2018 IEEE 48th International Symposium on Multiple-Valued Logic

On a Memory-Based Realization of Sparse Multiple-Valued Functions

Tsutomu Sasao
Department of Computer Science, Meiji University,
Kawasaki, Kanagawa 214-8571, Japan

***Abstract*—This paper presents multi-valued (MV) functions, which are generalizations of index generation functions and switching functions. First, an efficient memory-based realization of sparse MV functions, where the number of specified combinations is much smaller than the number of possible input combinations, is presented. Then, a formula for the expected number of variables to represent random sparse MV functions is derived. Finally, the theoretical analysis is compared with the experimental results.**

***Keywords*-logic minimization; incompletely specified function; statistical analysis; random function.**

I. Introduction

One of the important tasks in information processing is to partition the given data into classes.

The simplest case is to partition the data into two classes. A partition of the set of binary vectors of n bits into two sets can be represented by a **binary logic function** (or a switching function):

$$f : \{0,1\}^n \rightarrow \{0,1\}.$$

Another case is a partition of the set of vectors into single elements. A partition of a subset D of p-nary vectors of n digits into k distinct elements can be represented by an **index generation function** [7]:

$$f : D \rightarrow \{1,2,\ldots,k\}.$$

In this paper, we introduce a **multi-valued function** (MV function), which partitions a set D of p-nary vectors of n digits into q disjoint sets:

$$f : D \rightarrow \{1,2,\ldots,q\}.$$

Note that, in the case of an index generation function, $|D| = k = q$, while in the case of an MV function, $|D| = k > q$, where $|D|$ denotes the number elements in D. Thus, an MV function is a generalization of an index generation function, and also a generalization of a binary logic function.

When $|D| < p^n$, the function is **incompletely specified** (or partially defined). When $|D| \ll p^n$, the function is **sparse**. Sparse functions of n variables can be often represented with fewer variables than n.

In this paper, we present an efficient memory-based method to realize sparse MV functions. Also, we derive a formula for the expected number of variables to represent random sparse MV functions.

TABLE 2.1
Registered vector table.

Vector				Index
x_1	x_2	x_3	x_4	
1	0	0	0	1
0	1	0	0	2
0	0	1	0	3
1	0	1	1	1

The rest of the paper is organized as follows: Section II defines the MV function; Section III considers the number of variables to represent sparse MV functions; Section IV shows efficient methods to implement sparse MV functions; Section V derives the expected number of variables to represent random sparse MV functions; Section VI shows experimental results; Section VII surveys related works; and Section VIII concludes the paper.

II. MV Functions

In this part, we introduce MV functions.

Definition 2.1: Let D be a set of k distinct p-nary vectors of n digits. That is, $D \subseteq P^n$, where $P = \{0,1,\ldots,p-1\}$, and $|D| = k$. These vectors are **registered vectors**. Assume that these vectors are partitioned into q disjoint subsets: $T_1, T_2, \ldots, T_q$, where $T_1 \cup T_2 \cup \ldots \cup T_q = D$ and $T_i \cap T_j = \phi$ for $(i \neq j)$. For each subset T_i, assign a unique **index** from 1 to q. A **registered vector table** shows the index of each registered vector. An **incompletely specified MV function** is a mapping $D \rightarrow Q$, where $Q = \{1,2,\ldots,q\}$, and $D \neq P^n$. It produces the corresponding index i if the input matches a registered vector in T_i. The number of registered vectors in D is called the **weight**. The weight $|D| = k$ is often much smaller than p^n, the total number of possible input combinations. In such a case, the function is **sparse.** Note that p may be different from q.

Example 2.1: Table 2.1 shows a registered vector table for the MV function with $n = 4$, $p = 2$, $q = 3$, and weight $k = 4$. ■

III. Number of Variables to Represent a Sparse MV Function

In this part, we show that a sparse MV function f can often be represented with fewer variables than the original function, if its *don't care* values are properly replaced by index values.

Lemma 3.1: Assume that an MV function f is represented by a decomposition chart [4]. If no column of the decomposi-

2378-2226/18/$31.00 ©2018 IEEE
DOI 10.1109/ISMVL.2018.00017

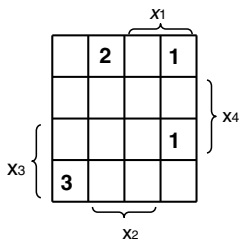

Fig. 3.1. 4-variable MV function.

tion chart has indices with different values, then the function can be represented by only the column variables.

Example 3.1: Consider the decomposition chart in Fig. 3.1. It shows the MV function in Table 2.1. x_1 and x_2 specify columns, while x_3 and x_4 specify rows. Also, blank cells denote *don't cares*. Since all care values in each column are the same, it is possible to represent the function with only the column variables x_1 and x_2:

$$f = 1 \cdot x_1 \vee 2 \cdot \bar{x}_1 x_2 \vee 3 \cdot \bar{x}_1 \bar{x}_2.$$

In this case, *don't care* values in the column $(x_1, x_2) = (1, 1)$ are assigned to 1. ■

Thus, the minimization of the variables is **to obtain a completely specified function of as few variables as possible, that is compatible with the given sparse function**.

As for an upper bound on the number of variables to represent index generation functions, by an extensive computer simulation using randomly generated functions, we have the following:

Conjecture 3.1: [10] When the number of the variables n is sufficiently large, most sparse index generation functions with weight k (≥ 7) can be represented by $\lceil 2\log_p(k+1) - \log_p 5.485 \rceil$ variables.

A lower bound on the number of variables to represent an index generation function is given by:

Theorem 3.1: [9] To represent an index generation function f with weight k, at least $LB = \lceil \log_p(k+1) \rceil$ variables are necessary.

IV. Realization of Completely Specified Functions

Consider the index generation function of 6 variables shown in Table 4.1. Note that only the outputs for 6 registered vectors are specified. Table 4.1 can be interpreted in two

TABLE 4.1
REGISTERED VECTOR TABLE FOR EXAMPLE 4.1.

Inputs						Index
x_1	x_2	x_3	x_4	x_5	x_6	
1	0	0	0	0	0	1
0	1	1	1	1	0	2
0	0	1	0	0	1	3
1	0	1	1	1	1	4
0	1	0	0	1	0	5
0	1	0	0	0	0	6

different ways. The first interpretation is that for non-registered vectors, the outputs are unspecified. In this case, it shows an incompletely specified function.

The second interpretation is that for non-registered vectors, the outputs are zero. In this case, it shows a completely specified function $f : \{0,1\}^6 \rightarrow \{0,1,2,\ldots,6\}$. Such a case occurs quite often in practical applications.

For example, assume that you have a trouble with many spam mails, and want to avoid them. In such a case, you can use a **white list** that stores registered senders. You can accept mails only from the sender in the white list. Note that the number of entries in the white list is much smaller than that of all possible addresses.[1]

In this part, we consider the case for the second interpretation. That is, a completely specified function, where the number of registered vectors is much smaller than the possible combinations. In such a case, we can often reduce the number of variables for the memory, and design an efficient memory-based circuit.

In this paper, we use binary hardware to implement functions, and assume that **the cost of the circuit is proportional to the total memory size**, since the area for the memory is dominant in the LSI chip.

A. Circuit for Index Generation Functions

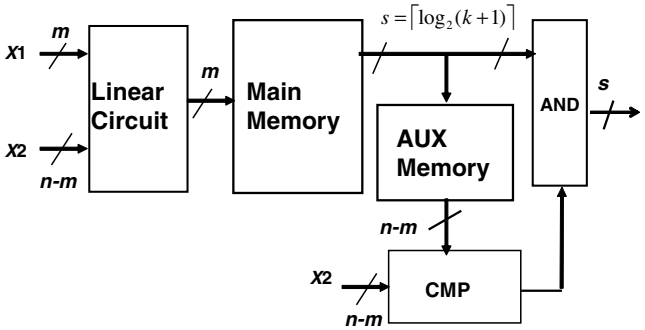

Fig. 4.1. Index Generation Unit (IGU).

Definition 4.1: An **index generation unit** (IGU) [7] is shown in Fig. 4.1. The **linear circuit** has n inputs and m outputs, where $m < n$. It produces different outputs for different registered vectors. It is used to reduce the size of the main memory. Let $X_1 = (x_{i_1}, x_{i_2}, \ldots, x_{i_m})$ and $X_2 = (x_{i_{m+1}}, x_{i_{m+2}}, \ldots, x_{i_n})$ be a partition of the input variables $X = (x_1, x_2, \ldots, x_n)$. The **main memory** has m inputs and $s = \lceil \log_2(k+1) \rceil$ outputs. The main memory produces correct indices for registered vectors. However, it may produce incorrect indices for non-registered vectors, because the number of the input variables to the main memory is reduced. To check whether the main memory produces the correct index or not, we use the **AUX memory**. The AUX memory has $s = \lceil \log_2(k+1) \rceil$ inputs and $(n-m)$ outputs. It stores the X_2 part of the registered vectors for

[1]IP addresses used in the internet are often represented with 128-bit numbers. Even if the white list contains a million entries, only 2.94×10^{-33} of the possible combinations are specified.

each index. The **comparator** (CMP) checks if the X_2 part of the inputs are the same as the X_2 part of the registered vector. If they are the same, the main memory produces a correct index. Otherwise, the main memory produces a wrong index, and the input vector is non-registered. In this case, the **output AND gates** produce 0, showing that the input vector is non-registered. Thus, the IGU realizes a completely specified function $f : \{0,1\}^n \rightarrow \{0,1,2,\ldots,k\}$. In this way, the main memory can implement an incompletely specified index generation function instead of a completely specified one. When the output value of the comparator is 0, the output of the circuit is 0, corresponding to a non-registered vector.
An index generation function can be realized as follows:

Algorithm 4.1: (Realization of an Index Generation Function)

1) Given a set of registered vectors. Partition the input variables X into two (X_1, X_2), so that for any pair of distinct registered vectors (A_1, A_2) and (B_1, B_2), $A_1 \neq B_1$ holds.
2) In the main memory, realize the index, where X_1 denotes the inputs to the main memory. For non-registered vectors, realize 0.
3) In the AUX memory, realize the X_2 part of the registered vectors for each index.
4) Check if the input vector is a registered vector or not by the comparator.

Example 4.1: Consider the index generation function in Table 4.1.

Single LUT realization: A LUT requires $2^6 \times 3 = 192$ bits, because to represent 6 different output values, three bits are necessary.

Realization using an IGU: Let $X = (x_1, x_2, \ldots, x_6)$ be the input variables. Let (X_1, X_2) be the partition of X, where $X_1 = (x_1, x_3, x_4)$ and $X_2 = (x_2, x_4, x_5)$. In this case, the linear circuit realizes the transformation:

$$y_1 = x_1;\ y_2 = x_3;\ y_3 = x_5.$$

Consider the function g in Table 4.2. In this case, for any pair of distinct registered vectors, (A_1, A_2) and (B_1, B_2), $g(A_1) \neq g(B_1)$ holds. Thus, the registered vectors in Table 4.1 can be distinguished with only three variables: $(y_1, y_2, y_3) = (x_1, x_3, x_5)$. Consider the realization with an index generation unit (IGU) shown in Fig. 4.1. In this case, $n = 6$, $m = 3$, $k = 6$, $s = 3$, $n - m = 3$, $X_1 = (x_1, x_3, x_5)$, and $X_2 = (x_2, x_4, x_6)$.

The linear circuit realizes (y_1, y_2, y_3). The main memory realizes the function g in Table 4.2, where (w_1, w_2, w_3) denotes the outputs. The AUX memory realizes the function in Table 4.3. It realizes $(\underline{x}_2, \underline{x}_4, \underline{x}_6)$, the corresponding values of $X_2 = (x_2, x_4, x_6)$ for the input (w_1, w_2, w_3). Note that with the values for (y_1, y_2, y_3), the values for (x_1, x_3, x_5) can be obtained as follows:

$$x_1 = y_1;\ x_3 = y_2;\ x_5 = y_3.$$

Thus, if the input values of (x_2, x_4, x_6) are equal to the output values of the second memory $(\underline{x}_2, \underline{x}_4, \underline{x}_6)$, then the input $(x_1, x_2, \ldots, x_6)$ is a registered vector. Hence, the IGU in Fig. 4.1 realizes the given function. In this case, the total memory size in the IGU is $2^3 \times 3 + 2^3 \times 3 = 48$ bits, which is reduced to a quarter of the single LUT realization. ■

TABLE 4.2
FUNCTION g REALIZED BY MAIN MEMORY IN EXAMPLES 4.1 AND 4.2.

Inputs			Outputs			Index
y_1	y_2	y_3	w_1	w_2	w_3	
1	0	0	0	0	1	1
0	1	1	0	1	0	2
0	1	0	0	1	1	3
1	1	1	1	0	0	4
0	0	1	1	0	1	5
0	0	0	1	1	0	6
1	0	1	0	0	0	–
1	1	0	0	0	0	–

TABLE 4.3
FUNCTION REALIZED BY AUX MEMORY IN EXAMPLE 4.1.

Inputs			Outputs		
w_1	w_2	w_3	$\underline{x}_2$	$\underline{x}_4$	$\underline{x}_6$
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0

B. Circuit for MV Functions

Consider the 6-variables function shown in Table 4.4. It is not an index generation function, but an MV function. Unfortunately, an IGU in Fig. 4.1 cannot be used to implement the MV function, since the same values are produced for two different registered vectors. To implement an MV function, we use the following:

TABLE 4.4
REGISTERED VECTOR TABLE FOR EXAMPLE 4.2.

Vector						Index
x_1	x_2	x_3	x_4	x_5	x_6	
1	0	0	0	0	0	1
0	1	1	1	1	0	1
0	0	1	0	0	1	2
1	0	1	1	1	1	2
0	1	0	0	1	0	3
0	1	0	0	0	0	3

Definition 4.2: The **extended index generation unit** (EIGU) shown in Fig. 4.2 consists of a **linear circuit**, **the first memory** (first memory), **the second memory** (second memory), a **comparator** (CMP), and **output AND gates**. The linear circuit produces m linear functions from n input variables. It produces different outputs for different registered vectors. The first memory has m inputs and produces the temporary output with $s = \lceil \log_2(k+1) \rceil$ bits. The second memory has s inputs and produces $(n-m)$ bits showing X_2, the part of the corresponding registered vector. In addition, it also produces the corresponding output values for the function. The comparator compares the outputs of the second memory with X_2. If they are the same, it produces 1. Otherwise,

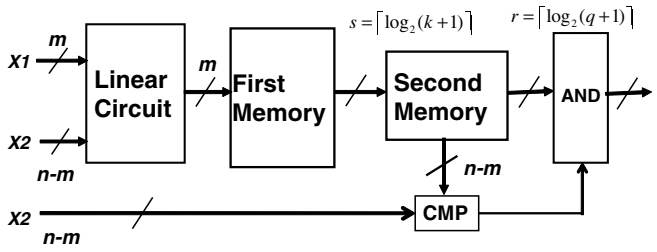

Fig. 4.2. Extended Index Generation Unit (EIGU).

it produces 0. The AND gates produce the same values as the second memory if the output of the comparator is 1. If not, the AND gates produce zeros, which show that the input vector does not match the registered vector. Since the number of different outputs for the function is q, the number of outputs from the second memory to the AND gates is $r = \lceil \log_2(q+1) \rceil$. The additional $+1$ is needed to include non registered vectors. Thus, the EIGU realizes a completely specified function $f : \{0,1\}^n \rightarrow \{0,1,2,\ldots,q\}$.

The design algorithm for an MV function is similar to that for an index generation function.

Example 4.2: Consider the MV function in Table 4.4.

Single LUT realization: A LUT requires $2^6 \times 2 = 128$ bits, because to represent three different output values, two bits are necessary.

Realization using an EIGU: Consider the partition of the input variables. $X_1 = (x_1, x_3, x_5), X_2 = (x_2, x_4, x_6)$. Let

$$y_1 = x_1;\ y_2 = x_2;\ y_3 = x_5.$$

Again, the registered vectors in Table 4.4 can be distinguished with only three variables: $(y_1, y_2, y_3) = (x_1, x_3, x_5)$. Consider the realization of Table 4.4 with an extended index generation unit (EIGU) shown in Fig. 4.3, where $n = 6$, $m = 3$, $k = 6$, $s = 3$, $n - m = 3$, $X_1 = (x_1, x_3, x_5)$, and $X_2 = (x_2, x_4, x_6)$. The first memory realizes the function in Table 4.2. The second memory realizes the function in Table 4.5. It realizes $(\underline{x}_2, \underline{x}_4, \underline{x}_6)$, the corresponding values of (x_2, x_4, x_6) for the input (w_1, w_2, w_3), and the output values of the function (z_2, z_1). Note that with the values for (y_1, y_2, y_3), the values for $X_1 = (x_1, x_3, x_5)$ can be obtained as follows:

$$x_1 = y_1;\ x_3 = y_2;\ x_5 = y_3.$$

Thus, if the input values of (x_2, x_4, x_6) are equal to the output values of the second memory $(\underline{x}_2, \underline{x}_4, \underline{x}_6)$, then the input $(x_1, x_2, \ldots, x_6)$ is a registered vector. Hence, the EIGU in Fig. 4.3 realizes the given function. In this case, the total memory size in the EIGU is $2^3 \times 3 + 2^3 \times 5 = 64$ bits, which is reduced to a half of the single LUT realization[2]. ∎

V. Expected Number of Variables to Represent Random Sparse MV Functions

In this part, we derive the expected number of variables to represent random sparse MV functions.

[2]In this case, the first memory and the second memory can be merged. Thus, the size of the memory can be reduced to $2^3 \times 5 = 40$ bits.

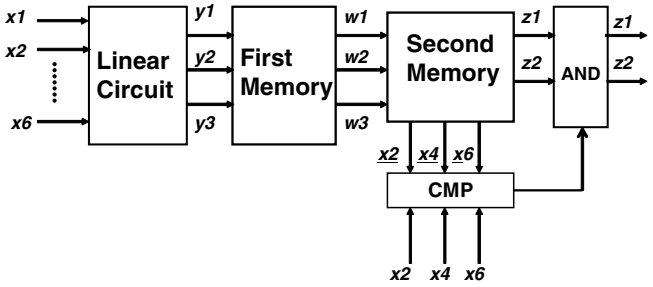

Fig. 4.3. EIGU for Example 4.2.

TABLE 4.5
Second Memory in Example 4.2.

Vector			Outputs				
w_1	w_2	w_3	$\underline{x}_2$	$\underline{x}_4$	$\underline{x}_6$	z_2	z_1
0	0	1	0	0	0	0	1
0	1	0	1	1	0	0	1
0	1	1	0	0	1	1	0
1	0	0	0	1	1	1	0
1	0	1	1	0	0	1	1
1	1	0	1	0	0	1	1

Assumption 5.1: **A random sparse MV function** $f : P^n \rightarrow \{d, 1, 2, \ldots q\}$, where $P = \{0, 1, 2, \ldots, p-1\}$, and d denotes a *don't care*, satisfies the following: For any input combination $\vec{a} \in P^n$, the probability such that $f(\vec{a}) = i$ is $\alpha_i = \alpha$ for any $i \in Q = \{1, 2, \ldots, q\}$. We assume that $\alpha \ll 1$.

Thus, the expected number of input combinations such that $f(\vec{a}) = i$ is $u = p^n\alpha$. Also, the expected total number of *care elements* is qu.

Lemma 5.1: [5] Let $f : D \rightarrow Q$, where $D \subset P^n$, $P = \{0, 1, \ldots, p-1\}$, and $Q = \{1, 2, \ldots, q\}$, be a random sparse MV function. Then, the probability that a certain variable x_{i_1} is redundant in f is given by

$$\delta_1(n, p, q, u) \simeq \gamma_1^M,$$

where $\gamma_1 = q(\alpha + \beta)^p - (q-1)\beta^p$, $\alpha = \frac{u}{p^n}$, $\beta = 1 - q\alpha$, $M = p^{n-1}$, and u denotes the expected number of input combinations such that $f(\vec{a}) = i$ for $i \in Q$.

Lemma 5.2: [5] Let $f : D \rightarrow Q$, where $D \subset P^n$, $P = \{0, 1, \ldots, p-1\}$ and $Q = \{1, 2, \ldots, q\}$, be a random sparse MV function. Then, the probability that f has a certain set of r redundant variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ is

$$\delta_r(n, p, q, u) \simeq \gamma_r^M,$$

where $\gamma_r = q(\alpha + \beta)^{p^r} - (q-1)\beta^{p^r}$, $\alpha = \frac{u}{p^n}$, $\beta = 1 - q\alpha$, $M = p^{n-r}$, and u denotes the expected number of input combinations such that $f(\vec{a}) = i$ for $i \in Q$.

Lemma 5.3: For an n variable random sparse MV function f, let δ_r be the probability that a set of r variables is redundant. Then, the probability that any group of r variables are not redundant in f is

$$\lambda_r(n, p, q, u) \simeq (1 - \delta_r)^{\binom{n}{r}},$$

where u denotes the expected number of input combinations such that $f(\vec{a}) = i$ for $i \in Q$.

(Proof) The probability that $\{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ is not redundant is $1 - \delta_r$. The probability that all the groups of r variables are not redundant in f is $(1 - \delta_r)^{\binom{n}{r}}$. □

From the above lemmas, we can derive the expected number of variables necessary to represent random sparse MV functions.

Theorem 5.1: In an n-variable random sparse MV function $f : D \to Q$, where $D \subset P^n$, $P = \{0, 1, \ldots, p-1\}$, and $Q = \{1, 2, \ldots, q\}$. Let $\delta_r(n, p, q, u)$ be the probability that a set of r variables are redundant in a random sparse n variable MV function f, where u denotes the expected number of input combinations such that $f(x) = i$ for $i \in Q$. Then, $Epc(n, p, q, u)$, the expected number of variables to represent a random sparse n-variable MV function, is

$$Epc(n, p, q, u) \simeq \sum_{r=1}^{n} (1 - \delta_r)^{\binom{n}{r}},$$

where $\delta_r = \gamma_r^M$, $\gamma_r = q(\alpha + \beta)^{p^r} - (q-1)\beta^{p^r}$, $\alpha = \frac{u}{p^n}$, $\beta = 1 - q\alpha$, and $M = p^{n-r}$.

(Proof) Consider groups of at least one variable. Let λ_r be the probability that any group of r variables are not redundant in f. Then, the probability that there exists a representation of f using a group of $n - r$ variables is $\mu_r = 1 - \lambda_r$. The probability that f has a representation using exactly $n - r$ variables is $\mu_r - \mu_{r+1}$. Let $\lambda_0 = 0$. Then

$$\begin{aligned} & Epc(n, p, q, u) \\ \simeq & \sum_{i=0}^{n} (n-i)(\mu_i - \mu_{i+1}) = \sum_{i=0}^{n} (n-i)(\lambda_{i+1} - \lambda_i) \\ = & \sum_{i=0}^{n} (n-i)\lambda_{i+1} - \sum_{i=0}^{n} (n-i)\lambda_i \\ = & \sum_{j=1}^{n+1} (n-j+1)\lambda_j - \sum_{i=0}^{n} (n-i)\lambda_i = \sum_{i=1}^{n} \lambda_i \end{aligned}$$

By Lemma 5.3, we have the theorem. □

In this section, we used probability to estimate the number of variables to represent random sparse functions. In the proofs of the theorem and lemmas, we assumed independence of certain events, which does not hold for functions with a few variables, or for functions with small weights. In the next section, we show that these approximations are reasonably accurate for functions with a moderate number of variables.

VI. Experimental Results

We used a computer program to find minimum sets of variables for MV functions [5], [6].

To see the validity of Theorem 5.1, for each set of parameters (n, p, q, u), we randomly generated 1000 sample functions and obtained the minimum sets of variables to represent the functions, where u denotes the **exact** number of elements in each subset T_i. Table 6.1 shows the experimental results. In the table, the first column shows n, the number of input variables; the second column shows p; the third column shows q; the fourth column shows u, the number of elements in each subset T_i; the fifth column shows k, the weight of the function; the sixth column shows $Epc(n, p, q, u)$, the values derived from Theorem 5.1; and the last column denotes experimental results. For example, when $(n, p, q, u) = (20, 2, 2, 32)$, an estimate using Theorem 5.1 shows that $Epc(n, p, q, u) = 6.44$. On the other hand, randomly generated functions required 6.994 variables, on the average. In this experiment, only the reduction of original variables are considered.

TABLE 6.1
Numbers of variables to represent random sparse multi-valued functions.

n	p	q	u	$k = qu$	Theorem 5.1	Experiment
20	2	2	32	64	6.44	6.994
20	2	2	64	128	8.72	8.990
20	2	2	128	256	10.88	10.972
20	2	2	256	512	12.96	12.991
20	2	2	512	1024	14.99	15.143
20	2	20	4	80	8.00	8.836
20	2	20	8	160	10.02	10.588
20	2	20	16	320	12.20	12.606
20	2	20	32	640	14.65	14.792
20	2	20	64	1280	16.92	17.065
14	3	3	16	48	4.47	4.963
14	3	3	32	96	5.94	5.996
14	3	3	64	192	7.00	7.008
14	3	3	128	384	8.12	8.345
14	3	3	256	768	9.89	9.923
14	3	20	4	80	5.94	5.999
14	3	20	8	160	7.00	7.010
14	3	20	16	320	8.10	8.345
14	3	20	32	640	9.88	9.922
14	3	20	64	1280	11.00	11.078
14	3	20	128	2560	12.61	12.683
14	4	4	64	256	6.00	6.246
14	4	4	128	512	7.00	7.271
14	4	4	256	1024	8.00	8.419
14	4	4	512	2048	9.01	9.630
14	4	4	1024	4096	10.08	10.835
14	4	20	1	20	3.00	3.015
14	4	20	2	40	3.93	4.000
14	4	20	4	80	4.95	5.000
14	4	20	8	160	5.96	5.999
14	4	20	16	320	6.96	6.997
14	4	20	32	640	7.97	7.996
14	4	20	64	1280	8.98	9.008
14	4	20	128	2560	9.98	10.068

n: Number of the input variables
k: Weight of the function
Theorem 5.1: Expected number of variables $Epc(n, p, q, u)$
Experiment: Average of 1000 randomly generated functions.

Table 6.2 shows the results for binary input index generation functions (i.e., $p = 2$ and $u = 1$) for different values of $k = q$. When k is small, the discrepancy between estimated results and experimental results are large[3]. However, when k is large, $Epc(n, p, q, u)$ estimates the experimental results fairly well.

Table 6.3 shows the results for binary input index generation functions (i.e., $p = 2$ and $u = 1$) with weight $k = 255$ for different values of n. The necessary numbers of variables decrease as n increases. This result is consistent with the observation in [12].

[3]In Theorem 5.1, u denotes the **expected** number of input combinations such that $f(x) = i$, while in the experiments, u denotes the **exact** number.

TABLE 6.2
NUMBERS OF VARIABLES TO REPRESENT RANDOM SPARSE INDEX GENERATION FUNCTIONS.

n	p	q	u	$k = qu$	Theorem 5.1	Experiment [7]
20	2	15	1	15	3.059	4.947
20	2	31	1	31	5.162	6.115
20	2	63	1	63	7.492	8.007
20	2	127	1	127	9.784	10.000
20	2	255	1	255	11.926	11.996
20	2	511	1	511	13.980	14.019
20	2	1023	1	1023	16.049	16.293
20	2	2047	1	2047	18.670	18.758
20	2	4095	1	4095	19.993	19.992

n: Number of the input variables
k: Weight of the function
Theorem 5.1: Expected number of variables $Epc(n, p, q, u)$
Experiment: Average of 1000 randomly generated functions. This data is taken from Table 11.6 of [7].

TABLE 6.3
NUMBERS OF VARIABLES TO REPRESENT RANDOM SPARSE INDEX GENERATION FUNCTIONS.

n	p	q	u	$k = qu$	Theorem 5.1	Experiment
14	2	255	1	255	12.884	12.967
16	2	255	1	255	12.238	12.607
18	2	255	1	255	11.984	12.113
20	2	255	1	255	11.926	11.995
22	2	255	1	255	11.727	11.945
24	2	255	1	255	11.325	11.842
26	2	255	1	255	11.031	11.633
28	2	255	1	255	11.000	11.286
30	2	255	1	255	11.000	11.071

n: Number of the input variables
k: Weight of the function
Theorem 5.1: Expected number of variables $Epc(n, p, q, u)$
Experiment: Average of 1000 randomly generated functions.

VII. RELATED WORKS

In [2], Halatsis-Gaitanis, and in [3], Kambayashi considered the reduction of variables in incompletely specified two-valued logic functions. They presented algorithms to minimize the number of variables.

In [5], Sasao showed a minimization problem for MV functions. He obtained the expected number of variables to represent sparse MV functions by both statistical analysis and computer simulation. In [8], Sasao presented an index generation unit (IGU) that can efficiently implement an index generation function. A property of an incompletely specified function is used to reduce the number of inputs to the main memory. Also, he conjectured that to represent an index generation function with weight k, at most $m = 2\lceil \log_2(k+1) \rceil - 3$ variables are sufficient for most functions. In [9], Sasao presented an algorithm to find a good linear transformation to reduce the number of variables. He also presented the experimental results for random functions, IP address tables, and list of English words.

In [13], Simovici et al. showed a method to find a linear transformation using a difference matrix.

In [10], Sasao presented an algorithm to reduce the number of variables for multiple-valued index generation functions by a linear transformation. In [12], Sasao et al. showed a lower bound on the number of variables to represent incompletely specified random index generation functions.

In [1], Astola et al. showed that an upper bound on the number of variables to represent index generation function with weight k is $m = \lceil 2 \log_p k \rceil$ for $p = 2$, and $m = 2\lceil \log_p k \rceil + 1$ for $p \geq 3$, when linear transformations of input variables are used.

VIII. CONCLUSION

In this paper, the author 1) Defined sparse MV functions, and showed their efficient realizations using an EIGU: 2)Derived a formula for the expected number of variables to represent sparse MV functions with given parameters (n, p, q, u): and 3) Generated many random MV functions, and compared experimental results with the estimated values.

ACKNOWLEDGMENTS

This research is partly supported by the Japan Society for the Promotion of Science (JSPS) Grant in Aid for Scientific Research. Comments of reviewers and discussion with Prof. Jon T. Butler improved the presentation.

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," *International Symposium on Multiple-Valued Logic*, (ISMVL-2016), Sapporo, Japan, May 17-19, 2016, pp. 107-112.

[2] C. Halatsis and N. Gaitanis, "Irredundant normal forms and minimal dependence sets of a Boolean functions," *IEEE Trans. on Computers*, vol. C-27, no. 11, Nov. 1978, pp. 1064-1068.

[3] Y. Kambayashi, "Logic design of programmable logic arrays," *IEEE Trans. on Computers*, vol. C-28, no. 9, Sept. l979, pp. 609-617.

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

[5] T. Sasao, "On the number of dependent variables for incompletely specified multiple-valued functions," *International Symposium on Multiple-Valued Logic (ISMVL-2000)*, Portland, OR, USA, pp. 91-97, May, 2000.

[6] T. Sasao, "On the numbers of variables to represent sparse logic functions,"*International Conference on Computer Aided Design (ICCAD-2008)*, pp. 45-51, November, 2008.

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

[8] T. Sasao,"Index generation functions: Recent developments," (invited paper) *International Symposium on Multiple-Valued Logic* (ISMVL-2011), Tuusula, Finland, May 23-25, 2011.

[9] T. Sasao, "Linear decomposition of index generation functions," *17th Asia and South Pacific Design Automation Conference* (ASPDAC-2012), Jan. 30–Feb. 2, 2012, Sydney, Australia, pp. 781-788.

[10] T. Sasao, "Multiple-valued index generation functions: Reduction of variables by linear transformation," *Journal of Multiple-Valued Logic and Soft Computing*, Vol. 21, No. 5-6, pp. 541-559, 2013.

[11] T. Sasao, "Index generation functions: Tutorial," *Journal of Multiple-Valued Logic and Soft Computing*, Vol. 23, No. 3-4, pp. 235-263, 2014.

[12] T. Sasao, Y. Urano and Y. Iguchi, "A lower bound on the number of variables to represent incompletely specified index generation functions,"*International Symposium on Multiple-Valued Logic* (ISMVL-2014), Bremen, Germany, May 19-22, 2014, pp. 7-12.

[13] D. A. Simovici, M. Zimand, and D. Pletea, "Several remarks on index generation functions," *International Symposium on Multiple-Valued Logic*, (ISMVL-2012), Victoria, Canada, May 2012, pp. 179-184.