index generation functions, it works well especially for symmetric index generation functions. On the other hand, for functions such that balanced decision trees tend to be constructed, it still has room for improvement. This would be our future work.

ACKNOWLEDGMENTS

This research is partly supported by the JSPS KAKENHI Grant (C), No.19K11881, 2020. The reviewers' comments were helpful in improving the paper.

REFERENCES

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

[2] J. Astola, P. Astola, R. Stankovic, and I. Tabus, "An algebraic approach to reducing the number of variables of incompletely defined discrete functions," *46th International Symposium on Multiple-Valued Logic*, pp. 107–112, May 2016.

[3] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," *International Conference on Computer Aided Design (ICCAD)*, pp. 78–82, 1997.

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

Table V
NUMBER OF COMPOUND VARIABLES AND COMPUTATION TIME FOR LARGE INDEX GENERATION FUNCTIONS.

Number of compound variables		Compound degree t									
Benchmarks	Methods	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
SST numbers	[12]	42	37	36	35	35	35	35	35	35	35
$(n=48, k=1,000,000)$	Proposed	**41**	**36**	36	35	35	35	35	35	35	35
Bible	[12]	**44**	31	28	27	**25**	25	25	**24**	24	24
$(n=560, k=20,827)$	Proposed	47	31	28	27	26	25	25	25	24	24
US constitution	[12]	**15**	12	11	11	10	10	10	10	11	11
$(n=1,512, k=253)$	Proposed	16	12	11	**10**	10	10	10	10	**10**	**10**

Computation time (sec.)		$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
SST numbers	[12]	81.31	167.19	252.92	334.85	416.14	499.50	572.41	648.32	720.80	790.85
	Proposed	87.55	176.30	264.63	349.18	439.85	521.89	602.10	682.86	758.79	831.27
Bible	[12]	3.96	6.72	9.55	12.42	15.11	17.93	20.73	23.44	26.25	29.01
	Proposed	5.31	8.42	12.05	15.34	18.67	21.92	25.41	28.83	32.23	35.37
US constitution	[12]	0.06	0.08	0.11	0.14	0.15	0.18	0.20	0.22	0.25	0.28
	Proposed	0.07	0.10	0.14	0.16	0.19	0.22	0.26	0.28	0.31	0.33

Boldfaced numbers denote better solutions.

[5] C. Files, R. Drechsler, and M. A. Perkowski, "Functional decomposition of MVL functions using multi-valued decision diagrams," *27th International Symposium on Multiple-Valued Logic*, pp. 27–32, 1997.

[6] R. J. Lechner, "Harmonic analysis of switching functions," in A. Mukhopadhyay (ed.), *Recent Developments in Switching Theory*, Academic Press, New York, Chapter V, pp. 121–228, 1971.

[7] T. Mazurkiewicz, "Non-disjoint functional decomposition of index generation functions," *50th International Symposium on Multiple-Valued Logic*, pp. 137–142, Nov. 2020.

[8] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," *30th Design Automation Conference*, pp. 272–277, 1993.

[9] A. Mishchenko, B. Steinbach, and M. A. Perkowski, "An algorithm for bi-decomposition of logic functions," *38th Design Automation Conference*, pp. 103–108, 2001.

[10] S. Nagayama, T. Sasao, and J. T. Butler, "An efficient heuristic for linear decomposition of index generation functions," *46th International Symposium on Multiple-Valued Logic*, pp. 96–101, May 2016.

[11] S. Nagayama, T. Sasao, and J. T. Butler, "An exact optimization algorithm for linear decomposition of index generation functions," *47th International Symposium on Multiple-Valued Logic*, pp. 161–166, May 2017.

[12] S. Nagayama, T. Sasao, and J. T. Butler, "A balanced decision tree based heuristic for linear decomposition of index generation functions," *IEICE Transactions on Information and Systems*, Vol. E100-D, No. 8, pp. 1583–1591, Aug. 2017.

[13] S. Nagayama, T. Sasao, and J. T. Butler, "An exact optimization method using ZDDs for linear decomposition of index generation functions," *48th International Symposium on Multiple-Valued Logic*, pp. 144–149, May 2018.

[14] S. Nagayama, T. Sasao, and J. T. Butler, "An exact optimization method using ZDDs for linear decomposition of symmetric index generation functions," *International Federation of Computational Logic Journal of Logic and Their Applications*, Vol. 5, No. 9, pp. 1849–1866, Dec. 2018.

[15] S. Nagayama, T. Sasao, and J. T. Butler, "A dynamic programming based method for optimum linear decomposition of index generation functions," *49th International Symposium on Multiple-Valued Logic*, pp. 144–149, May 2019.

[16] S. Nagayama, T. Sasao, and J. T. Butler, "On optimum linear decomposition of symmetric index generation functions," *50th International Symposium on Multiple-Valued Logic*, pp. 130–136, Nov. 2020.

[17] 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 (in Russian).

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

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

[20] T. Sasao, "Index generation functions: recent developments (invited paper)," *41st International Symposium on Multiple-Valued Logic*, pp. 1–9, May 2011.

[21] T. Sasao, "Linear transformations for variable reduction," *Reed-Muller Workshop 2011*, May 2011.

[22] T. Sasao, "Linear decomposition of index generation functions," *17th Asia and South Pacific Design Automation Conference*, pp. 781–788, Jan. 2012.

[23] T. Sasao, "Row-shift decompositions for index generation functions," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1585–1590, 2012.

[24] T. Sasao, Y. Urano, and Y. Iguchi, "A lower bound on the number of variables to represent incompletely specified index generation functions," *44th International Symposium on Multiple-Valued Logic*, pp. 7–12, May 2014.

[25] T. Sasao, Y. Urano, and Y. Iguchi, "A method to find linear decompositions for incompletely specified index generation functions using difference matrix," *IEICE Transactions on Fundamentals*, Vol. E97-A, No. 12, pp. 2427–2433, Dec. 2014.

[26] 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*, pp. 164–169, May 2015.

[27] 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*, pp. 161–167, June 2015.

[28] T. Sasao, I. Fumishi, and Y. Iguchi, "On an exact minimization of variables for incompletely specified index generation functions using SAT," *Note on Multiple-Valued Logic in Japan*, Vol. 38, No. 3, pp. 1–8, Sept. 2015 (in Japanese).

[29] T. Sasao, *Index Generation Functions*, Morgan & Claypool Publishers, 2017, Chapt. 2.

[30] 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*, pp. 812–817, March 2017.

[31] D. A. Simovici, M. Zimand, and D. Pletea, "Several remarks on index generation functions," *42nd International Symposium on Multiple-Valued Logic*, pp. 179–184, May 2012.

[32] B. Steinbach and C. Postoff, "Fast optimal synthesis of symmetric index generation functions," *International Workshop on Boolean Problems*, paper 1, pp. 1–16, Sept. 2020.