On Optimum Linear Decomposition of Symmetric 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 shows study results on linear decomposition of symmetric index generation functions. We analyze properties of symmetric index generation functions, and derive a theorem on the number of 1's in a compound variable for linear decomposition. The paper presents an algorithm using the theorem to exactly minimize the number of linear functions in linear decomposition. By taking advantage of the theorem and the symmetry properties, the algorithm can find an optimum linear decomposition quickly. Experimental results using symmetric index generation functions show the efficiency of the proposed algorithm.**

***Keywords*-Symmetric index generation functions; functional decomposition; linear decomposition; logic design; theoretical analysis.**

I. Introduction

Index searches are basic operations used in many applications, such as information lookup, computer virus detection, and analysis of DNA. The operations can be modeled as multiple-valued functions, called *index generation functions* [11], [12]. Index generation functions $f(x_1, x_2, \ldots, x_n)$ are efficiently realized by *linear decomposition* [2], [9], as shown in Fig. 1, where L realizes linear functions y_i $(i = 1, 2, \ldots, p)$, and G realizes a function storing indices of an index generation function. The first part L produces y_i from inputs $x_1, x_2, \ldots, x_n$ of f, and the second part G generates an index of f from y_i.

A memory-based architecture [11] for linear decomposition of index generation functions has been proposed, and shows much promise as a fast programmable circuit for the above applications. 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. As the number of linear functions p increases, the size of G exponentially increases. Thus, minimization of p is required in the architecture, and many minimization methods [1], [4]–[8], [13], [14], [16]–[20] have been proposed thus far. Among them, we focus on exact minimization methods [5]–[8], [18], [19] because of academic interest as well as practical requirements.

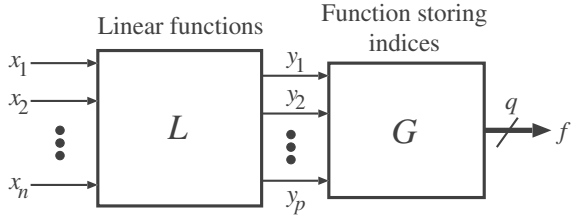


Figure 1. Linear decomposition of index generation functions [14].

Most of the existing exact minimization methods have no specific target class of functions. However, by targeting only *symmetric* index generation functions, we can find an optimum linear decomposition more efficiently [18]. A dedicated optimization algorithm for symmetric index generation functions has been proposed in [7], and it is shown that utilization of a symmetric property is effective to reduce the solution search space. However, there is room for improvement in the algorithm, since symmetric properties have not been fully analyzed.

Thus, this paper begins with analyzing properties of symmetric index generation functions, and presents an optimization algorithm using the properties for linear decomposition of the functions. Since the presented algorithm is based on dynamic programming [8] and uses zero-suppressed binary decision diagrams (ZDDs) as well, the solution search space is significantly reduced, resulting in fast solution search.

The rest of this paper is organized as follows: Section II defines symmetric index generation functions, linear decomposition, and ZDDs. In Section III, we analyze properties of symmetric index generation functions, and derive a theorem on the number of 1's in a compound variable for linear decomposition. Section IV formulates the minimization problem for the number of linear functions, and presents a dynamic programming based algorithm using the symmetric properties and ZDDs. Section V shows experimental results using some symmetric index generation functions, and Section VI concludes the paper.

II. Preliminaries

A. Index Generation Functions and Linear Decomposition

This subsection shows definitions of symmetric index generation functions [7], [11], [12], [18] and linear decompositions [2], [9], [14].

Definition 1: An **incompletely specified index generation function**, or simply **index generation function**, $f(X)$ is a multiple-valued function, where X is a set of n binary

978-1-7281-5406-0/20/$31.00 ©2020 IEEE
DOI 10.1109/ISMVL49045.2020.00-17

Table I
EXAMPLE OF SYMMETRIC INDEX GENERATION FUNCTION.

Registered vectors				indices of
x_1	x_2	x_3	x_4	S_1^4
0	0	0	1	1
0	0	1	0	2
0	1	0	0	3
1	0	0	0	4

Table II
FUNCTION g STORING INDICES IN LINEAR DECOMPOSITION OF S_1^4.

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

variables $(x_1, x_2, \ldots, x_n)$, and k is the number of assignments of values to the binary variables, where each assignment maps to a member of $K = \{1, 2, \ldots, k\}$. 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 $x_1, x_2, \ldots, x_n$ and K. Other assignments are left unspecified. The k assignments of values to $x_1, x_2, \ldots, x_n$ are called the **registered vectors**. K is called the set of **indices**. $k = |K|$ is called the **weight** of the index generation function f.

Definition 2: A **characteristic function** χ of an index generation function $f(X)$ is a logic function: $\{0,1\}^n \rightarrow \{0,1\}$ defined as

$$\chi(X) = \begin{cases} 1 & (f(X) \in K) \\ 0 & (Otherwise). \end{cases}$$

Definition 3: A **symmetric logic function** χ satisfies

$$\chi(x_1, x_2, \ldots x_i, \ldots, x_j, \ldots, x_n) = \chi(x_1, x_2, \ldots x_j, \ldots, x_i, \ldots, x_n)$$

for $\forall x_i, x_j \in X$. In this function, function values are decided only by the number of 1's in an assignment of values to $x_1, x_2, \ldots, x_n$. An **elementary symmetric function** S_m^n is a special case of symmetric logic functions where $S_m^n = 1$ if and only if the number of 1's in an assignment to $x_1, x_2, \ldots, x_n$ is m [10].

Definition 4: Let $\chi(x_1, x_2, \ldots, x_n)$ be a characteristic function of an index generation function f. When χ is symmetric, f is a **symmetric index generation function**.

Note that "f is a symmetric index generation function" does not mean f is a symmetric function (where any permutation of the input values leaves the function value unchanged).

For simplicity, we consider only the case where χ is an elementary symmetric function. Thus, in this paper, *symmetric index generation function* S_m^n means that its χ is an elementary symmetric function S_m^n. A symmetric index generation function S_m^n has

$$k = \binom{n}{m}$$

registered vectors (i.e., indices).

Example 1: Table I shows an example of 4-variable symmetric index generation function S_1^4 with weight four. Note that, in this function, input values other than 0001, 0010, 0100, and 1000 are NOT assigned to any function values. □

Definition 5: 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.

When all the subsets I_i are singletons (i.e., $|I_i| = 1$), $|\mathcal{P}| = |K| = k$. This is a trivial partition. Another trivial partition is when $I_1 = K$, in which case $|\mathcal{P}| = 1$.

Definition 6: **Linear decomposition** of an index generation function $f(x_1, x_2, \ldots, x_n)$ realizes f using a function $g(y_1, y_2, \ldots, y_p)$ storing indices and linear functions y_i:

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

where $i \in \{1, 2, \ldots, p\}$, $a_{ij} \in \{0,1\}$ $(j \in \{1, 2, \ldots, n\})$, 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 $\sum$ is an integer sum.

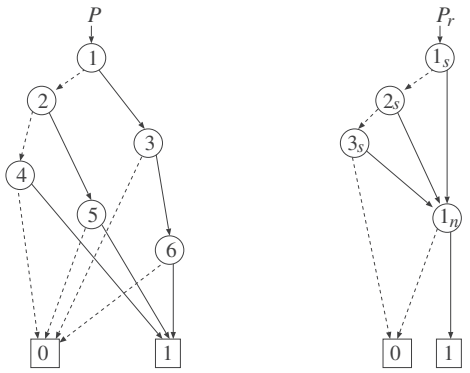
Definition 7: An inverse function of an index storing function $z = g(y_1, y_2, \ldots, y_p)$ in a linear decomposition is a mapping from $K = \{1, 2, \ldots, k\}$ to a set of p-bit vectors $\{0,1\}^p$, denoted by $g^{-1}(z)$. In this inverse function $g^{-1}(z)$, a mapping obtained by focusing only on the i-th bit of the p-bit vectors: $K \rightarrow \{0,1\}$ is called an **inverse function to a compound variable** y_i, denoted by $(g^{-1})_i(z)$.

Definition 8: 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 S_1^4 in Example 1 can be decomposed into two linear functions: $y_1 = x_1 \oplus x_2$ and $y_2 = x_1 \oplus x_3$, and a function $g(y_1, y_2)$ shown in Table II. In this case, $deg(y_1) = deg(y_2) = 2$, and S_1^4 can be realized by the architecture in Fig. 1 with a $(2^2 \times 3)$-bit memory for G, while a $(2^4 \times 3)$-bit memory is needed to directly realize S_1^4 without linear decomposition.

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. □

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



(a) For $\mathcal{P}$ [6] (b) For $\mathcal{P}_r$ [7]

Figure 2. ZDDs for $\mathcal{P} = \{\{1,3,6\},\{2,5\},\{4\}\}$ and its $\mathcal{P}_r$.

B. Zero-suppressed Binary Decision Diagrams (ZDDs)

In this subsection, we briefly define ZDDs [3].

Definition 9: A **zero-suppressed binary decision diagram (ZDD)** is a rooted directed acyclic graph (DAG) representing a logic function. It consists of two terminal nodes representing function values 0 and 1, and nonterminal nodes representing input variables. Each nonterminal node has two outgoing edges, a 0-edge and a 1-edge, that correspond to the values of an input variable. Neither terminal node has outgoing edges.

A ZDD is obtained by repeatedly applying the Shannon expansion $f = \overline{x_i} f_0 \vee x_i f_1$ to a logic function, where $f_0 = f(0 \rightarrow x_i)$, and $f_1 = f(1 \rightarrow x_i)$, and by applying the following two reduction rules:

1) Coalesce equivalent sub-graphs.
2) Delete nonterminal nodes v whose 1-edge points to the terminal node representing 0, and redirect edges pointing to v to its child node u that is pointed by the 0-edge of v.

As is well known, ZDDs can represent sets compactly and uniquely [3]. Similarly, a partition of an index set $\mathcal{P} = \{I_1, I_2, \ldots, I_u\}$ can be also represented compactly and uniquely using a ZDD. In a partition of indices $\mathcal{P}$, we define a *relation* $\mathcal{P}_r$ between size $|I_i|$ of $I_i \in \mathcal{P}$ and the number of I_i's with the same size. Then, the relation $\mathcal{P}_r$ can be represented compactly by a ZDD since $\mathcal{P}_r$ is a set of pairs of $|I_i|$ and the number of I_i's.

Example 3: Let an index set be $K = \{1,2,3,4,5,6\}$, and a partition of K be $\mathcal{P} = \{\{1,3,6\},\{2,5\},\{4\}\}$. In this partition, size of each subset is 3, 2, and 1, respectively. Thus, we have a relation $\mathcal{P}_r = \{(3,1),(2,1),(1,1)\}$. In another partition $\mathcal{P}' = \{\{1,2\},\{3,4\},\{5,6\}\}$, we have a relation $\mathcal{P}'_r = \{(2,3)\}$ since size of all the three subsets is 2. □

Example 4: Fig. 2(a) shows a ZDD *directly* representing $\mathcal{P} = \{\{1,3,6\},\{2,5\},\{4\}\}$, and Fig. 2(b) shows a ZDD representing the relation $\mathcal{P}_r = \{(3,1),(2,1),(1,1)\}$. In Fig. 2, dashed lines and solid lines denote 0-edges and 1-edges, respectively. In Fig. 2(b), nodes $1_s, 2_s, 3_s$ represent subset sizes in $\mathcal{P}$, and the node 1_n denotes that the number of subsets with the same size is 1. Since the size of each subset is different than others, all the pairs in $\mathcal{P}_r$ share the nonterminal node of 1_n in the ZDD. □

Theorem 1: [7] Let an index set be $K = \{1,2,\ldots,k\}$, a partition of K be $\mathcal{P}$, and a relation between subset size and the number of subsets in $\mathcal{P}$ be $\mathcal{P}_r$. An upper bound on the number of nonterminal nodes in a ZDD for $\mathcal{P}_r$ is $\sqrt{8k+1} - 1$.

III. Analysis of Symmetric Properties

In a symmetric index generation functions S^n_m, any variable x_i has the same number of 1's, and the number is

$$|ON(x_i)| = \binom{n-1}{m-1},$$

since this is the number of registered vectors where $x_i = 1$. In addition, for any combination of t variables,

$$|ON(x_{i_1}) \cap ON(x_{i_2}) \cap \ldots \cap ON(x_{i_t})| = |ON(x_{j_1}) \cap ON(x_{j_2}) \cap \ldots \cap ON(x_{j_t})|$$

holds. That is, any combination has the same number of overlapping 1's in its variables. Thus, we can compute the number of 1's in a compound variable y_i from only n, m, and its compound degree t, without using registered vectors.

Lemma 1: For any combination of t variables $x_{i_1}, x_{i_2}, \ldots, x_{i_t}$ in a symmetric index generation function S^n_m,

$$|ON(x_{i_1}) \cap ON(x_{i_2}) \cap \ldots \cap ON(x_{i_t})| = \binom{n-t}{m-t}$$

holds, where $t \leq m$.

Proof: The left side of the equation denotes the number of registered vectors such that $x_{i_1} = x_{i_2} = \ldots = x_{i_t} = 1$. Since the number of 1's in each registered vector of S^n_m is exactly m, the set

$$ON(x_{i_1}) \cap ON(x_{i_2}) \cap \ldots \cap ON(x_{i_t})$$

includes registered vectors, in which the remaining $m-t$ 1's are arbitrarily assigned to variables other than $x_{i_1}, x_{i_2}, \ldots,$ and x_{i_t}. Thus, the number of combinations choosing $m-t$ variables from $n-t$ variables is the number of overlapping 1's in all the t variables. ∎

Theorem 2: For a symmetric index generation function S^n_m, the number of 1s n_1 in any compound variable y_i with compound degree t is

$$|ON(y_i)| = n_1(n,m,t) = \sum_{j=1}^{d} (-2)^{j-1} \binom{t}{j} \binom{n-j}{m-j},$$

where $d = \min(t,m)$.

Proof: From Definitions 6 and 8, we have

$$|ON(y_i)| = |ON(x_{i_1}) \oplus ON(x_{i_2}) \oplus \ldots \oplus ON(x_{i_t})|, \qquad (1)$$

where $\oplus$ is the symmetric difference of sets:

$$A \oplus B = A \cup B - A \cap B.$$

The value of (1) can be computed by extending *the principle of inclusion and exclusion* for union of sets. From Lemma A.1 in Appendix A, the value of (1) is

$$\left| \bigoplus_{j=1}^{t} ON(x_{i_j}) \right| = \sum_{j=1}^{t} (-2)^{j-1} \sum_{H_j \in \{2^T - \emptyset\}} \left| \bigcap_{h \in H_j} ON(x_{i_h}) \right|.$$

In addition, from Lemma 1,

$$\left| \bigcap_{h \in H_j} ON(x_{i_h}) \right| = \binom{n-j}{m-j}.$$

Since the number of different H_j's is $\binom{t}{j}$, we have the theorem. When $m < t$, it is clear from Definition 3 that for any $j > m$,

$$\bigcap_{h \in H_j} ON(x_{i_h}) = \emptyset.$$

Therefore, the summation is terminated by $d = \min(t, m)$. ∎

In addition to the theorem on the number of 1s in a compound variable, we derive another useful theorem.

Theorem 3: For a symmetric index generation function S_1^n, let H_t be a subset of its index set $K = \{1, 2, \ldots, k\}$ including exactly t indices (i.e., $|H_t| = t$). Then, for any H_t, there exists a compound variable y_i with compound degree t such that $ON(y_i) = H_t$.

Proof: From Definition 3, for S_1^n, it is clear that

$$ON(x_i) \cap ON(x_j) = \emptyset \qquad \text{for any } i, j \ (i \neq j),$$

and from Theorem 2,

$$|ON(y_i)| = t.$$

Since each x_i corresponds to exactly one index, we can produce a compound variable y_i for H_t as

$$y_i = x_{i_1} \oplus x_{i_2} \oplus \ldots \oplus x_{i_t},$$

where $ON(x_{i_j}) \in H_t$. ∎

Note that Theorem 3 does not hold for S_m^n with $m \geq 2$. We intend to consider this in a future paper.

IV. Optimization of Linear Decompositions

We formulate the optimization problem on linear decomposition of symmetric index generation functions, and show a dynamic programming based algorithm using the above theorems.

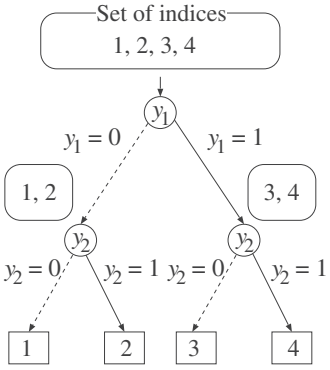


Figure 3. Binary decision tree for g of Table II.

A. Formulation of Optimization Problem

The optimum linear decomposition problem of symmetric index generation functions is formulated as follows:

Problem 1: Given a symmetric index generation function S_m^n and the maximum compound degree t, find the minimum number of linear functions for linear decomposition of S_m^n where no compound degree exceeds t.

Problem 1 can be reduced to a tree height minimization problem of an ordered binary decision tree that recursively divides subsets of indices into smaller subsets until all subsets are singletons [4].

Example 5: Fig. 3 shows an ordered binary decision tree representing g in Table II. The tree divides the set of indices into singletons by compound variables y_1 and y_2. The tree height corresponds to the number of compound variables. □

B. Dynamic Programming Based Algorithm

In symmetric index generation functions, any permutation of indices does not change the minimum number of linear functions in their linear decomposition. This is because exchange of indices does not change the characteristic function of a symmetric index generation function. Thus, instead of directly representing partitions $\mathcal{P}$ of indices, we can abstract them as $\mathcal{P}_r$ introduced in Section II. Based on this observation, the existing method [7] uses ZDDs for $\mathcal{P}_r$, as shown in Fig. 2(b), to represent partitions of indices more efficiently than direct representation of $\mathcal{P}$ shown in Fig. 2(a).

We improve the existing method using the theorems in Section III. Using Theorem 2, we can compute the optimum number of 1's in a compound variable in advance without compounding original variables x_i. The optimum number is the closest number to $k/2$ among ones obtained by compounding up to t variables. In addition, from Theorem 3, we can consider only the ways to divide a partition of indices $\mathcal{P}$ based on the optimum number of 1's, rather than combinations of compound variables. Since the number of 1's can be considered as the number of indices separated from subsets in $\mathcal{P}$, we can concentrate only on how many indices

Algorithm 1: Overview of the proposed algorithm

Input: a symmetric index generation function S_1^n and the maximum compound degree t
Output: the smallest number of compound variables h_{min}

```
min_DPsearch(S_1^n, t) {
  n1_opt = compute_opt_numof1s(S_1^n, t);
  Let an initial set of partitions be C = {P_r},
   where P_r = {(n,1)};
  for (h = 1; {(1,n)} ∉ C; h++) {
    for (each P_r ∈ C: current set of partitions) {
      for (each different partition P'_r generated by
            dividing P_r with n1_opt) {
        N = store_as_next(P'_r, h);
      }
    }
    Update the set of partitions C with N;
  }
  return h − 1 as the solution;
}
```

should be separated from each subset for optimization, and avoid time consuming searches of compound variables.

As the first step toward optimization of any S_m^n by this approach, we present a dynamic programming based algorithm for S_1^n. Algorithm 1 shows the overview of the proposed dynamic programming based algorithm. In the algorithm, C denotes a set of $\mathcal{P}_r$, and its element $\mathcal{P}_r$ is represented by a ZDD. By updating C while generating a *different partition* from $\mathcal{P}_r$ in turn, the algorithm searches for a solution. Different partitions $\mathcal{P}'_r$ are generated by separating indices from subsets of indices S_i in $\mathcal{P}_r$, where the number of indices separated from a subset I_i is at most

$$\left\lfloor \frac{|I_i|}{2} \right\rfloor,$$

and the total number of separated indices is $n1_{opt}$. Subsets in $\mathcal{P}_r$ are extracted from a ZDD for $\mathcal{P}_r$ by traversing all the 1-paths in the ZDD. After separating $n1_{opt}$ indices from subsets, another ZDD for $\mathcal{P}'$ are constructed using *Change* and *Union* operations that are basic operations in a ZDD package [3].

A generated partition $\mathcal{P}'_r$ is screened by the procedure *store_as_next*() to store it as a set of promising partitions for the next stage of search. Partitions $\mathcal{P}'_r$ are discarded if one of the following two conditions holds:

1) The same $\mathcal{P}'_r$ has already stored.
2) Sum of h and the *lower bound* on the number of compound variables needed to divide $\mathcal{P}'$ into singletons is not smaller than the *upper bound* on the minimum solution.

1) Although only different partitions are generated from a $\mathcal{P}_r$, the same partitions can be generated from other partitions than $\mathcal{P}_r$ in C. Since partitions are represented by ZDDs, this checking (equivalence checking) is performed on ZDDs in $O(1)$ time. 2) To discard unpromising partitions as many as possible, we use the *lower bound* derived in [5], and a result

Algorithm 2: Overview of the greedy algorithm

Input: a partition of indices $\mathcal{P}_r$ and a tree height h
Output: an estimated tree height h_{heu}

```
greedy_search(P_r, h) {
  P = turn_back(P_r);
  for (l = 1; |P| < n; l++) {
    N1 = 0;
    for (i = 1; i < n1_opt; i++) {
      Select a subset I ∈ P making the cost function
        minimum by separating an index from I;
      n1_I++;
    }
    P = divide_sets(P, N1);
  }
  return h + l − 1;
}
```

Table III
COMPUTATION TIME OF METHODS (IN SECONDS).

Symmetric functions	t	h_{heu}	h_{min}	Existing [8]	Existing [7]	Proposed
S_1^{10} ($k=10$)	1	9	9	*<0.01	*<0.01	*<0.01
	2	7	6	0.10	*<0.01	*<0.01
	3	5	5	0.18	*<0.01	*<0.01
	4	4	4	*<0.00	*<0.01	*<0.01
	5	4	4	*<0.01	*<0.01	*<0.01
S_1^{20} ($k=20$)	1	19	19	*<0.01	*<0.01	*<0.01
	2	14	13	†	0.03	*<0.01
	3	10	10	†	0.53	*<0.01
	4	8	8	†	1.92	*<0.01
	5	7	7	†	2.78	*<0.01
S_1^{30} ($k=30$)	1	29	29	*<0.01	*<0.01	*<0.01
	2	22	20	†	0.26	*<0.01
	3	16	15	†	9.19	*<0.01
	4	12	12	†	127.85	0.01
	5	10	10	†	769.06	0.01

* Time is less than 0.01 sec..
† Computation was aborted since it was too long.

of the heuristic method shown in Algorithm 2 as the *upper bound*. The cost function used in Algorithm 2 is

$$\mathrm{cost}(\mathcal{P},N1) = \sqrt{\sum_{I\in\mathcal{P}} \left(\frac{|I|}{2} - n1_I\right)^2},$$

where $\mathcal{P}$ is a partition turned back from $\mathcal{P}_r$, and $N1$ is a vector of numbers of indices, $n1_I$, separated from each I.

V. EXPERIMENTAL RESULTS

The proposed exact minimization algorithm 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 Linux, and C-compiler: gcc -O2 (version 4.1.2).

To evaluate the efficiency of the proposed algorithm, we compare the proposed algorithm with the existing methods [7], [8], in terms of computation time. The method in [7] is based on a branch-and-bound approach targeting only symmetric index generation functions. On the other hand, the method in [8] is based on a dynamic programming approach targeting general index generation functions. Table III shows computation time, in seconds, of each method for some symmetric index generation functions. The column labeled

Table IV
COMPUTATION TIME FOR LARGER SYMMETRIC FUNCTIONS.

Symmetric functions	t	h_{heu}	h_{min}	No. of ZDDs	No. of nodes	Time (sec.)
S_1^{40} ($k=40$)	1	39	39	1	4	*<0.01
	2	29	26	327	698	*<0.01
	3	21	20	1,116	2,341	0.01
	4	17	16	3,508	7,136	0.07
	5	14	13	4,562	9,903	0.18
S_1^{50} ($k=50$)	1	49	49	1	4	*<0.01
	2	37	33	542	1,141	0.01
	3	27	25	2,345	4,826	0.03
	4	21	20	9,094	18,125	0.24
	5	17	17	18,533	37,388	1.01
S_1^{60} ($k=60$)	1	59	59	1	4	*<0.01
	2	44	40	807	1,681	0.01
	3	32	30	4,240	8,617	0.06
	4	25	24	19,297	38,171	0.63
	5	21	20	46,374	92,891	3.50
S_1^{70} ($k=70$)	1	69	69	1	4	*<0.01
	2	52	46	1,107	2,293	0.02
	3	38	35	6,922	13,990	0.13
	4	30	28	36,117	71,298	1.39
	5	24	23	99,803	199,422	9.67
S_1^{80} ($k=80$)	1	79	79	1	4	*<0.01
	2	59	53	1,472	3,036	0.02
	3	44	40	10,511	21,181	0.22
	4	34	32	61,913	122,152	2.72
	5	28	27	194,809	387,639	1731.25

* Time is less than 0.01 sec..

"h_{heu}" in Table III shows results obtained by only the proposed heuristic method in Algorithm 2.

As shown in Table III, the computation time of the proposed method is a few orders of magnitude shorter than computation time of the existing ones. The functions in Table III are too small for the proposed method, and thus, we could not obtain its computation time precisely for almost all functions.

Then, we applied the proposed method to larger index generation functions. Table IV shows computation time of the proposed method for larger symmetric index generation functions. In this table, the column "No. of ZDDs" shows the number of ZDDs constructed during solution search. And, the column "No. of nodes" shows the total number of nodes in ZDDs. From Table IV, we can see that the number of ZDDs is not large. This is because unpromising solutions are effectively reduced, and equivalent solutions are efficiently merged using ZDDs. Therefore, computation time of the proposed method is still short even for large functions for which the existing methods cannot find a solution in a reasonable time.

VI. CONCLUSION AND COMMENTS

This paper shows theorems for symmetric index generation functions, and presents an exact optimization algorithm for their linear decomposition. By taking advantage of the theorems, we can avoid time consuming searches of compound variables, and find an optimum linear decomposition quickly. Experimental results show that the proposed method finds optimum solutions much faster than the existing ones.

Our future work includes analysis of symmetric properties for S_m^n with $m \geq 2$, and proposal of an exact optimization algorithm using these properties. In addition, we will generalize theorems and algorithms for elementary symmetric index generation functions to general ones toward fully theoretical analysis of optimum linear decomposition of symmetric index generation functions.

ACKNOWLEDGMENTS

This research is partly supported by the JSPS KAKENHI Grant (C), No.19K11881, 2020. We would like to thank Prof. Michael Miller for motivating us to use symmetric properties. The reviewers' comments were helpful in improving the paper.

REFERENCES

[1] 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.

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

[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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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).

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

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

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

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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).

[20] 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.

APPENDIX

A. Extending Principle of Inclusion and Exclusion

From the principle of inclusion and exclusion for union of t sets, we have

$$\left|\bigcup_{i=1}^{t} A_i\right| = \sum_{i=1}^{t} (-1)^{i-1} \sum_{J_i \in \{2^T - \emptyset\}} \left|\bigcap_{j \in J_i} A_j\right|,$$

where $T = \{1,2,\ldots,t\}$, 2^T is the power set of T, and J_i is a subset of T including i elements. By extending this principle for symmetric difference of t sets, we have the following:

Lemma A.1: For symmetric difference of t sets, the following holds:

$$\left|\bigoplus_{i=1}^{t} A_i\right| = \sum_{i=1}^{t} (-2)^{i-1} \sum_{J_i \in \{2^T - \emptyset\}} \left|\bigcap_{j \in J_i} A_j\right|, \qquad \text{(A.1)}$$

where $T = \{1,2,\ldots,t\}$, 2^T is the power set of T, and J_i is a subset of T including i elements.

Proof: Unlike union of sets, in symmetric difference, elements belonging to an even number of subsets must be completely eliminated. Thus, we prove that the total sum of coefficients in (A.1) is 0 when t is even, and on the other hand, it is 1 when t is odd. The total sum of coefficients is obtained by the following:

$$\sum_{i=1}^{t} 2^{i-1} \binom{t}{i},$$

since the number of different J_i's is $\binom{t}{i}$.

From the binomial theorem, we have

$$\begin{aligned} (1-2)^t &= \sum_{i=0}^{t} \binom{t}{i} (-2)^i \\ (-1)^t &= 1 + \sum_{i=1}^{t} \binom{t}{i} (-2)^i \\ \frac{1-(-1)^t}{2} &= \sum_{i=1}^{t} \binom{t}{i} (-2)^{i-1}. \end{aligned}$$

When t is even,

$$0 = \sum_{i=1}^{t} \binom{t}{i} (-2)^{i-1}.$$

When t is odd,

$$1 = \sum_{i=1}^{t} \binom{t}{i} (-2)^{i-1}.$$

Therefore, we have the lemma. ∎