IEICE TRANS. INF. & SYST., VOL.E107–D, NO.8 AUGUST 2024

PAPER Special Section on Multiple-Valued Logic and VLSI Computing

Functional Decomposition of Symmetric Multiple-Valued Functions and Their Compact Representation in Decision Diagrams*

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

SUMMARY This paper proposes a decomposition method for symmetric multiple-valued functions. It decomposes a given symmetric multiple-valued function into three parts. By using suitable decision diagrams for the three parts, we can represent symmetric multiple-valued functions compactly. By deriving theorems on sizes of the decision diagrams, this paper shows that space complexity of the proposed representation is low. This paper also presents algorithms to construct the decision diagrams for symmetric multiple-valued functions with low time complexity. Experimental results show that the proposed method represents randomly generated symmetric multiple-valued functions more compactly than the conventional representation method using standard multiple-valued decision diagrams. Symmetric multiple-valued functions are a basic class of functions, and thus, their compact representation benefits many applications where they appear.

key words: *symmetric functions, multiple-valued functions, functional decomposition, decision diagrams*

1. Introduction

Symmetric functions are functions whose values are unchanged by any permutation of input variable labels. Often studies in digital system design involve symmetric functions. For example, studies of arithmetic operations, cryptography, and voting systems in ensemble machine learning [7], [12], [13], [20], [25] often use symmetric functions. Whole sections of textbooks deal with symmetric functions [10], [22]. More than 50 years ago, it also was shown that symmetric functions can represent any switching functions using repetition of input variables [2], [8], [11], [29]. Also, *multiple-valued* symmetric functions have been widely studied [5], [19], [26]. A compact representation, such as the one presented here, can result in a breakthrough in multiple-valued symmetric functions.

To represent symmetric functions compactly, this paper focuses on their functional decomposition. Decision diagrams [1], [3], [9] for symmetric functions are not so large even in a monolithic (undecomposed) decision diagram because of their regular structure. However, we aim for a more compact representation of symmetric functions by decomposing the functions and using decision diagrams for the decomposed parts. The problem to achieve such a compact representation is how to decompose symmetric functions and what decision diagrams are used.

This paper proposes a method to decompose symmetric multiple-valued functions based on equivalence classes of input vectors. By using an index generation function [23], [24], a symmetric multiple-valued function is decomposed into three parts. For each of the three parts, we propose suitable decision diagrams to obtain a compact representation of symmetric multiple-valued functions. This paper derives the exact number of nodes in the decision diagrams for symmetric multiple-valued functions to show the compactness of the proposed representation method theoretically. This paper also proposes algorithms to construct the decision diagrams for symmetric multiple-valued functions with low time complexity. Experimental results using randomly generated symmetric multiple-valued functions show that the size of the proposed decision diagrams is smaller than the size of monolithic multiple-valued decision diagrams (MDDs) that are conventionally used for representation of symmetric multiple-valued functions.

The rest of this paper is organized as follows: Section 2 shows some definitions for symmetric multiple-valued functions and conventional decision diagrams. Section 3 presents a decomposition method of symmetric multiple-valued functions. Based on the decomposition method, Sect. 4 presents a compact representation method using decision diagrams. Section 4 also shows some theorems on the size of the decision diagrams and algorithms to construct them. Section 5 compares the size of the proposed decision diagrams with the size of monolithic MDDs for randomly generated symmetric multiple-valued functions, and Sect. 6 concludes the paper.

2. On Symmetric Functions and Decision Diagrams

In this section, we briefly define symmetric multiple-valued functions [5] and basic decision diagrams.

2.1 Symmetric Multiple-Valued Functions

Definition 1: For an n-variable r-valued function $f(X_1,$

Manuscript received October 27, 2023.
Manuscript revised March 8, 2024.
Manuscript publicized May 14, 2024.

[†]Dept. of Computer and Network Eng., Hiroshima City Univ., Hiroshima-shi, 731–3194 Japan.
[††]Dept. of Computer Science, Meiji Univ., Kawasaki-shi, 214–8571 Japan.
[†††]Dept. of Electrical and Computer Eng., Naval Postgraduate School, Monterey, CA 93943-5121 USA.

*This paper is an extension of [18] (new theorems, a new decision diagram, and new algorithms to generate decision diagrams for symmetric functions are mainly added).

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

DOI: 10.1587/transinf.2023LOP0010

Table 1 Example of a three-valued symmetric function.

X_1	X_2	X_3	f
0	0	0	0
1	1	1	2
2	2	2	1
0	0	1	2
0	1	0	2
1	0	0	2
0	0	2	0
0	2	0	0
2	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1
0	2	2	1
2	0	2	1
2	2	0	1
1	1	2	0
1	2	1	0
2	1	1	0
1	2	2	2
2	1	2	2
2	2	1	2
0	1	2	2
0	2	1	2
1	0	2	2
1	2	0	2
2	0	1	2
2	1	0	2

$X_2, \ldots, X_n) : \{0, 1, \ldots, r-1\}^n \rightarrow \{0, 1, \ldots, r-1\}$, an assignment of values to the n variables is an **input vector** $\vec{X}$. When the r^n input vectors $(0, 0, \ldots, 0) \sim (r-1, r-1, \ldots, r-1)$ are applied to the input variables of the function f in ascending order, the vector of obtained function values is the **function vector** $\vec{F}$. In this paper, we assume that all the r^n values of f are specified (i.e., f is a completely specified function) unless otherwise stated.

Definition 2: An n-variable function f ($n \geq 2$) is **symmetric**[†] if its function vector $\vec{F}$ is unchanged by any permutation of any variable labels. That is, in this paper, a symmetric function means a totally symmetric function.

Example 1: Table 1 shows an example of a three-variable three-valued symmetric function f. In this table, input vectors are reordered and grouped into the same combinations of input values. As shown in Table 1, function values of the function are independent of permutations of input values, but are dependent only on *combinations of input values*. □

2.2 Basic Decision Diagrams

Definition 3: A **multiple-valued decision diagram (MDD)** [9] is a rooted directed acyclic graph (DAG) representing an r-valued function. The MDD is obtained by recursively applying the extended Shannon expansion to the r-valued function. It consists of r terminal nodes representing function values, 0 to $r-1$, and nonterminal nodes representing input r-valued variables. Each nonterminal node has r outgoing edges that correspond to r values of an input variable. Terminal nodes have no outgoing edges. In this paper, an MDD is obtained by fixing the variable order in an MDD, and by applying the following two reduction rules:

1. Coalesce equivalent sub-graphs.
2. Delete nonterminal nodes v all of whose outgoing edges point to the same node, and redirect edges pointing to v to its child node u.

[†]This paper focuses only on variable-symmetry [5].

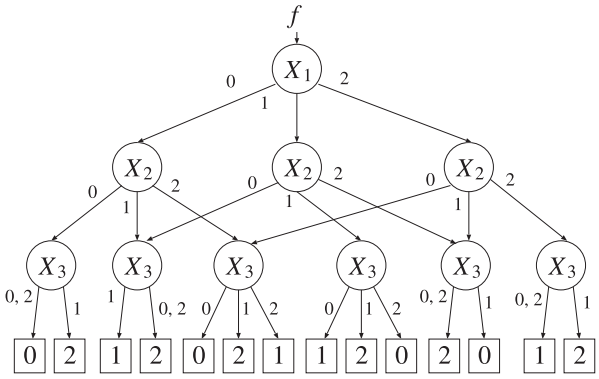

Fig. 1 MDD for symmetric function f specified in Table 1.

Example 2: Figure 1 shows an MDD for the symmetric multiple-valued function f in Table 1. In Fig. 1, for readability, terminal nodes in the MDD are NOT shared completely. The number of nodes in this MDD is 13 (3 distinct terminal nodes and 10 distinct non-terminal nodes). □

Definition 4: An **edge-valued MDD (EVMDD)** [15], [16] is a variant of an MDD. It consists of one terminal node representing 0 and nonterminal nodes with edges having integer weights; 0-edges always have zero weights. In an EVMDD, the function value is represented as a sum of weights for edges traversed from the root node to the terminal node.

Definition 5: A **zero-suppressed binary decision diagram (ZDD)** [14] is a variant of a binary decision diagram (BDD) [1], [3] that is a special case of an MDD for representing a binary function. It consists of two terminal nodes representing function values 0 and 1 respectively, and nonterminal nodes representing input binary variables. Each nonterminal node has two unweighted outgoing edges, 0-edge and 1-edge, that correspond to two values of an input variable. Both terminal nodes have no outgoing edges. In this paper, a ZDD is obtained by fixing the variable order in a ZDD, 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 pointed by v's 0-edge.

3. Functional Decomposition of Symmetric Functions

For symmetric functions, if *combinations of input values* are

Table 2 Correspondences between $\vec{\alpha}$'s and function values of f.

α_2	α_1	α_0	f
0	0	3	0
0	3	0	2
3	0	0	1
0	1	2	2
1	0	2	0
0	2	1	1
2	0	1	1
1	2	0	0
2	1	0	2
1	1	1	2

Table 3 Index generation function i_{dx} and h' in decomposition of h.

α_2	α_1	α_0	i_{dx}
0	0	3	index_0
0	3	0	index_1
3	0	0	index_2
0	1	2	index_3
1	0	2	index_4
0	2	1	index_5
2	0	1	index_6
1	2	0	index_7
2	1	0	index_8
1	1	1	index_9

i_{dx}	h'
index_0	0
index_1	2
index_2	1
index_3	2
index_4	0
index_5	1
index_6	1
index_7	0
index_8	2
index_9	2

the same, they are assigned to the same function value, as shown in Example 1. Thus, it is unnecessary to distinguish each of input vectors having the same combination of input values to represent symmetric functions. We classify such input vectors into an equivalence class which has the same combination of input values. In this paper, by using the following notation introduced in [5], [6], [17], we represent equivalence classes of input vectors efficiently.

Definition 6: For an n-variable r-valued symmetric function $f(X_1, X_2, \ldots, X_n)$, we classify input vectors $\vec{X} = (X_1, X_2, \ldots, X_n)$ into equivalence classes, each of which has the same combination of input values. Then, we represent each equivalence class as follows:

$$\vec{\alpha} = (\alpha_{r-1}, \alpha_{r-2}, \ldots, \alpha_1, \alpha_0),$$

where α_i denotes the number of variables whose values are i, and $\sum_{i=0}^{r-1} \alpha_i = n$. In this paper, such equivalence classes are called **α-equivalence classes**.

Lemma 1: [6] For an n-variable r-valued symmetric function, the number of α-equivalence classes, N_α, is

$$N_\alpha = \binom{n+r-1}{r-1}.$$

Since $N_\alpha < r^n$, a table with $\vec{\alpha}$ and its corresponding function value is smaller than the truth table where the function value of every input vector is separately specified as Table 1.

Example 3: Table 2 shows correspondences between $\vec{\alpha}$'s and function values of the symmetric function f in Table 1. Since $N_\alpha = 10$ and $r^n = 27$, this table is much smaller than the truth table of f. □

In this way, by using $\vec{\alpha}$, we can decompose a symmetric multiple-valued function $f(\vec{X})$ into two functions g and h: the first function $g(\vec{X})$ that transforms $\vec{X}$ into $\vec{\alpha}$ and the second function $h(\vec{\alpha})$ that produces function values from $\vec{\alpha}$. That is, we have

$$f(\vec{X}) = h(g(\vec{X})).$$

Although the function g remains symmetric and completely specified, the function h is *asymmetric and incompletely specified*. In general, it is difficult to predict structures of decision diagrams for asymmetric and incompletely specified functions, and thus, whether they are compact or not. Thus, decision diagrams for h can be large even though a table for h is smaller than the truth table of f, as shown in Example 3.

To represent the function h with decision diagrams compactly, we decompose h into two functions: an *index generation function* i_{dx} [23], [24] and h'. The index generation function i_{dx} produces a unique index for each $\vec{\alpha}$, and the function h' produces a function value of f from an index. By using the three functions g, i_{dx}, and h', we decompose the original symmetric function f as follows:

$$f(\vec{X}) = h'(i_{dx}(g(\vec{X}))).$$

The function h' that is a map from the set of N_α indices to the set of r values is a *one-variable completely specified* function. On the other hand, the index generation function i_{dx} is usually *asymmetric and incompletely specified*. However, we can freely choose any values for indices as long as they are unique, since they do not affect the original function values. Thus, we choose values for indices of i_{dx} so that decision diagrams for i_{dx} and h are compact. In the next section, we discuss the values for indices making decision diagrams compact along with suitable decision diagrams for the three functions: g, h, and i_{dx}.

Example 4: Table 3 shows an index generation function i_{dx} and h' in decomposition of h for the symmetric function f. Since we can freely choose any values for 10 indices, each index is denoted abstractly as "index_i" in the tables. □

4. Representation of Decomposed Functions by Decision Diagrams

This section presents suitable decision diagrams for the three subfunctions: g, i_{dx}, and h', obtained by the proposed decomposition method in Sect. 3.

4.1 For Conversion of Input Vectors $\vec{X}$ into $\vec{\alpha}$

The first function g is a map from an input *vector* $\vec{X}$ to a *vector* $\vec{\alpha}$, and thus, it can be considered as a *multiple-output* multiple-valued function. To represent such a multiple-output function g, we introduce another decision diagram:

Definition 7: A **vectorized EVMDD (VEVMDD)** [28] is a variant of an EVMDD, and its edges have vectors instead

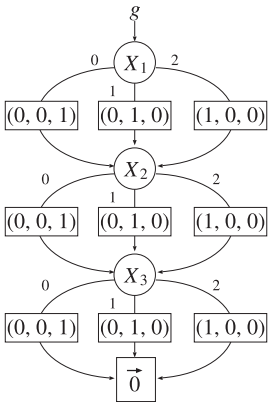

Fig. 2 VEVMDD for map g from three-variable three-valued $\vec{X}$ to $\vec{\alpha}$.

of scalar values. The vectors consist of integers, and 0-edges always have the zero vector. The terminal node also represents the zero vector. Output vectors of the function are represented as a sum of vectors of edges traversed from the root node to the terminal node.

Each element α_i of the $\vec{\alpha} = (\alpha_{r-1}, \alpha_{r-2}, \ldots, \alpha_i, \ldots, \alpha_1, \alpha_0)$ represents the number of input values i included in an input vector $\vec{X}$. Thus, we convert $\vec{X}$ into $\vec{\alpha}$ as follows:

1. Represent each input value i by one-hot encoding, in which only the $(i+1)$-th bit is 1 and the others are 0.
2. Vectorize the encoded value by considering each bit as each element of an r-element vector.
3. Compute a sum of one-hot encoded vectors according to values of input variables.

By assigning vectors obtained by the steps 1 and 2 to edges of a node for each input variable in a VEVMDD, we can represent g by a VEVMDD, and perform the computation of step 3 on the VEVMDD.

Example 5: Figure 2 shows a VEVMDD for the function g converting three-variable three-valued $\vec{X}$ into $\vec{\alpha}$. In this figure, each edge has a three-element vector denoted by a rectangle. Note that for ease of understanding, non-zero vectors are assigned to 0-edges. After normalizing the VEVMDD such that all 0-edges have only the zero vector, its edge to the root node has a vector $(0,0,3)$, 1-edges have $(0,1,-1)$, and 2-edges have $(1,0,-1)$.

Consider an input vector $\vec{X} = (2,0,2)$. The one-hot encoded vectors for values in $\vec{X}$ are $(1,0,0)$, $(0,0,1)$, and $(1,0,0)$, respectively. The sum of these three vectors is $(2,0,1)$, and it is equal to the $\vec{\alpha} = (\alpha_2, \alpha_1, \alpha_0)$ of $\vec{X}$. We can perform the same computation on the VEVMDD by traversing edges from the root node to the terminal node according to values of input variables, and summing up vectors of traversed edges. Note that $\vec{X} = (2,2,0)$ and $(0,2,2)$ also yield the same $\vec{\alpha} = (2,0,1)$. □

Since the function g is symmetric, and vectors for input variables can be chosen independently from each other, we have the following:

Theorem 1: A VEVMDD for an n-variable r-valued function that converts $\vec{X}$ into $\vec{\alpha}$ has exactly $n+1$ nodes, regardless of the order of variables.

(Proof) From the explanation just before the theorem, it is clear that the conversion from $\vec{X}$ into $\vec{\alpha}$ can be represented by a VEVMDD whose edges have the one-hot encoded vectors. Since each vector is chosen by an input variable independently, branching does not occur at each nonterminal node. Therefore, the number of nodes is always $n+1$ including the terminal node. Since the function is symmetric, the number of nodes is unchanged by any permutation of variables. The normalization of VEVMDDs affects only vectors of edges, and thus, the number of nodes is still unchanged. ∎

4.2 Function for Producing Original Function Values

Next, consider the function h' producing a function value from an index. As shown in Table 3, this function can be considered as a set of pairs of an index and a function value. Although it is well known that a ZDD is suitable for representation of such a set [14], we take a different approach shown in the following:

1. Represent indices by one-hot encoding.
2. Let the function from one-hot encoded indices to function values of f be h'_1. Then, h'_1 is an incompletely specified N_α-bit input r-valued output function.
3. Produce a completely specified function χ' from h'_1 by assigning a special value $\emptyset$ meaning invalid to *don't cares* of h'_1, where $\chi' : \{0,1\}^{N_\alpha} \to \{\emptyset, 0, 1, \ldots, r-1\}$.

To represent χ' by a compact decision diagram, we present the following variant of ZDD:

Definition 8: A **multiple-terminal ZDD (MTZDD)** is a variant of a ZDD for representing a binary input $(r+1)$-valued output function χ', and it has $r+1$ terminal nodes. One of the $r+1$ terminal nodes represents the invalid value $\emptyset$, and the others represent valid function values: $0, 1, \ldots, r-1$. In MTZDDs, the second reduction rule for ZDDs is modified as follows:

2. Delete nonterminal nodes v whose 1-edge points to the terminal node representing $\emptyset$, and redirect edges pointing to v to its child node u pointed by v's 0-edge.

Example 6: Figure 3 shows an MTZDD for the one-hot encoded function χ' of the function h' in Table 3. In Fig. 3, dashed lines and solid lines denote 0-edges and 1-edges, respectively. Note that each nonterminal node represents a binary variable obtained by the one-hot encoding of each index. For viewability, original indices are used as labels of the nonterminal nodes. The MTZDD has four terminal nodes for the invalid value $\emptyset$ as well as the three valid function values. The number of nodes in this MTZDD is 14. □

Theorem 2: For an r-valued function h' from N_α distinct indices to $\{0, 1, \ldots, r-1\}$, an MTZDD for its one-hot encoded function χ' has exactly N_α *nonterminal nodes*, regardless of the order of variables for indices.

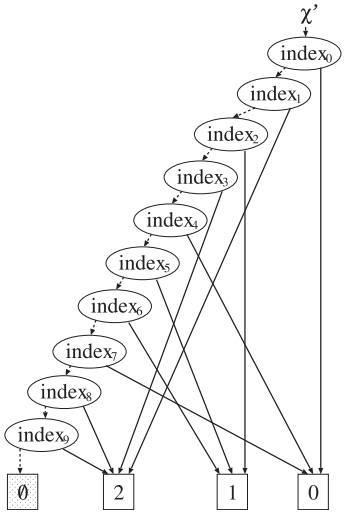

Fig. 3 MTZDD for one-hot encoded function χ' of h'.

(Proof) Let a *valid-path* in an MTZDD for χ' be a sequence of edges and nodes leading from the root node to a terminal node representing a valid function value. Since a valid-path represents a pair of an index and a function value, the number of distinct valid-paths is exactly N_α. That is, N_α distinct indices are represented by N_α distinct valid-paths, respectively. Thus, exactly N_α nonterminal nodes are needed to represent indices.

By considering the valid function values and the invalid value as logic values 1 and 0, respectively, the function χ' can be considered as a binary symmetric function $S^1_{N_\alpha}$ that its function value is 1 only when one of N_α input variables has 1 [22]. In ZDDs for binary symmetric functions, the number of nodes is unchanged by any permutation of variables. Since the only difference between ZDDs and MTZDDs is the values of terminal nodes, the number of nodes in MTZDDs for χ' is unchanged as well. Thus, we have the theorem. ∎

4.3 Index Generation Function i_{dx}

As shown in Sect. 4.2, the compactness of MTZDDs for χ' is due, in large part, to the use of an index generation function i_{dx}. However, if a decision diagram for i_{dx} is large, the advantage of the compactness can be canceled out. Fortunately, values of indices are still independent of the complexity of MTZDDs, and thus, we can freely decide index values so that a decision diagram for i_{dx} will be compact. The only constraint on index values is that they must be unique.

The simplest way to produce unique indices from $\vec{\alpha}$ is considering an $\vec{\alpha} = (\alpha_{r-1}, \alpha_{r-2}, \ldots, \alpha_0)$ as an r-digit *base* $(n+1)$ *number* $(\alpha_{r-1}\alpha_{r-2}\ldots\alpha_0)_{n+1}$, and converting it into a decimal number d as follows:

$$d = \sum_{i=0}^{r-1} \alpha_i (n+1)^i. \tag{1}$$

This computation can be considered as a completely specified function from $\vec{\alpha}$ to d that is compatible with the index generation function i_{dx}. It is well-known that EVMDDs can represent the function converting into decimal numbers with $r+1$ nodes [21], [27]. Thus, we produce indices using (1).

Then, the produced indices are encoded with the one-hot encoding for input of χ'. The one-hot encoded indices d_1 are obtained by a bit shift operation (left shift) "<<" of a unit vector $\vec{e}$, as follows:

$$d_1 = \vec{e} << d,$$

where $\vec{e}$ is an $(n+1)^r$-bit unit vector $(0, 0, \ldots, 0, 1)$. This bit shift operation can be computed using each term $\alpha_i(n+1)^i$ of (1) sequentially, instead of d, as follows:

$$\begin{aligned} d_1 = (((\vec{e} << (\alpha_0(n+1)^0)) << (\alpha_1(n+1)^1)) \\ << \ldots << (\alpha_{r-1}(n+1)^{r-1})). \end{aligned} \tag{2}$$

To represent (2) compactly, we present a decision diagram that is a variant of a VEVMDD [28] and a factored EVBDD (FEVBDD) [21].

Definition 9: An **MDD with edge values for shifting** is a variant of a VEVMDD and an FEVBDD, and its m-branch nonterminal nodes are based on the following extended Shannon expansion:

$$\begin{aligned} f = Y_i^0 f_0 + Y_i^1 (f_1 << s_1(i)) + \cdots \\ + Y_i^{m-1} (f_{m-1} << s_{m-1}(i)), \end{aligned}$$

where Y_i is a multiple-valued variable represented by a nonterminal node, Y_i^j is its literal that is

$$Y_i^j = \begin{cases} 1 & (Y_i = j) \\ 0 & (Otherwise), \end{cases}$$

$s_j(i)$ is an edge value, and f_j is a cofactor with $f(Y_i = j)$. The terminal node represents a unit vector $\vec{e} = (0, 0, \ldots, 1)$. The unit vector $\vec{e}$ is shifted sequentially by values $s_j(i)$ of edges traversed from the root node to the terminal node. For convenience, we call this **SEVMDD**.

Let a variable Y_i be α_i, an edge value $s_j(i)$ be $j(n+1)^i$, and $\vec{e}$ at the terminal node be the $(n+1)^r$-bit unit vector. Then, we can represent (2) by an SEVMDD.

Example 7: Figure 4 shows an SEVMDD representing (2) for i_{dx} in Table 3. Each edge has a shift amount of the 4^3-bit unit vector $\vec{e} = (0, 0, \ldots, 0, 1)$.

Consider an $\vec{\alpha} = (1, 0, 2)$. By traversing edges of the SEVMDD from the root node to the terminal node according to values of $\vec{\alpha}$, and shifting $\vec{e}$ with 16, 0, and 2 sequentially, we obtain an one-hot encoded index whose only 19th bit from the right is 1. □

Theorem 3: An SEVMDD for an r-variable function transforming $\vec{\alpha}$ into a one-hot encoded index has exactly $r+1$ nodes, regardless of the order of variables.

(Proof) It is similar to the proof of Theorem 1. Since the resultant one-hot vector is obtained independently of the

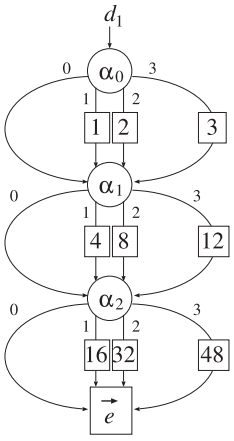

Fig. 4 SEVMDD for map from $\vec{\alpha}$ to one-hot encoded indices.

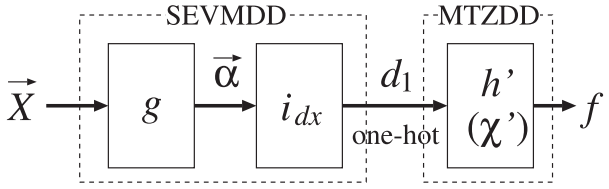

Fig. 5 Functional decomposition and their decision diagrams.

order of applying the shift operations, we have the theorem. ∎

4.4 Combining Two Functions g and i_{dx}

Section 4.3 showed how to produce one-hot encoded indices d_1 from $\vec{\alpha}$ using (2). However, we can produce d_1 from input vectors $\vec{X}$ as well, instead of $\vec{\alpha}$. Each α_i in $\vec{\alpha}$ represents the number of X_j's whose values are i. Thus, instead of shifting by α_i at once, we can shift $\vec{e}$ partially and sequentially by each $X_j = i$ to produce the same indices. The amount of shifting is $(n+1)^i$ bits for each $X_j = i$.

That is, by setting X_j to a variable Y_j for an SEVMDD and setting $(n+1)^i$ to its edge value $s_i(j)$, we can obtain an SEVMDD for a composite function $i_{dx}(g(\vec{X}))$ that produces one-hot encoded indices d_1 from $\vec{X}$. Note that in the obtained SEVMDD for $i_{dx}(g(\vec{X}))$, 0-edges have non-zero edge values $s_0(j) = (n+1)^0 = 1$. However, it is easy to normalize the SEVMDD so that all 0-edges have the zero value. Since the normalization process is the same as one for ordinary EVMDDs, we omit its detail.

For the size of the SEVMDD for $i_{dx}(g(\vec{X}))$, the following corollary can be easily derived from Theorem 3.

Corollary 1: Let $i_{dx}(g(\vec{X}))$ be an n-variable $(n+1)^r$-bit output composite function that transforms $\vec{X}$ into a one-hot encoded index. Then, an SEVMDD for $i_{dx}(g(\vec{X}))$ has exactly $n+1$ nodes, regardless of the order of variables.

Figure 5 shows the relation between the proposed decomposition of symmetric multiple-valued functions and its decision diagram based representation. In this way, we can represent any n-variable symmetric r-valued function using two decision diagrams: an SEVMDD and an MTZDD. From Theorem 2 and Corollary 1, the total number of nonterminal nodes needed to represent a symmetric function is

$$N_\alpha + n.$$

In the proposed representation method, $\vec{\alpha}$ is unnecessary as a result. However, $\vec{\alpha}$ plays an important role to derive this compact representation.

Since a standard MDD for an n-variable symmetric r-valued function requires $O(n^r/r!)$ nodes [4], the proposed method using an SEVMDD and an MTZDD requires much smaller nodes for large n.

Algorithm 1: Construction of SEVMDD for $i_{dx}(g(\vec{X}))$

Input: the number of variables n and the number of values r Output: the SEVMDD for $i_{dx}(g(\vec{X}))$
1. Create the terminal node with $\vec{e}$ with $(n+1)^r$ bits. 2. For each X_i $(i = n, n-1, \ldots, 2, 1)$, do the following: 2-1. Create a nonterminal node labeled with X_i. 2-2. Connect all edges to the topmost node. 2-3. Set $(n+1)^j$ to the edge value of the j-edge. 3. Normalize the SEVMDD so that all 0-edges have the zero value.

Algorithm 2: Construction of MTZDD for χ'

Input: N_α pairs of a one-hot encoded index and a function value of f and the number of values r Output: an MTZDD for χ'
1. Create r terminal nodes labeled with each valid function value. 2. Create a terminal node labeled with the invalid value $\emptyset$. 3. For each pair of index$_i$ and f $(i = N_\alpha - 1, N_\alpha - 2, \ldots, 1, 0)$, do the following: 3-1. Create a nonterminal node labeled with index$_i$. 3-2. Connect the 0-edge to the topmost node or the terminal node $\emptyset$ (if $i = N_\alpha - 1$). 3-3. Connect the 1-edge to a terminal node for f.

4.5 Algorithms to Construct SEVMDD and MTZDDs

This subsection shows algorithms to construct the SEVMDD for the composite function $i_{dx}(g(\vec{X}))$ shown in Sect. 4.3 and MTZDDs for χ' shown in Sect. 4.2. The structure of the SEVMDD for $i_{dx}(g(\vec{X}))$ is unchanged for any n-variable r-valued symmetric function. Thus, it is enough to construct an SEVMDD for each n and r only once. On the other hand, an MTZDD has to be constructed for each symmetric function f since χ' depends on the function values of f.

Algorithm 1 shows how to construct the SEVMDD for $i_{dx}(g(\vec{X}))$. Its time complexity is clearly $O(n)$. Algorithm 2 shows how to construct an MTZDD for χ'. Its time complexity is $O(N_\alpha)$. In this way, both algorithms construct SEVMDD and MTZDD efficiently in linear time of their input size.

5. Experimental Results

To evaluate the effectiveness of the proposed representation

Table 4 Average number of nodes in MDD and proposed method.

n	r	MDD	Proposed Method			Ratio
			SEVMDD	MTZDD	Total	(%)
3	3	11.6	4	13.8	17.8	153
4	3	19.6	5	18.8	23.8	121
5	3	32.6	6	24.9	30.9	95
6	3	51.1	7	32.0	39.0	76
7	3	74.9	8	40.0	48.0	64
8	3	104.9	9	49.0	58.0	55
9	3	142.3	10	59.0	69.0	48
10	3	187.8	11	70.0	81.0	43
11	3	242.5	12	82.0	94.0	39
12	3	302.8	13	95.0	108.0	36
13	3	386.6	14	109.0	123.0	32
3	4	18.8	4	25.0	29.0	154
4	4	38.6	5	40.0	45.0	117
5	4	71.9	6	61.0	67.0	93
6	4	124.2	7	89.0	96.0	77
7	4	200.5	8	125.0	133.0	66
8	4	309.2	9	170.0	179.0	58
9	4	452.8	10	225.0	235.0	52
10	4	642.4	11	291.0	302.0	47
11	4	886.9	12	369.0	381.0	43
3	5	26.0	4	41.0	45.0	173
4	5	60.8	5	76.0	81.0	133
5	5	130.4	6	132.0	138.0	106
6	5	254.9	7	216.0	223.0	87
7	5	459.9	8	336.0	344.0	75
8	5	778.5	9	501.0	510.0	66
9	5	1,252.1	10	721.0	731.0	58

Ratio = Total / MDD $\times$ 100

method quantitatively, we compare the total number of nodes in the proposed method with the number of nodes in standard MDDs for randomly generated n-variable r-valued symmetric functions. For this size evaluation, we randomly generated 10 symmetric functions[†] in various n and r, and represented them by standard MDDs and the proposed method. Table 4 shows their average number of nodes rounded to one decimal place for 10 symmetric functions. Since the proposed representation method consists of two types of decision diagrams: an SEVMDD and an MTZDD, Table 4 shows the size of "SEVMDD", the size of "MTZDD", and their "Total" size. The column "Ratio" shows the ratio of the total size to the size of standard MDD in percentage.

As shown in Table 4, when n is small ($n = 3$ to 5 in this experiment), MTZDDs are larger than standard MDDs because of the overhead caused by the one-hot encoding that increases the number of input variables. On the other hand, when n is large ($n \geq 6$), MTZDDs are smaller than MDDs, resulting in more compact representations even including the size of SEVMDDs. As the number of variables n increases, the ratio of of the total size in the proposed method to the size of MDD gets small. Thus, we can say that the size complexity of the proposed method is lower than that of MDDs.

For 10 symmetric functions generated for each n and r, their 10 MDDs have different sizes, depending on a combination of generated function values. On the other hand, 10 SEVMDDs have the same size, as shown in Corollary 1. Similarly, MTZDDs also have the same size except for the cases of $n = 3$ to 5 and $r = 3$. When $n = 3$ to 5 and $r = 3$, only a few functions have 1 or 2 of 3 kinds of function values. However, in even such a case, the number of *nonterminal nodes* in an MTZDD is exactly the same as N_α, as shown in Theorem 2. Thus, the proposed method can compactly represent symmetric multiple-valued functions, regardless of combinations of function values.

[†]They are obtained by generating random integers from 0 to $r - 1$ as function values in a table of $\vec{\alpha}$, as shown in Table 2.

6. Conclusion and Future Works

This paper proposes a decomposition method of a symmetric multiple-valued function into three parts. We also propose an SEVMDD and an MTZDD to represent the three parts compactly. This paper derived some theorems on sizes of the SEVMDD and the MTZDD. The theorems and experimental results using randomly generated symmetric multiple-valued functions showed that the size complexity of the proposed decomposition-based representation method is lower than the size complexity of ordinary MDDs.

The proposed decomposition method includes an index generation function. Thus, a decomposition method for index generation functions [24] would be useful to reduce the size complexity furthermore. Since our decomposition based method can require longer time to evaluate functions, studying evaluation methods targeting some applications would be more practical. In addition, investigating a decomposition method for maximally asymmetric functions [17] would be interesting.

Acknowledgments

This research is partly supported by the JSPS KAKENHI Grant (C), No.23K11038, 2023. The reviewers' comments were helpful in improving the paper.

References

[1] S.B. Akers, "Binary decision diagrams," IEEE Trans. Comput., vol.C-27, no.6, pp.509–516, June 1978.
[2] R.C. Born, "An iterative technique for determining the minimal number of variables for a totally symmetric function with repeated variables," IEEE Trans. Comput., vol.C-21, no.10, pp.1129–1131, Oct. 1972.
[3] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[4] J.T. Butler, D.S. Herscovici, T. Sasao, and R.J. Barton III, "Average and worst case number of nodes in decision diagrams of symmetric multiple-valued functions," IEEE Trans. Comput., vol.46, no.4, pp.491–494, April 1997.
[5] J.T. Butler and T. Sasao, "On the properties of multiple-valued functions that are symmetric in both variable values and labels," 1998 28th IEEE International Symposium on Multiple-Valued Logic, pp.83–88, 1998.
[6] J.T. Butler and T. Sasao, "Maximally asymmetric multiple-valued functions," 2019 IEEE 49th International Symposium on Multiple-Valued Logic, pp.188–193, 2019.
[7] A. Canteaut and M. Videau, "Symmetric boolean functions," IEEE

Trans. Inf. Theory, vol.51, no.8, pp.2791–2811, Aug. 2005.
[8] B. Dahlberg, "On symmetric functions with redundant variables–weighted functions," IEEE Trans. Comput., vol.C-22, no.5, pp.450–458, May 1973.
[9] T. Kam, T. Villa, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," Multiple-Valued Logic: An International Journal, vol.4, no.1-2, pp.9–62, 1998.
[10] Z. Kohavi, Switching and Finite Automata Theory, McGraw-Hill Book Company, 1979.
[11] D.T. Lee and S.J. Hong, "An algorithm for transformation of an arbitrary switching function to a completely symmetric function," IEEE Trans. Comput., vol.C-25, no.11, pp.1117–1123, Nov. 1976.
[12] L. Breiman, "Random forests," Machine Learning, vol.45, no.1, pp.5–32, 2001.
[13] S. Maitra and P. Sarkar, "Maximum nonlinearity of symmetric boolean functions on odd number of variables," IEEE Trans. Inf. Theory, vol.48, no.9, pp.2626–2630, Sept. 2002.
[14] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," 30th Design Automation Conference, pp.272–277, 1993.
[15] S. Nagayama, T. Sasao, and J.T. Butler, "A systematic design method for two-variable numeric function generators using multiple-valued decision diagrams," IEICE Trans. Inf. & Syst., vol.E93-D, no.8, pp.2059–2067, Aug. 2010.
[16] S. Nagayama, T. Sasao, and J.T. Butler, "Analysis of multi-state systems with multi-state components using EVMDDs," 2012 IEEE 42nd International Symposium on Multiple-Valued Logic, pp.122–127, May 2012.
[17] S. Nagayama, T. Sasao, and J.T. Butler, "On decision diagrams for maximally asymmetric functions," 2022 IEEE 52nd International Symposium on Multiple-Valued Logic, pp.164–169, May 2022.
[18] S. Nagayama, T. Sasao, and J.T. Butler, "Decomposition-based representation of symmetric multiple-valued functions," 2023 IEEE 53rd International Symposium on Multiple-Valued Logic, pp.76–81, May 2022.
[19] J. Pieprzyk and C.X. Qu, "Fast hashing and rotation symmetric functions," Journal of Universal Computer Science, vol.5, no.1, pp.20–31, 1999.
[20] P. Sarkar and S. Maitra, "Balancedness and correlation immunity of symmetric boolean functions," Electronic Notes in Discrete Mathematics, vol.15, pp.176–181, 2003.
[21] T. Sasao and M. Fujita (eds.), Representations of Discrete Functions, Kluwer Academic Publishers 1996.
[22] T. Sasao, Switching Theory for Logic Synthesis, Kluwer Academic Publishers, 1999.
[23] T. Sasao, Memory-Based Logic Synthesis, Springer New York, 2011.
[24] T. Sasao, "Index generation functions: Recent developments," 2011 41st IEEE International Symposium on Multiple-Valued Logic, pp.1–9, May 2011.
[25] P. Savický, "On the bent boolean functions that are symmetric," European J. Combinatorics, vol.15, no.4, pp.407–410, 1994.
[26] I. Stojmenović, "On sheffer symmetric functions in three-valued logic," Discrete Applied Mathematics, vol.22, no.3, pp.267–274, 1988.
[27] I. Wegener, Branching Programs and Binary Decision Diagrams: Theory and Applications, SIAM, 2000.
[28] B. Xue, S. Nagayama, M. Inagi, and S. Wakabayashi, "A programmable architecture based on vectorized EVBDDs for network intrusion detection using random forests," International Symposium on Nonlinear Theory and Its Applications, pp.132–135, 2017.
[29] S.S. Yau and Y.S. Tang, "Transformation of an arbitrary switching function to a totally symmetric function," IEEE Trans. Comput., vol.C-20, no.12, pp.1606–1609, Dec. 1971.

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. His research interest includes decision diagrams, analysis of multi-state systems, logic design for index generation functions, and multiple-valued logic.

Tsutomu Sasao received the B.E., M.E., and Ph.D. 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, NY, the Naval Postgraduate School, Monterey, CA, Kyushu Institute of Technology, Iizuka, Japan, and Meiji University, Kawasaki, Japan. Now, he is a visiting researcher at 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, Logic Synthesis and Verification, Memory-Based Logic Synthesis, Index Generation Functions, and Classification Functions for Machine Learning and Data Mining, in 1993, 1996, 1999, 2001, 2011, 2019, and 2023, 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, Takeda Techno-Entrepreneurship Award in 2001, and Distinctive Contribution Awards from the IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003, 2004 and 2012. 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. From 1987, he served on the faculty of the Naval Postgraduate School, in Monterey, California, retiring at the rand of Distinguished Professor in 2013. 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.