PAPER Special Section on Multiple-Valued Logic and VLSI Computing

A Balanced Decision Tree Based Heuristic for Linear Decomposition of Index Generation Functions*

Shinobu NAGAYAMA[†a)], **Tsutomu SASAO**[††b)], *Members*, *and* **Jon T. BUTLER**[†††c)], *Nonmember*

SUMMARY Index generation functions model content-addressable memory, and are useful in virus detectors and routers. Linear decompositions yield simpler circuits that realize index generation functions. This paper proposes a balanced decision tree based heuristic to efficiently design linear decompositions for index generation functions. The proposed heuristic finds a good linear decomposition of an index generation function by using appropriate cost functions and a constraint to construct a balanced tree. 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.

key words: *index generation functions, linear decomposition, incompletely specified functions, balanced decision tree, content-addressable memory, logic design, heuristic*

1. Introduction

Index generation functions [6], [7] are logical models of pattern matching and text search that are used in many applications, such as detection of computer viruses and packet classification. These network applications require not only fast computation of index generation functions but also their frequent update, since rules (patterns) for computer viruses and packet classification are frequently updated. Thus, a memory-based design of index generation functions is desired to satisfy the need for both fast computation and quick update.

To design index generation functions using memory efficiently, a method using *linear decomposition* [2], [5] of index generation functions has been proposed [9]. This method realizes an index generation function $f(x_1, x_2, \ldots, x_n)$ using two blocks L and G, as shown in Fig. 1. The first block L realizes linear functions y_i ($i = 1, 2, \ldots, p$) with EXOR gates, registers, and multiplexers, and the second one G realizes a general function using a $(2^p \times q)$-bit memory [9], where p is the number of linear functions y_i, and q is the number of bits needed to represent function values. Because of the need to reduce the memory size of G, we want p to be as small as possible. So, $p < n$.

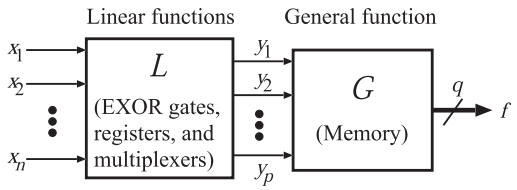


Fig. 1 Linear decomposition of an index generation function.

In this design method, minimization of p is important to reduce size of the memory for G. Thus, an exact minimization algorithm [13] and various heuristics for minimization have been proposed [8], [9], [11], [12]. 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 for 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 2 defines index generation functions and linear decomposition. Section 3 formulates the minimization problem of the number of linear functions, and shows our heuristic to solve it. Section 4 shows experimental results from practical examples, and Sect. 5 concludes the paper.

2. Preliminaries

We briefly define index generation functions [6], [7] and their linear decompositions [2], [5], [9].

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. An

Manuscript received October 14, 2016.
Manuscript revised February 20, 2017.
Manuscript publicized May 19, 2017.

[†]The author is with the Department of Computer and Network Engineering, Hiroshima City University, Hiroshima-shi, 731–3194 Japan.
[††]The author is with the Department of Computer Science, Meiji University, Kawasaki-shi, 214–8571 Japan.
[†††]The author is with the Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA 93943-5121 USA.

*This paper is an extension of [4] (improvement of the heuristic, new experimental results, and verification of the time complexity are added).

a) E-mail: s_naga@hiroshima-cu.ac.jp
b) E-mail: sasao@cs.meiji.ac.jp
c) E-mail: jon_butler@msn.com

DOI: 10.1587/transinf.2016LOP0013

Table 1 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
1	0	0	0	3
1	1	1	0	4

Table 2 General functions g_1 and g_2 in linear decomposition of f.

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

assignment of values to $x_1, x_2, \ldots$, and x_n is called a **registered vector**. K is called a set of **indices**. $k = |K|$ is called **weight** of the index generation function f.

Definition 2: Let K be a set of indices. If $K = \{K_1\} \cup \{K_2\} \cup \ldots \cup \{K_u\}$ and $\{K_i\} \cap \{K_j\} = \emptyset$ for $\forall i, j$ $(i \neq j)$, then $\mathcal{P} = \{K_1, K_2, \ldots, K_u\}$ is a **partition** of indices, and $|\mathcal{P}|$ denotes the number of the subsets K_i (i.e., $|\mathcal{P}| = u$).

Example 1: Table 1 shows a 4-variable index generation function with weight four. Note that, in this function, input values other than 0001, 0010, 1000, and 1110 are NOT assigned to any function values. □

An arbitrary n-variable index generation function with weight k can be realized by a $(2^n \times q)$-bit memory, where $q = \lceil \log_2(k+1) \rceil$. Linear decomposition can be used to reduce memory size [9].

Definition 3: 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 an integer sum.

Example 2: The index generation function f in Example 1 can be represented by $y_1 = x_1$, $y_2 = x_2 \oplus x_4$, and $g_1(y_1, y_2)$ shown in Table 2. (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 $y_1 = x_1$, $y_2 = x_3$, 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. □

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

3. Minimization of Number of Linear Functions

In this section, a formulation of the minimization problem is presented (reducing the number of linear functions), as well as a heuristic to solve the problem.

3.1 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, we can state the minimization problem as follows:

Problem 1: Given an index generation function f and an integer t, find a linear decomposition of f such that the number of linear functions p is the minimum, and the compound degrees are at most t. Among the linear decompositions with the same value of p, those with the lowest compound degree are optimum.

As will be shown later, increasing the compound degree t tends to reduce the value of p. But, a point of diminishing returns, $t = t'$, is reached where a small increase in t beyond t' greatly increases the delay and/or area of the circuit L.

Example 3: For linear decompositions of f in Example 2, the decomposition with $y = x_1$, $y_2 = x_3$, and $g_2(y_1, y_2)$ is optimum. □

3.2 Heuristic for Minimization Problem

Since the solution space of Problem 1 is too large to solve the problem exactly, various heuristics have been proposed [8], [9], [11], [12]. 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). The key idea is to *construct an ordered binary decision tree with the smallest height*.

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

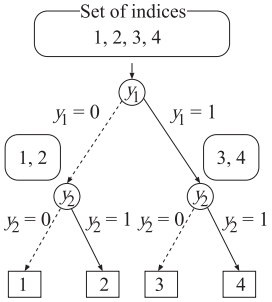


Fig. 2 Point of view as a binary decision tree.

Although there exist heuristics to construct optimum decision trees and diagrams based on linear functions [1], [3], their objective function for optimization is essentially different from that of Problem 1. Even if the existing heuristics could be applied to Problem 1, they are not always efficient. Therefore, we propose a dedicated heuristic to solve Problem 1.

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. We observed that balanced decision trees are constructed by compound variables that recursively divide a set of indices into two subsets with balanced size. That is, compound variables that recursively bisects a set of indices are a solution of Problem 1. To find such compound variables efficiently, we introduce cost functions to our heuristic.

3.2.1 Cost Functions to Find Compound Variables

Before describing cost functions to find compound variables, we define additional terms and show a requirement for ideal compound variables.

Definition 4: 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 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 1s included in** y_i.

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

On compound variables that construct a balanced decision tree of an index generation function, the following theorem holds.

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 any subset Y of the set of the m compound variables,

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

holds, 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, we heuristically find compound variables closer to the ideal ones satisfying the above requirements by using the following cost function:

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

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. Then, our heuristic sequentially selects compound variables that minimize this cost function from among the unselected compound variables.

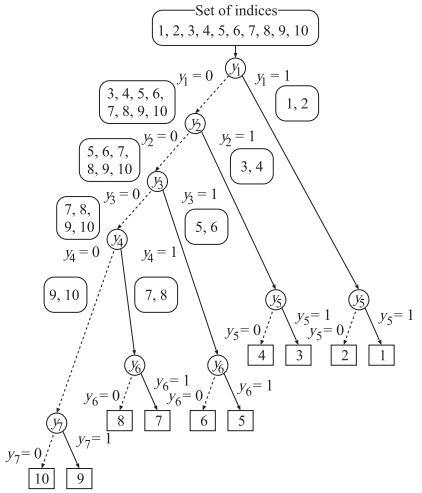
The cost function (1) is defined with the view of a Euclidean distance (2-norm) between y_i and an optimum compound variable that divides all subsets into halves. Thus, a compound variable with a small value of (1) tends to be a member of the optimum set of variables. When values of the cost function are equal among compound variables, selecting a compound variable that divides subsets into smaller subsets is desired to divide a set of indices into singletons as quickly as possible. To select such a variable, we use the following as the second cost function:

$$cost2(\mathcal{P}, y_i) = \max_{S \in \mathcal{P}} (\max(|S \cap ON(y_i)|, |S \setminus ON(y_i)|)). \tag{2}$$

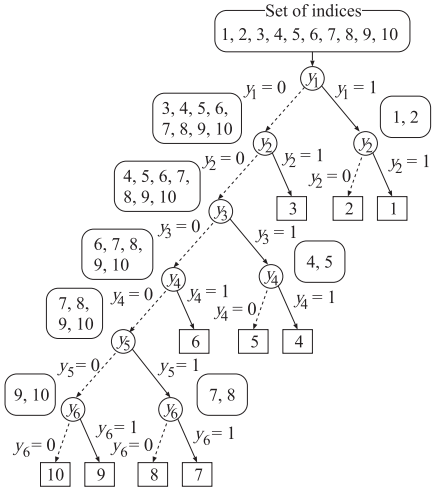
Since this cost function computes the size of the largest subset among subsets divided by the value (0 and 1) assigned to y_i, a compound variable with a smaller value of (2) is better.

3.2.2 Constraint to Construct Balanced Tree

When the number of 1s included in any compound variable is much smaller than $\frac{k}{2}$, the set of indices is inescapably divided into a large subset and a small subset. Then, only the large subset is recursively divided into imbalanced subsets, as suggested by the example in Fig. 3 (a). This is because



(a) Without constraint



(b) With constraint

Fig. 3 Binary decision trees for 1-out-of-10 code.

the cost functions (1) and (2) tend to assign the highest priority to divide the largest subset. Since only one subset is divided by a compound variable (i.e., a subset *monopolizes a compound variable*), other subsets are left undivided. As a result, many compound variables are needed to divide many small subsets into singletons.

In this way, when the number of subsets becomes larger than the number of 1s in a compound variable, the number of needed compound variables is also likely to be larger. Thus, we introduce a *constraint that does not allow a subset to monopolize a compound variable* when the number of subsets becomes larger than the number of 1s in a compound variable. The number of 1s in a compound variable is estimated as follows:

$$t \times (\text{average number of 1s in an original variable } x_i),$$

where t is a compound degree given as an input of Problem 1.

Example 6: Consider linear decompositions of an index generation function shown in Table 3 with a compound degree $t = 2$. For this function, the number of 1s in a compound variable is, at most two, since $t = 2$ and each original variable x_i has only one 1. Thus, only up to two indices can be extracted from subsets when we divide subsets of indices with a compound variable. When we use only the cost functions (1) and (2), a binary decision tree shown in Fig. 3 (a) is produced, and the number of compound variables needed to divide a set of indices into singletons is 7. On the other hand, when we add the above constraint, monopolization of a compound variable by a subset is constrained, and a tree shown in Fig. 3 (b) is produced. Then, the number of compound variables needed is 6. □

Table 3 Benchmark 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

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. Let $\mathcal{P}_1$ be a set consisting of singletons included in $\mathcal{P}$.
3. Let $X = \{x_1, x_2, \ldots, x_n\}$.
4. If the estimated number of 1s in a compound variable is smaller than or equal to $|\mathcal{P} \setminus \mathcal{P}_1|$, then x_i is removed from X such that there exists an $S \in \mathcal{P}$ monopolizing a compound variable $y \oplus x_i$ for $i = 1, 2, \ldots, n$ (**constraint**).
5. Find x_i with the minimum $cost(\mathcal{P}, y \oplus x_i)$ among X. $cost2(\mathcal{P}, y \oplus x_i)$ is used to break a tie.
6. Replace y with $y \oplus x_i$ and $deg(y)$ with $deg(y) + 1$.
7. If $cost(\mathcal{P}, y)$ is smaller than the previous smallest one, then $y_{opt} = y$. $cost2(\mathcal{P}, y)$ is used to break a tie.
8. If $cost(\mathcal{P}, y) = 0$, then terminate the heuristic.
9. Else, iterate Steps 3 to 8 until $deg(y) = t$.

3.2.3 Balanced Decision Tree Based Heuristic

Heuristic 1 shows a heuristic to find a good compound variable using the cost functions and the constraint. 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 functions, and compounds only those variables, it can find a good compound variable with small time and space complexities. Since the heuristic begins with the smallest compound degree, a compound variable with smaller compound degree is prioritized when values of the cost functions are equal among compound variables.

By using Heuristic 1 iteratively until a binary decision

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.	Replace $\mathcal{P}$ with the divided subsets derived in Step 3.
5.	$i = i + 1$.
6.	Iterate Steps 2 to 5 until $\|\mathcal{P}\| = k$.

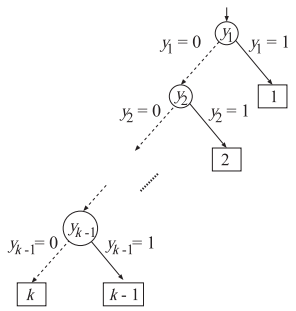


Fig. 4 Imbalanced decision tree with height $k - 1$ [4].

tree is constructed, 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 recursively using compound variables selected by Heuristic 1, and it terminates when a set of indices is divided into singletons.

3.3 Time and Space Complexities of the Heuristic

Since the cost functions $cost(\mathcal{P}, y_i)$ and $cost2(\mathcal{P}, y_i)$ are computed by checking which subset $S \in \mathcal{P}$ each index belongs to and whether it belongs to $ON(y_i)$, their time complexities are $O(k)$. The constraint is computed similarly, and thus, its time complexity is also $O(k)$. In Heuristic 1, the cost functions and the constraint are invoked at most n times to find the best x_i among x_1 to x_n. Since this computation is iterated t times, the time complexity of Heuristic 1 is

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

Similarly to the cost functions, 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. 4. However, it rarely happens because the heuristic intends to construct a balanced decision tree. Since the number of iterations is $O(\log(k))$ on the average, the time complexity of Heuristic 2 is

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

As for the space complexity, Heuristic 2 needs to store a partition of indices $\mathcal{P}$, selected compound variables, and given registered vectors. Note that storing a tree structure is unnecessary because we can divide subsets of indices if only a partition of indices $\mathcal{P}$ is stored. Memory size (the number of words) to store $\mathcal{P}$ is $O(k)$. Memory size needed for selected compound variables is $O(nt)$ because the upper bound on the number of compound variables is n, each compound variable consists of at most t original variables, and Heuristic 2 outputs original variables for each compound variable. Memory size to store given registered vectors is $O(kn)$. Thus, total memory size for the above is $O(kn) + O(nt) + O(k)$. Since t is much smaller than k, and other working spaces require much less memory size, the space complexity of Heuristic 2 is

$$O(kn).$$

Table 4 Time and space complexities of heuristics.

Heuristics	Time	Space
Ours	$O(ntk \log(k))$	$O(nk)$
RM2011 [8]	$O(n^5k)$	$O(nk)$
ASP-DAC2012 [9]	$O(n^tk \log(k))$	$O(n^tk)$
IEICE2014 [11]	N/A	$O(nk^2)$
ISMVL2015 [12]	$O(n^sk \log(k))$	$O(nk)$

n = the number of variables.
k = the number of indices.
t = the maximum compound degree.

Table 4 compares our heuristic with four existing heuristics, in terms of time and space complexities. Since those four existing heuristics are based on quite different approaches than our balanced tree based approach, we briefly introduce the four heuristics, as follows: The first one, denoted by RM2011 [8], is an iterative improvement method. It produces an initial solution using compound variables with small compound degrees, and then improves the solution using heuristic transformations iteratively until the solution cannot be improved anymore. Thus, a long computation time is required until convergence of solutions.

The second one, denoted by ASP-DAC2012 [9], begins with generating all the possible compound variables whose compound degrees are up to t, and selects compound variables with a greedy method. Since there are $O(n^t)$ compound variables, computation time to generate them and memory size to store them are large when n is large.

The third one, denoted by IEICE2014 [11], is based on an approach using a matrix, called the difference matrix. It finds compound variables needed to distinguish two indices using the difference matrix, and then heuristically selects good variables out of them. Since the size of the difference matrix is $O(nk^2)$, this method suffers from memory overflow when k is large.

The last one, denoted by ISMVL2015 [12], is another iterative improvement method. It selects s variables out of n variables, produces compound variables using the s variables, and then replaces the s variables with the produced compound variables. By iterating these processes, a solution continues to be improved. The number of combinations to select s variables out of n variables is $\binom{n}{s}$, that is $O(n^s)$.

Even if $s = 2$ or 3, the number of combinations is still large when n is large.

As just described, in the existing heuristics, time or space complexity is still large for large index generation functions. Table 4 shows that our heuristic has smaller time and space complexities. Thus, it can solve even large 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 the existing heuristics.

4. 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).

4.1 On Quality of Solutions

Among the existing heuristics in Table 4, the heuristic presented in ASP-DAC2012 [9] produces the best solutions (i.e., the smallest number of compound variables) known so far in most cases. Thus, we compare our heuristic with it in terms of quality of solutions. Table 5 compares the numbers of compound variables selected by both heuristics for some benchmarks shown in [9]. For all of these benchmarks, except for the m-out-of-20 code, the optimum solutions are not yet known because there is no exact optimization algorithm that can solve such large instances of Problem 1.

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 5. Particularly, for the benchmark of 1-out-of-20 code, our heuristic found the exact minimum number of compound variables [13] for $t = 1$ to 8. This shows that our heuristic finds good solutions efficiently by pruning unpromising solutions heuristically.

Table 5 Comparison in terms of the number of compound variables.

Benchmarks	k	Heur.	Compound degree t					
			1	2	3	4	5	6
1-out-of-20 code	20	[9]	19	14	10	8	7	6
		Ours	19	13	10	8	7	6
3-out-of-20 code	1,140	[9]	19	17	14	12	12	11
		Ours	19	17	14	13	13	13
IP address table	4,591	[9]	24	20	19	18	18	18
		Ours	23	21	20	20	20	20
IP address table	7,903	[9]	23	21	20	20	20	20
		Ours	23	22	22	22	21	21
English words (ListB)	3,366	[9]	31	21	19	17	17	-
		Ours	31	21	20	19	19	19
English words (ListC)	4,705	[9]	37	24	20	19	18	-
		Ours	37	24	21	20	20	20

"-" indicates that data was not available.

4.2 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 [14]; 2) the bible [18]; and 3) the US constitution [19] including amendments [20], [21]. 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 6 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 [10].

Property 1: [10] 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 corresponding to 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 corresponding to the bible, it is $L - 1 = 2 \times \lceil \log_2(20{,}828) \rceil - 5 = 25$; and for the function corresponding to the US constitution, it is $L - 1 = 2 \times \lceil \log_2(254) \rceil - 5 = 11$. As shown in Table 6, our heuristic achieves those numbers or even smaller numbers when $t \geq 4$ 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.

4.3 Verification of Time Complexity

To verify that the time complexity of Heuristic 2 is $O(ntk \log(k))$, we observed the number of iterations (i.e., loop count) in Heuristic 2 using randomly generated index generation functions for SST numbers with different weight k. We apply our heuristic to the functions with compound

Table 6 Computation time in seconds and number of compound variables.

Computation time (sec.)			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	81.31	167.19	252.92	334.85	416.14	499.50	572.41	648.32	720.80	790.85
Bible	560	20,827	3.96	6.72	9.55	12.42	15.11	17.93	20.73	23.44	26.25	29.01
US constitution	1,512	253	0.06	0.08	0.11	0.14	0.15	0.18	0.20	0.22	0.25	0.28

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

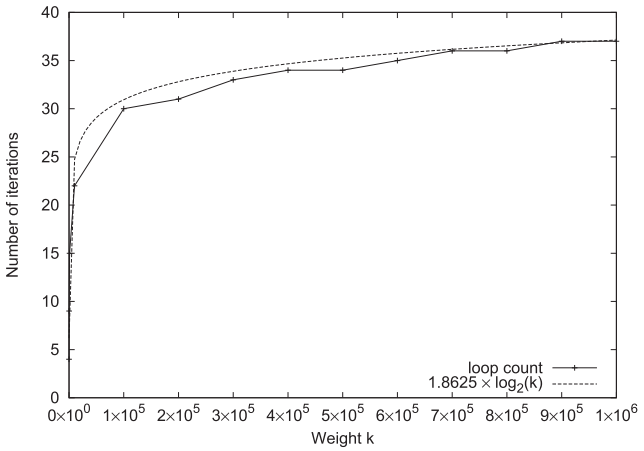


Fig. 5 The number of iterations in Heuristic 2 for different k.

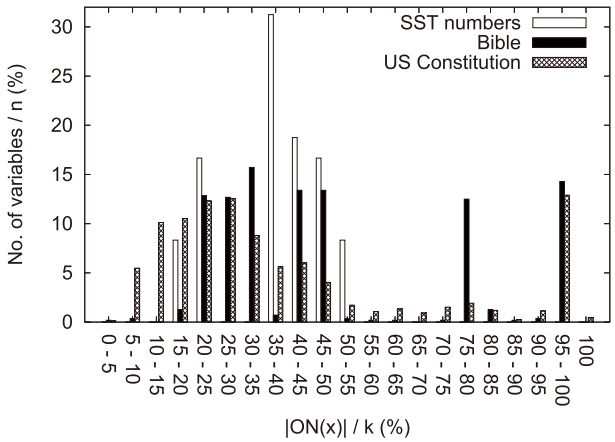


Fig. 6 Histograms for the number of variables x_i and $|ON(x_i)|$ [4].

degree $t = 2$. Figure 5 shows the number of iterations for $k = 100{,}000$ to $1{,}000{,}000$. From this graph, we can see that as the weight k increases, the number of iterations grows approximately as $1.8625 \times \log_2(k)$. Since in each iteration of Heuristic 2, Heuristic 1 whose time complexity is $O(ntk)$ is invoked, the time complexity of Heuristic 2 is $O(ntk \log(k))$.

We can also verify the time complexity by Property 1. Since Heuristic 2 iterates the computation until a binary decision tree is constructed, the number of iterations corresponds to height of the tree (i.e., the number of compound variables). From Property 1, the number of compound variables is about $2\lceil \log_2(k+1) \rceil - 4$ for most index generation functions. Therefore, we can say that the number of iterations of Heuristic 2 is also $O(\log(k))$ on the average.

4.4 Number of Compound Variables vs. Compound Degree

Figure 6 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, and the vertical axis shows ratios of the number of x_i having the same ratio of $|ON(x_i)|$.

As shown in Fig. 6, 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 6. 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 6, however, it is not reduced so much for $t > 5$. This shows that most effective value for the compound degree does not exceed 5 for the example index generation functions.

5. 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 unpromising solutions can be pruned using the heuristic. We will study an exact minimization algorithm based on a branch-and-bound method.

Acknowledgments

This research is partly supported by the JSPS Kaken (C), 16K00079 and 26330072, and Hiroshima City University Grant for Academic Research (General Studies), 2016. The reviewers' comments were helpful in improving the paper.

References

[1] S. Aborhey, "Binary decision tree test functions," IEEE Trans. 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 and its application in synthesis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.19, no.5, pp.521–533, May 2000.
[4] 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.
[5] 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.
[6] T. Sasao, Memory-Based Logic Synthesis, Springer, 2011.
[7] T. Sasao, "Index generation functions: recent developments (invited paper)," 41st International Symposium on Multiple-Valued Logic, pp.1–9, May 2011.

[8] T. Sasao, "Linear transformations for variable reduction," Reed-Muller Workshop 2011, May 2011.
[9] T. Sasao, "Linear decomposition of index generation functions," 17th Asia and South Pacific Design Automation Conference, pp.781–788, Jan. 2012.
[10] 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.
[11] 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.
[12] 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.
[13] 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.
[14] The Social Security and Tax Number System Cabinet Secretariat, http://www.cas.go.jp/jp/seisaku/bangoseido/english.html, Oct. 14, 2015.
[15] 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.
[16] 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.
[17] 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.
[18] Open Source Bible Data, The King James Version, Internet Sacred Text Archive, http://www.sacred-texts.com/bib/osrc/, Oct. 27, 2015.
[19] 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.
[20] 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.
[21] 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 values $y_i = 0$ 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. 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. ∎

Appendix 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) [14]. 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 [15]. 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 [17]. 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 [16]:

$$d_0 = \begin{cases} 0 & (r \le 1) \\ 11 - r & (\text{otherwise}) \end{cases}$$

$$r = \left(\sum_{i=7}^{11} d_i \times (i-5) + \sum_{i=1}^{6} d_i \times (i+1) \right) (\mathrm{mod}\, 11).$$

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 [18] consists of 31,102 verses, and we took the first 80 characters from each verse excluding its reference number and verses consisting of 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 [19] consists of 256 sentences, including amendments [20], [21] 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.

Shinobu Nagayama received the B.S. and M.E. degrees from the Meiji University, Kanagawa, Japan, in 2000 and 2002, respectively, and the Ph.D. degree in computer science from the Kyushu Institute of Technology, Japan, in 2004. He is now a Professor at Hiroshima City University, Japan. He received the Outstanding Contribution Paper Awards from the IEEE Computer Society Technical Committee on Multiple-Valued Logic (MVL-TC) in 2005 and 2013 for papers presented at the International Symposium on Multiple-Valued Logic in 2004 and 2012, respectively, the Young Author Award from the IEEE Computer Society Japan Chapter in 2009, and the Outstanding Paper Award from the Information Processing Society of Japan (IPS) in 2010 for a paper presented at the IPSJ Transactions on System LSI Design Methodology. His research interest includes decision diagrams, analysis of multi-state systems, logic design for index generation functions, numeric function generators, regular expression matching, and multiple-valued logic.

Tsutomu Sasao received the BE, ME, and PhD degrees in electronics engineering from Osaka University, Osaka, Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, the IBM T.J. Watson Research Center, Yorktown Heights, New York, and the Naval Postgraduate School, Monterey, California. He is now a Professor of the Department of Computer Science at the Meiji University, Kawasaki, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than nine books on logic design, including *Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis*, and *Logic Synthesis and Verification*, Kluwer Academic Publishers, 1993, 1996, 1999, and 2001, respectively. He has served as Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan, in 1998. He received the NIWA Memorial Award in 1979, Distinctive Contribution Awards from the IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003 and 2004, and Takeda Techno-Entrepreneurship Award in 2001. He has served as an Associate Editor of the *IEEE Transactions on Computers*. He is a life fellow of the IEEE.

Jon T. Butler received the BEE and MEngr degrees from Rensselaer Polytechnic Institute, Troy, New York, in 1966 and 1967, respectively. He received the PhD degree from The Ohio State University, Columbus, in 1973. Since 1987, he has been a professor at the Naval Postgraduate School, Monterey, California. From 1974 to 1987, he was at Northwestern University, Evanston, Illinois. During that time, he served two periods of leave at the Naval Postgraduate School, first as a National Research Council Senior Postdoctoral Associate (1980-1981) and second as the NAVALEX Chair Professor (1985-1987). He served one period of leave as a foreign visiting professor at the Kyushu Institute of Technology, Iizuka, Japan. His research interests include logic optimization and multiple-valued logic. He has served on the editorial boards of the *IEEE Transactions on Computers*, *Computer*, and IEEE Computer Society Press. He has served as the editor-in-chief of *Computer* and IEEE Computer Society Press. He received the Award of Excellence, the Outstanding Contributed Paper Award, and a Distinctive Contributed Paper Award for papers presented at the International Symposium on Multiple-Valued Logic. He received the Distinguished Service Award, two Meritorious Awards, and nine Certificates of Appreciation for service to the IEEE Computer Society. He is a life fellow of the IEEE.