An Efficient Heuristic for Linear Decomposition of 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 proposes a heuristic for linear decomposition of index generation functions using a balanced decision tree. The proposed heuristic finds a good linear decomposition of an index generation function by recursively dividing a set of its function values into two balanced subsets. Since the proposed heuristic is fast and requires a small amount of memory, it is applicable even to large index generation functions that cannot be solved in a reasonable time by existing heuristics. This paper shows time and space complexities of the proposed heuristic, and experimental results using some large examples to show its efficiency.**

Keywords-**Heuristic; balanced decision tree; linear decomposition; index generation functions; logic design.**

I. Introduction

Pattern matching and text search are basic operations used in many applications, such as detection of computer viruses and packet classification. Logical behavior of these operations can be specified as *index generation functions* [4], [5]. Since index generation functions are frequently updated particularly in the above network applications, a memory-based design of index generation functions is desired.

To design index generation functions using memory efficiently, a method using *linear decomposition* [2], [12] of index generation functions has been proposed [7]. This method realizes an index generation function $f(X)$ using two blocks L and G, as shown in Fig. 1. The first block L realizes linear functions with EXOR gates, registers, and multiplexers, and the second one G realizes a general function with a $(2^p \times q)$-bit memory, where p is the number of linear functions, and q is the number of bits needed to represent function values.

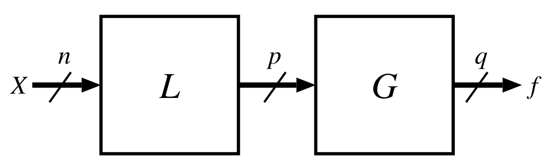

Figure 1. Linear decomposition of an index generation function.

Table I
Example of index generation function.

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

In this design method, minimization of p is important to reduce size of the memory for G. Thus, various heuristics for minimization have been proposed [6], [7], [9], [10]. However, for larger index generation functions, more efficient minimization heuristics are still required. Hence, in this paper, we propose a heuristic with smaller time and space complexities than the existing heuristics. The proposed heuristic is useful not only to find good linear decompositions of large index generation functions, but also to investigate a trade-off between complexity of L and memory size of G for large index generation functions.

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 shows our heuristic to solve it. Section IV shows experimental results from practical examples, and Section V concludes the paper.

II. Preliminaries

We briefly define index generation functions [4], [5] and their linear decompositions [2], [7], [12].

Definition 1: An **incompletely specified index generation function**, or simply **index generation function**, $f(x_1,x_2,\ldots,x_n)$ is a multi-valued function, where k assignments of values to binary variables $x_1,x_2,\ldots,$ and x_n map to $K=\{1,2,\ldots,k\}$. 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 $x_1,x_2,\ldots,$ and x_n and K. Other assignments are left unspecified. The k assignments of values to $x_1,x_2,\ldots,$ and x_n are called a set of **registered vectors**. K is called a set of **indices**. $k=|K|$ is called **weight** of the index generation function f.

Example 1: Table I shows a 4-variable index generation function with weight four. Note that, in this function, input

Table II
GENERAL FUNCTIONS g_1 AND g_2 IN LINEAR DECOMPOSITION OF f.

y_1	y_2	g_1	g_2
0	0	1	2
0	1	2	1
1	0	3	3
1	1	4	4

values other than 0001, 0010, 0100, and 1101 are NOT assigned to any function values. (End of Example)

An arbitrary n-variable index generation function with weight k can be realized by a $(2^n \times q)$-bit memory, where $q = \lceil \log_2(k+1) \rceil$. To reduce the memory size, linear decomposition is effective [7].

Definition 2: **Linear decomposition** of an index generation function $f(x_1, x_2, \ldots, x_n)$ is a representation of f using a general function $g(y_1, y_2, \ldots, y_p)$ and linear functions y_i:

$$y_i(x_1, x_2, \ldots, x_n) = c_{i1}x_1 \oplus c_{i2}x_2 \oplus \ldots \oplus c_{in}x_n \quad (i = 1, 2, \ldots, p),$$

where $c_{ij} \in \{0, 1\}$ $(j = 1, 2, \ldots, n)$, 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, $\sum_{j=1}^{n} c_{ij}$ is called a **compound degree** of y_i, denoted by $deg(y_i)$, where c_{ij} is viewed as an integer, and $\sum$ is integer sum.

Example 2: The index generation function f in Example 1 can be represented by $y_1 = x_2$, $y_2 = x_1 \oplus x_3$, and $g_1(y_1, y_2)$ shown in Table II. (i.e. all four 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. f can be also represented by $y_1 = x_2$, $y_2 = x_4$, and $g_2(y_1, y_2)$ in the same table. In this case, both $deg(y_1)$ and $deg(y_2)$ are 1. In either case, f can be realized by the architecture in Fig. 1 with a $(2^2 \times 3)$-bit memory. (End of Example)

In this way, by using linear decomposition, memory size needed to realize an index generation function can be reduced. But, to realize a compound variable with compound degree d, $(d-1)$ 2-input EXOR gates are required. Thus, lower compound degree is desirable when memory size is equal.

III. MINIMIZATION OF NUMBER OF LINEAR FUNCTIONS

This section formulates the minimization problem of the number of linear functions, and presents a heuristic to solve the problem.

A. Formulation of Minimization Problem

Since the architecture in Fig. 1 realizes an index generation function with EXOR gates and a $(2^p \times q)$-bit memory, to obtain an optimum realization of an index generation function, we have to solve the following problem:

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 the minimum, and compound degrees are at most t. For linear decompositions with the same value of p, the one with the lowest compound degree is optimum.

As will be shown later, increasing the compound degree tends to reduce the value of p. But, it is limited to t because of delay and area of the circuit L to realize linear functions.

Example 3: For linear decompositions of f in Example 2, the decomposition with $y = x_2$, $y_2 = x_4$, and $g_2(y_1, y_2)$ is optimum. (End of Example)

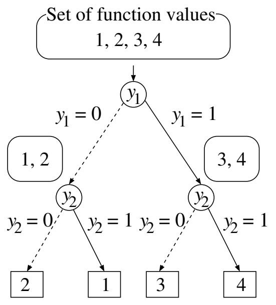

Figure 2. Point of view as a binary decision tree.

B. Heuristic for Minimization Problem

Since the solution space of Problem 1 is too large to solve the problem exactly, various heuristics have been proposed [6], [7], [9], [10]. However, for larger index generation functions, a heuristic that finds a good linear decomposition with smaller time and space complexities is still required. To reduce both time and space complexities, we propose a heuristic that searches *only promising linear decompositions* efficiently.

To solve Problem 1, we have to find the smallest set of compound variables such that k distinct combinations of values of the compound variables have a *one-to-one correspondence to a set of indices* for f. In other words, we have to find the fewest compound variables that *divide a set of indices into singletons* (sets consisting of exactly one index). This corresponds to *constructing a binary decision tree with the smallest height*. This is the key idea for our heuristic.

Example 4: As shown in Fig. 2, finding the optimum linear decomposition in Example 3 can be considered as constructing a binary decision tree with the smallest height that divides a set of indices into singletons by compound variables y_1 and y_2. (End of Example)

Since a binary decision tree with the smallest height is a *balanced decision tree*, we propose a heuristic to construct a balanced decision tree using compound variables. To do this, a set of indices is divided into two subsets with balanced size recursively by compound variables. Before describing requirements to find such compound variables, we define additional terms.

Heuristic 1: Heuristic to find a good 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. Find $x_i$ with the minimum $cost(\mathcal{P}, y \oplus x_i)$ among
   $x_1, x_2, \ldots,$ and $x_n$.
3. Replace $y$ with $y \oplus x_i$ and $deg(y)$ with $deg(y)+1$.
4. If $cost(\mathcal{P}, y)$ is smaller than the previous smallest
   one, then $y_{opt} = y$.
5. If $cost(\mathcal{P}, y) = 0$, then terminate the heuristic.
6. Else, iterate Steps 2 to 5 until $deg(y) = t$.
```

Definition 3: An inverse function of a general 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 B^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 4: 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 1s included in** y_i.

Example 5: For $g_2(y_1, y_2)$ in Table II, its inverse functions to y_1 and y_2 are $(g_2^{-1})_1(z)$ and $(g_2^{-1})_2(z)$, respectively. We have $(g_2^{-1})_1(2) = 0$, $(g_2^{-1})_1(1) = 0$, $(g_2^{-1})_1(3) = 1$, and $(g_2^{-1})_1(4) = 1$. Similarly, $(g_2^{-1})_2(2) = 0$, $(g_2^{-1})_2(1) = 1$, $(g_2^{-1})_2(3) = 0$, and $(g_2^{-1})_2(4) = 1$. The cardinalities of both y_1 and y_2 are 2. (End of Example)

Theorem 1: An index generation function with weight $k = 2^m$, where m is a positive integer, can be represented by a completely balanced binary decision tree with m compound variables, if and only if there exist m compound variables satisfying the following requirement: For all subsets Y of the set of the m compound variables, (1) holds,

$$\left| \bigcap_{y_i \in Y} ON(y_i) \right| = 2^{m-h}, \tag{1}$$

where $h = |Y|$.
(Proof) See Appendix A.

Although compound variables satisfying the above requirements are ideal to construct a balanced decision tree, weights of index generation functions are not always 2^m, and only limited functions have such compound variables. In addition, finding such m compound variables is hard. Thus, to construct a balanced decision tree, we heuristically select a compound variable closer to the ideal one satisfying the above requirements by minimizing the following cost function over the unselected compound variables:

$$cost(\mathcal{P}, y_i) = \sqrt{\sum_{S \in \mathcal{P}} \left(\frac{|S|}{2} - |S \cap ON(y_i)| \right)^2}, \tag{2}$$

Heuristic 2: Heuristic to find a good 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 $S \in \mathcal{P}$ with $y_i$.
4. Update $\mathcal{P}$ with the divided subsets.
5. $i = i + 1$.
6. Iterate Steps 2 to 5 until $|\mathcal{P}| = k$.
```

where $\mathcal{P}$ is a partition of a set of indices with already selected compound variables. Initially, when there are no selected compound variables, $\mathcal{P}$ is the trivial partition consisting of a single block containing all indices.

This cost function is defined with the view of a Euclidean distance (2-norm) between y_i and an optimum compound variable that divides all subsets into halves. A compound variable with a small cost function tends to be a member of the optimum set of variables. Heuristic 1 shows a heuristic to find a good compound variable using the cost function. In the heuristic, $\mathcal{P}$ is a partition of a set of indices with already selected compound variables, and t is the maximum compound degree.

Since Heuristic 1 selects promising variables x_i using the cost function, and compounds only those variables, it can find a good compound variable with small time and space complexities. In Steps 2 and 4, when the cost function is equal, the heuristic selects a compound variable that divides subsets into smaller subsets. That is, the following is used as the second cost function:

$$\max_{S \in \mathcal{P}} (\max(|S \cap ON(y_i)|, |S \setminus ON(y_i)|)).$$

In addition, a compound variable with smaller compound degree is prioritized since the heuristic begins with the smallest compound degree.

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

Although there exist heuristics to construct optimum decision trees and diagrams based on linear functions [1], [3], their objective functions for optimization is essentially different from the proposed heuristic. Therefore, even if the existing heuristics could be applied to Problem 1, the proposed heuristic is more efficient since it is a dedicated heuristic to solve Problem 1. Detailed comparison is omitted due to the page limitation.

C. Time and Space Complexities of the Heuristic

Since the cost function $cost(\mathcal{P}, y_i)$ is computed by checking which subset $S \in \mathcal{P}$ each index belongs to and whether

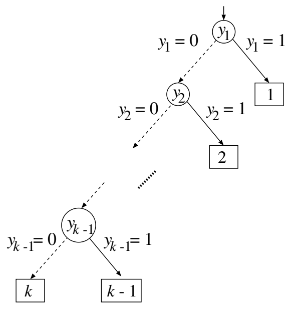

Figure 3. Imbalanced decision tree with height k.

Table III
TIME AND SPACE COMPLEXITIES OF HEURISTICS.

Heuristics	Time	Space
Ours	$O(nk\log(k))$	$O(nk)$
RM2011 [6]	$O(n^5k)$	$O(nk)$
ASP-DAC2012 [7]	$O(n^tk\log(k))$	$O(n^tk)$
IEICE2014 [9]	N/A	$O(nk^2)$
ISMVL2015 [10]	$O(n^sk\log(k))$	$O(nk)$

it belongs to $ON(y_i)$, its time complexity is $O(k)$. In Step 2 of Heuristic 1, the cost function is invoked n times to find the best x_i among x_1 to x_n. Since Step 2 is iterated t times, the time complexity of Heuristic 1 is

$$O(k) \times n \times t = O(knt).$$

Similarly to the cost function, the time complexity for dividing subsets of $\mathcal{P}$ in Step 3 of Heuristic 2 is $O(k)$. Heuristic 2 invokes this computation and Heuristic 1 iteratively until $|\mathcal{P}| = k$. Since the number of iterations in Heuristic 2 is $k-1$ in the worst case, its time complexity is $(O(knt) + O(k)) \times (k-1) = O(k^2nt)$. In this case, an extremely imbalanced decision tree is constructed as shown in Fig. 3. However, it rarely happens because the heuristic intends to construct a balanced decision tree. Thus, the number of iterations is $\log(k)$ on the average, resulting in a time complexity of $O(ntk\log(k))$. Since t is a small constraint parameter rather than the size of the problem, it can be considered as a constant. Therefore, the time complexity of Heuristic 2 is

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

Memory sizes to store subsets of indices and to store selected compound variables are $O(k)$ and $O(n)$, respectively. On the other hand, memory size to store given registered vectors is $O(kn)$. Since other working spaces require much less memory size, the space complexity of Heuristic 2 is

$$O(kn).$$

Table III compares our heuristic with existing heuristics, in terms of time and space complexities. Table III shows that our heuristic can solve even larger instances of Problem 1 (e.g., $n = 40$ and $k = 1,000,000$) with a computation time that is several orders of magnitude smaller and with smaller memory size than previous heuristics.

Table IV
COMPARISON IN TERMS OF QUALITY OF SOLUTIONS.

Benchmarks	k	Heur.	Compound degree t					
			1	2	3	4	5	6
1-out-of-20 code	20	[7]	19	14	10	8	7	6
		Ours	19	14	10	8	7	6
3-out-of-20 code	1,140	[7]	19	17	14	12	12	11
		Ours	19	17	14	13	13	13
IP address table	4,591	[7]	24	20	19	18	18	18
		Ours	23	21	20	20	20	20
IP address table	7,903	[7]	23	21	20	20	20	20
		Ours	23	22	22	22	21	21
English words (ListB)	3,366	[7]	31	21	19	17	17	-
		Ours	31	21	20	19	19	19
English words (ListC)	4,705	[7]	37	24	20	19	18	-
		Ours	37	24	21	20	20	20

IV. EXPERIMENTAL RESULTS

The proposed heuristic is 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, and C-compiler: gcc -O2 (version 4.1.2).

A. *On Quality of Solutions*

Among the existing heuristics in Table III, the heuristic presented in ASP-DAC2012 [7] produces the best solutions (i.e., the smallest numbers of compound variables). Thus, we compare our heuristic with it in terms of quality of solutions. Table IV compares the numbers of compound variables selected by both heuristics for some benchmarks shown in [7].

Even though the search space of our heuristic is much smaller than that of the existing heuristic, the number of compound variables selected by our heuristic is not much larger than that selected by the existing heuristic, as shown in Table IV. Particularly, for the benchmark of 1-out-of-20 code, our heuristic found the exact minimum number of compound variables [11] when $t = 1$ to 8 except for $t = 2$. This shows that our heuristic finds good solutions efficiently by pruning unpromising solutions heuristically.

B. *Results for Large Problems*

To show that our heuristic can be applied to larger problems, we used the following three examples: 1) random *social security and tax numbers* (SST numbers) in Japan [13]; 2) the bible [17]; and 3) the US constitution [18] including amendments [19], [20]. 1) is used as a numeric example with large k, and 2) and 3) are examples of text search with large n. For information on how to generate index generation functions from these examples, see Appendix B. Table V shows computation time of our heuristic and the number of compound variables for these examples.

For each example function, we can predict the number of compound variables using Property 1 shown in [8].

Table V
COMPUTATION TIME IN SECONDS AND NUMBER OF COMPOUND VARIABLES.

Computation time			Compound degree t									
Benchmarks	n	k	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
SST numbers	48	1,000,000	81.30	165.91	250.50	333.25	406.49	492.62	568.60	639.45	712.01	778.98
Bible	560	20,827	3.75	6.38	9.09	11.80	14.42	17.13	19.78	22.38	25.05	27.72
US constitution	1,512	253	0.05	0.08	0.10	0.12	0.14	0.16	0.18	0.20	0.23	0.25

Number of compound variables	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
SST numbers	42	37	36	35	35	35	35	35	35	35
Bible	44	31	28	27	25	25	25	24	24	24
US constitution	15	12	11	11	10	10	10	10	11	11

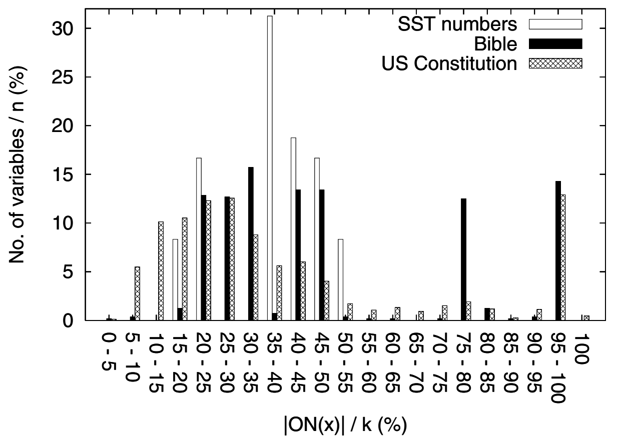

Figure 4. Histograms for the number of variables x_i and $|ON(x_i)|$
.

Property 1: [8] When n is sufficiently large and $k \ll 2^n$, most index generation functions with weight k can be represented by $L-1$, L, or $L+1$ compound variables, where $L = 2\lceil \log_2(k+1) \rceil - 4$.

For the function of the SST numbers, the predicted number of compound variables is $L-1 = 2 \times \lceil \log_2(1{,}000{,}001) \rceil - 5 = 35$; for the function of the bible, it is $L-1 = 2 \times \lceil \log_2(20{,}828) \rceil - 5 = 25$; and for the function of the US constitution, it is $L-1 = 2 \times \lceil \log_2(254) \rceil - 5 = 11$. As shown in Table V, our heuristic achieves those numbers or even smaller numbers when $t \geq 3$ for the SST numbers, $t \geq 5$ for the bible, and $t \geq 3$ for the US constitution.

These results show that even if n and k are large, our heuristic finds a good solution within a reasonable time.

C. Number of Compound Variables vs. Compound Degree

Fig. 4 shows distributions of $|ON(x_i)|$ for the example index generation functions. In the figure, the horizontal axis shows ratios of the number of 1s included in original variables x_i in registered vectors, and the vertical axis shows ratios of the number of x_i having the same ratio of $|ON(x_i)|$.

As shown in Fig. 4, the example functions have few variables x_i with $|ON(x_i)|/k \simeq 0.5$ that can divide a set of indices into halves. Thus, many variables are required when $t=1$, as shown in Table V. However, for such functions, we can produce variables with $|ON(x_i)|/k \simeq 0.5$ by increasing the compound degree, and thus, the number of variables can be reduced. As shown in Table V, however, it is not reduced so much for $t > 5$. This means that practically effective compound degree is at most 5 for the example index generation functions.

V. CONCLUSION AND COMMENTS

This paper proposes a balanced decision tree based heuristic to minimize the number of compound variables for linear decomposition of index generation functions. Since time and space complexities of the proposed heuristic are smaller than those of existing heuristics, it can be applied to larger index generation functions. Experimental results show that the proposed heuristic finds a good solution that is close to the best solution ever found, even though its search space is much smaller. And, this paper also shows a relation between the number of compound variables and compound degrees t, and shows that the number of compound variables is reduced by increasing t until $t=5$.

The proposed heuristic would be helpful for exact minimization algorithm based on a branch-and-bound method because we can prune unpromising solutions using the heuristic. We will study an exact minimization algorithm based on a branch-and-bound method. In addition, we will study a more efficient cost function than (2).

ACKNOWLEDGMENTS

This research is partly supported by the JSPS Kaken (C), 25330071 and 26330072, and Hiroshima City University Grant for Academic Research (General Studies), 2015. The reviewers' comments were helpful in improving the paper.

REFERENCES

[1] S. Aborhey, "Binary decision tree test functions," *IEEE Trans. on Comput.*, Vol. 37, No. 11, pp. 1461–1465, Nov 1988.

[2] 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.

[3] C. Meinel, F. Somenzi, and T. Theobald, "Linear sifting of decision diagrams," *34th Design Automation Conference*, pp. 202–207, 1997.

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

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

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] D. Varma and E. Trachtenberg, “Design automation tools for efficient implementation of logic functions by decomposition,” *IEEE Trans. on CAD*, Vol. 8, No. 8, pp. 901–916, Aug. 1989.

[13] *The Social Security and Tax Number System* Cabinet Secretariat, http://www.cas.go.jp/jp/seisaku/bangoseido/english.html, Oct. 14, 2015.

[14] *Document on the Social Security and Tax Number System* (in Japanese), Office for the Social Security and Tax Number System, Minister’s Secretariat, Cabinet Office and Social Security Reform Office, Cabinet Secretariat, http://www.cas.go.jp/jp/seisaku/bangoseido/pdf/gaiyou_siryou.pdf, Oct. 14, 2015.

[15] *Lows and Regulations for Social Security and Tax Numbers* (in Japanese), Ministry of Internal Affairs and Communications, http://law.e-gov.go.jp/announce/H26F11001000085.html, Oct. 14, 2015.

[16] *Lows and Regulations for Residents Identification Numbers* (in Japanese), Ministry of Internal Affairs and Communications, http://law.e-gov.go.jp/htmldata/H11/H11F04301000035.html, Oct. 14, 2015.

[17] *Open Source Bible Data, The King James Version*, Internet Sacred Text Archive, http://www.sacred-texts.com/bib/osrc/, Oct. 27, 2015.

[18] *The Constitution of the United States: A Transcription*, U.S. National Archives and Records Administration, http://www.archives.gov/exhibits/charters/constitution.html, Oct. 13, 2015.

[19] *The U.S. Bill of Rights: A Transcription*, U.S. National Archives and Records Administration, http://www.archives.gov/exhibits/charters/constitution.html, Oct. 14, 2015.

[20] *The Constitution: Amendments 11-27*, U.S. National Archives and Records Administration, http://www.archives.gov/exhibits/charters/constitution.html, Oct. 14, 2015.

Appendix

A. Proof for Theorem 1

(if) Assume that there exist m compound variables satisfying the requirement. Then, since for any compound variable y_i, $|ON(y_i)| = 2^m/2$ holds, each y_i can divide a set of indices into halves by $y_i = 0$ and $y_i = 1$. And, each subset of 2^{m-l} indices obtained by a partition with l variables can be further divided into halves by y_{l+1} because $|\bigcap_{i=1}^{l+1} ON(y_i)| = 2^{m-l}/2$ holds for $l+1$ variables. Since the m compound variables can divide a set of 2^m indices into two equal-sized subsets recursively, resulting in 2^m singletons. By considering partitions with each variable as a non-terminal node, and each singleton as a terminal node, we have a completely balanced binary decision tree with height m.

(only if) Assume that a completely balanced binary decision tree can be constructed. Then, in the tree, a set of indices is divided into halves recursively by $y_i = 0$ and $y_i = 1$, as shown in Fig. 2. Thus, for any compound variable y_i, $|ON(y_i)| = 2^m/2$ holds. And, since a set of indices is divided into equal-sized subsets recursively, for any h variables, $|\bigcap_{i=1}^{h} ON(y_i)| = 2^m/2^h$ holds. ∎

B. How to Generate Index Generation Functions

In 2015, Japan introduced the new *“social security and tax number” (SST number)* to replace the old “resident’s identification number” (RIN) [13]. The new SST number is a 12-digit decimal number $(d_{11}\ d_{10}\ \ldots\ d_1\ d_0)_{10}$, and it consists of a single check digit d_0 and an 11-digit number $(d_{11}\ d_{10}\ \ldots\ d_1)_{10}$ that is generated from the resident’s 11-digit RIN [14]. The RIN, in turn, consists of a single check digit and a 10-digit number that is randomly generated to prevent the identification of an individual from the number [16]. Thus, we randomly generated an 11-digit number, and attached a check digit to its least significant digit to generate an SST number. The check digit d_0 is obtained by the following computation [15]:

$$\begin{aligned} d_0 &= \begin{cases} 0 & (r \le 1) \\ 11 - r & (\text{otherwise}) \end{cases} \\ r &= \left(\sum_{i=7}^{11} d_i \times (i-5) + \sum_{i=1}^{6} d_i \times (i+1) \right) (\bmod 11). \end{aligned}$$

We randomly generated 1,000,000 distinct SST numbers, and assigned an index from 1 to 1,000,000 to each number. By converting each digit into a 4-bit number, we generated 1,000,000 registered vectors, each with $4 \times 12 = 48$ bits.

The bible [17] consists of 31,102 verses, and we took the first 80 characters from each verse excluding its reference number and verses shorter than 80 characters. Then, we obtained 20,827 distinct strings of the characters by removing the duplicated strings. By assigning an index from 1 to 20,827 to each string, and converting each character into a 7-bit binary number using the ASCII code, we generated the second index generation function.

The US constitution [18] consists of 256 sentences, including amendments [19], [20] but excluding headings. Similarly to the bible, we took the first 216 characters from each sentence, and then, we obtained 253 strings by removing the duplicated strings. For sentences shorter than 216 characters, blanks are padded to make their length 216. By assigning an index from 1 to 253 to each string, and converting each character into a 7-bit binary number using the ASCII code, we generated the third index generation function.