Row-Shift Decompositions for Classification Functions: Application to Machine Learning Problems

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

***Abstract*—A classification function is a multi-valued function, where the function values for only a fraction of the input combinations are defined. It shows a classification problem in machine learning. This paper shows a method to decompose the function, and to implement the function by a pair of look-up tables and an adder. With this technique, the size of memory to realize the function can be reduced drastically. Experimental results for various benchmark functions for machine learning show the effectiveness of the approach.**

***Index Terms*—linear decomposition, partially defined function, memory-based synthesis, machine learning, data compression.**

I. Introduction

A classification function is a multi-valued function, where the function values for only a fraction of the input combinations are defined. The function shows a classification problem in machine learning. An n-variable logic function f can be directly implemented by a look-up table (LUT). However, the size of the LUT to realize f is proportional to 2^n. Thus, when n is large, to implement f by a single LUT is impractical.

A typical classification function is sparse, i.e., the output values for only the small fraction the input combinations are defined[1]. In such a function, many variables are redundant, and can be eliminated. In fact, the number of variables necessary to represent such a function only depends on the number of specified combinations, which corresponds to the number of training instances.

A variable that can be represented as an EXOR of variables is called a **compound variable**. By using compound variables, we can further reduce the number of variables. Fig. 1.1 shows a circuit to realize the classification function, where L realizes linear functions (i.e., compound variables), while G realizes general functions of p variables, where $p < n$. We assume that G is implemented by a memory. However, resulting functions for G have still many variables, and may be too large to implement by a single LUT.

In this paper, we introduce **row-shift decomposition** shown in Fig. 1.2. In this decomposition, a given function is decomposed as $f(X_1, X_2) = g(h(X_1) + X_2)$, where, $+$ denotes *modulo* addition. This decomposition can realize sparse functions efficiently [13], [14].

The rest of the paper is organized as follows: Section II shows definitions and basic properties of classification functions and linear decompositions; Section III shows a method to realize a classification by a row-shift decomposition. Section IV shows a design method for a classification by a linear decomposition and a row-shift decomposition. Section V shows experimental results using various benchmark functions. Section VI shows the scalability of the method. Section VII shows related works. And, Section VIII summarizes the paper.

II. Definitions and Basic Properties

Definition 2.1: Let n be the number of variables, $B = \{0, 1\}$, $D \subset B^n$, m be the number of classes, and $M = \{1, 2, \ldots, m\}$. Then, a **classification function** is a mapping: $f : D \rightarrow M$.

Example 2.1: Table 2.1 is a **registered vector table** of the classification function with $n = 6$, $m = 2$, and $|D| = 10$. Each vector in the table is a **registered vector**. There are 2^6 possible input combinations, but the function is defined for only 10 input combinations. For the other $64 - 10 = 54$ input combinations, function values are undefined. ■

Next, we show a reduction of variables in classification functions.

Example 2.2: The function f in Table 2.1 can be represented by x_1, x_2, x_3 and x_4. That is, four variables are

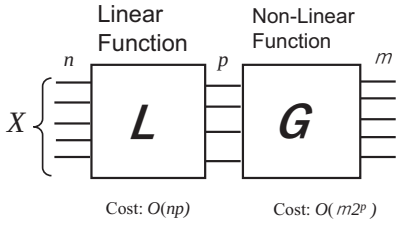

Fig. 1.1. Linear decomposition.

[1]For example, the number of images in the training set in the MNIST 2-class problem that appears in Section V is only 6×10^4, while all possible input combinations is $2^{784} \simeq 10^{236}$.

TABLE 2.1
REGISTERED VECTOR TABLE FOR f.

x_1	x_2	x_3	x_4	x_5	x_6	f
1	1	0	1	1	1	1
1	1	0	0	1	1	1
0	1	1	0	1	1	1
0	1	0	1	0	0	1
0	0	0	0	1	0	1
1	1	1	0	1	1	2
1	0	1	1	1	1	2
1	0	0	0	1	1	2
0	0	1	0	1	0	2
0	0	0	1	0	1	2

sufficient to represent the function. As shown in Table 2.2, all the bit patterns of the first four bits are distinct. ■

TABLE 2.2
f REPRESENTED WITH x_1, x_2, x_3 AND x_4.

x_1	x_2	x_3	x_4	f
0	0	0	0	1
0	1	0	1	1
0	1	1	0	1
1	1	0	0	1
1	1	0	1	1
0	0	0	1	2
0	0	1	0	2
1	0	0	0	2
1	0	1	1	2
1	1	1	0	2

TABLE 2.3
f REPRESENTED WITH y_1, x_2 AND x_3.

y_1	x_2	x_3	x_4	f
0	0	0	0	1
1	1	0	1	1
0	1	1	0	1
1	1	0	0	1
0	1	0	1	1
1	0	0	1	2
0	0	1	0	2
1	0	0	0	2
0	0	1	1	2
1	1	1	0	2

Consider the decomposition of the function shown in Fig. 1.1 [7], where L realizes linear functions, while G realizes general functions. The cost of L is $O(np)$, while the cost of G is $O(m2^p)$. We assume that L is implemented by EXOR gates, while G is implemented by a memory (LUT). When n is large, the cost of L can be neglected. The functions produced by the circuit L in Fig. 1.1 have the form:

$$y_i = a_1x_1 \oplus a_2x_2 \oplus \cdots \oplus a_nx_n,$$

where $a_j \in \{0,1\}$. y_i is called a **compound variable**. $\sum_{i=1}^{n} a_i$ is the **compound degree**. When the compound degree is one, y_i is called a **primitive variable**.

Example 2.3: When x_1 is replaced by $y_1 = x_1 \oplus x_4$, the Table 2.2 is converted to Table 2.3. Note that the resulting function is independent of x_4, and depends on only y_1, x_2 and x_3.

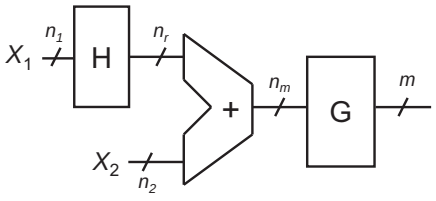

Fig. 1.2. Row-Shift Decomposition.

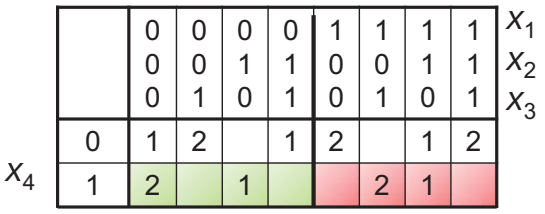

Fig. 2.1. Decomposition chart of Table 2.2.

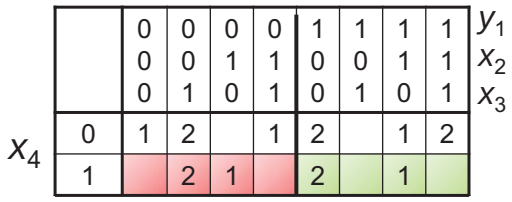

Fig. 2.2. Decomposition chart of Table 2.3.

This is verified by the decomposition charts. Fig. 2.1 corresponds to Table 2.1, and Fig. 2.2 corresponds to Table 2.3. Note that in Fig. 2.1, the column for $(x_1, x_2, x_3) = (0, 0, 0)$ has two different specified values. That is, the column (0,0,0) has a **collision**. This shows that the function depends on x_4. However, in Fig. 2.2, no column has a collision. This shows that the function is independent of x_4.

Also note that, by the linear transformation of the variable $y_1 = x_1 \oplus x_4$, the cells in the red area and the cells in the green area are interchanged. ■

From the above example, we have the following

Theorem 2.1: In a decomposition chart of a classification function, if no column has a collision, then the function is independent of the row variables, and can be represented by only the column variables.

Definition 2.2: A classification function f is **reducible** if f can be represented with fewer variables than the original function by a linear transformation. Otherwise, f is **irreducible**.

As for the number of compound variables p in an irreducible classification function, we have the following result [16].

Theorem 2.2: Let k_i be the number of registered vectors $\vec{a}$ such that $f(\vec{a}) = i$. Then, with a linear transformation, f can be represented by p compound variables, where

$$p \leq \lfloor \log_2(1 + \sum_{(i<j)} k_ik_j) \rfloor.$$

Thus, when the number of registered vectors k is much smaller than the possible number of input combinations 2^n, we can always reduce the number of variables p in Fig. 1.1[2].

[2]This is quite different from the conventional functional decompositions [1], [4], where most functions are undecomposable [10].

TABLE 2.4
REGISTERED VECTOR TABLE FOR f.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	0	0	0	1	0	1
0	0	1	0	1	0	2
0	1	0	1	0	0	3
0	1	1	0	1	1	4
1	0	0	0	1	1	5
1	0	1	1	1	1	6
1	1	0	0	1	1	7
1	1	0	1	1	1	8

TABLE 3.1
TRUTH TABLES FOR FUNCTIONS h AND g.

x_1	x_2	x_3	h
0	0	0	0
0	0	1	2
0	1	0	1
0	1	1	3
1	0	0	5
1	0	1	2
1	1	0	0
1	1	1	0

x_4	x_5	x_6	g
0	0	0	5
0	0	1	6
0	1	0	1
0	1	1	7
1	0	0	2
1	0	1	3
1	1	0	4
1	1	1	8

III. ROW-SHIFT DECOMPOSITION

In this section, we consider a **row-shift decomposition** for classification functions. It decomposes the function f into two subfunctions $f(X_1, X_2) = g(H(X_1), X_2)$.

Example 3.1: Fig. 3.1 is the decomposition chart of the classification function shown in Table 2.4. $X_1 = (x_1, x_2, x_3)$ denotes the rows, while $X_2 = (x_4, x_5, x_6)$ denotes the columns. Note that columns for $X_2 = (x_4, x_5, x_6) = (0, 1, 0)$, $(0, 1, 1)$ and $(1, 1, 1)$ have elements with different specified values. That is, these columns have **collisions**. Thus, this function depends on X_1, and cannot be represented with only column variables X_2.

Next, consider the decomposition chart shown in Fig. 3.2. This decomposition chart is derived from Fig. 3.1 by shifting rows for $X_1 = (x_1, x_2, x_3) = (0, 0, 1)$, $(0, 1, 0)$, $(0, 1, 0)$, $(0, 1, 1)$,$(1, 0, 0)$, and $(1, 0, 1)$. In Fig. 3.2, no column has a collision. Thus, this function can be represented with only the column variables $X_2 = (x_4, x_5, x_6)$.

In Fig. 1.2, the LUT for H stores the **amount of shift bits** for each row specified by X_1, and the LUT for G stores column values of non-zero[3] element after the shift operation $g(h(X_1) + X_2)$, where + denotes modulo 8 addition. In this case, the circuit in Fig. 1.2 realizes the given function f in the form $f(X_1, X_2) = g(h(X_1) + X_2)$, where h and g are shown in Table 3.1. Note that the number of right arrows in Fig. 3.2 corresponds to the values of h in Table 3.1. Both h and g can be implemented with 3-input 3-output LUTs. If the function in Table 2.4 is realized by a single memory, a 6-input 3-output LUT is necessary. ■

In Example 3.1, the function can be decomposed without increasing the number of columns. However, in general, the number of columns must be increased to avoid collisions. In a row-shift decomposition, the decomposition chart for an m class function, at least m columns are necessary to represent the function.

[3]Unspecified value is represented by 0 in the tables.

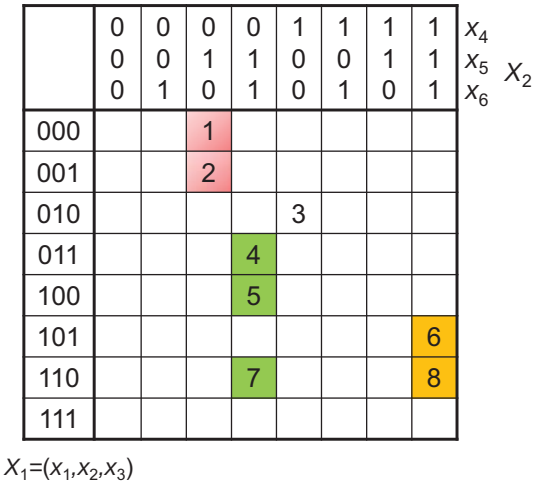

Fig. 3.1. Decomposition Chart for Original function.

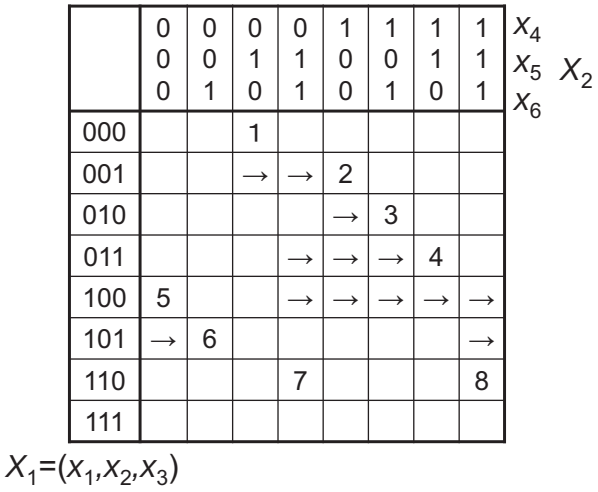

Fig. 3.2. Decomposition Chart after Shift Operation.

In Fig. 3.1, we can find various shifts that assure that **no column has collisions**.

To find a set of shifts, we use the **first-fit method** [19], since it is fast and simple:

Algorithm 3.1: (Obtain the amount of row shift)

1) Count the number of non-zero elements for each row. Reorder the rows in descending order of the number of non-zero elements. Then, perform the following steps.
2) For each row, obtain the minimum shift value $r(i)$ so that no column has a collision with any element in the upper rows.
3) If there is no such shift value, then we assume that the function is undecomposable with the given table size.

TABLE 3.2
REGISTERED VECTOR TABLE FOR f.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	0	0	0	1	0	1
0	0	0	1	0	1	2
0	0	1	0	1	0	3
0	1	0	1	0	0	4
0	1	1	0	1	1	5
1	0	0	0	1	1	6
1	0	1	1	1	1	7
1	1	0	0	1	1	8
1	1	0	1	1	1	9

So, double the table size by adding an extra column variable (i.e., expand horizontally), and go to Step 2.

4) If all the columns satisfy the condition, then stop.

Algorithm 3.1 finds cyclic shifts so that no collision occur in the above rows.

	000	001	010	011	100	101	110	111
000			1			2		
001			3					
010					4			
011				5				
100				6				
101								7
110				8				9
111								

X_1=(x_1,x_2,x_3) X_2 = (x_4,x_5,x_6)

Fig. 3.3. Decomposition chart for 6-variable function.

TABLE 3.3
ROW-SHIFT OPERATION IN THE TABLE FOR G.

Step	0	1	2	3	4	5	6	7	8	9	10
1	0	0	$*1$	0	0	$*2$	0	0	0	0	0
2	0	0	1	$*8$	0	2	0	$*9$	0	0	0
3	0	0	1	8	$*3$	2	0	9	0	0	0
4	0	0	1	8	3	2	$*4$	9	0	0	0
5	0	0	1	8	3	2	4	9	$*5$	0	0
6	0	0	1	8	3	2	4	9	5	$*6$	0
7	0	0	1	8	3	2	4	9	5	6	$*7$

The next example shows the case, where an increase of columns is necessary.

Example 3.2: Consider the classification function f in Table 3.2. Since this is a 6-variable function, the input variables are partitioned into $X_1 = (x_1, x_2, x_3)$ and $X_2 = (x_4, x_5, x_6)$.

Fig. 3.3 is the decomposition chart for f. Note that the rows for $X_1 = (0, 0, 0)$ and $X_1 = (0, 1, 1)$ have two non-zero elements. Other rows have at most one non-zero element. So, the table for G is organized as shown in Table 3.3. Since f takes 9 different values, the size of the table is at least 9.

- In the 1st step, the columns for 2 and 5 are filled with 1 and 2, respectively. Note that in this step, no shift operation is necessary. In Table 3.3, * marks show newly entered elements.
- In the 2nd step, the columns for 3 and 7 are filled with 8 and 9, respectively. Again, in this step, no shift operation is necessary.
- In the 3rd step, the column for 4 is filled with 3. In this step, a two-bit shift is necessary to avoid the collision.
- In the 4th step, the column for 6 is filled with 4. Again, in this step, a two-bit shift is necessary to avoid the collision.
- In the 5th step, the column for 8 is filled with 5. In this step, a five-bit shift is necessary to avoid the collisions.
- In the 6th step, the column for 9 is filled with 6. In this step, a six-bit shift is necessary to avoid the collision.
- In the 7th step, the column for 10 is filled with 7. In this step, a three-bit shift is necessary to avoid the collision.

Table 3.4 shows the number of bits to shift for each row. The bottom row of Table 3.3 shows the content of G. ■

TABLE 3.4
NUMBER OF SHIFT BITS IN H.

x_1	x_2	x_3	h
0	0	0	0
0	0	1	2
0	1	0	2
0	1	1	5
1	0	0	6
1	0	1	3
1	1	0	0
1	1	1	0

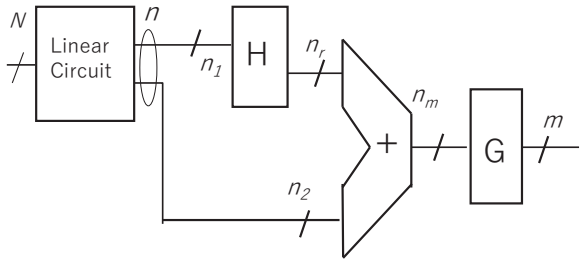

Fig. 4.1. Linear and Row-Shift Decomposed Realization.

IV. COMBINATION OF TWO DECOMPOSITIONS

In this part, we consider realization of classification functions using both linear and row-shift decompositions as shown in Fig. 4.1. Note that this architecture is programmable. Thus, user can easily modify the function. We assume that the output of the function is represented by 1-

hot encoding[4] Then, the number of outputs of G is m. The memory size for the original function is $m2^N$. The memory size for linear decomposed function is $m2^n$. Let $maxbits$ be the maximum number of bits to shift. Then, the size of H is $n_r \cdot 2^{n_1}$, where $n_r = \lceil \log_2 maxbits \rceil$.

Let n_m be the number of output bits of the adder. The total memory size for row-shift decomposed circuit is

$$Total = n_r \cdot 2^{n_1} + m \cdot 2^{n_m},$$

The reduction ratio by the row-shift decomposition is

$$Ratio = \frac{Total}{m2^n}.$$

The selection of the column variables X_1 in the decomposition chart greatly influences the total memory size. We selected the variables so that the number of non-zero elements are distributed uniformly in the decomposition chart. The reason for this heuristic is as follow: Suppose that only a small fraction of rows have many non-zero elements in the decomposition chart. In this case, we need a large memory for G to avoid collisions. On the other hand, if the non-zero elements are distributed uniformly in the decomposition chart, then we can easily find the shift to avoid collisions. In the extreme case, if each row has at most one non-zero elements, the selections of shift values are trivial.

To select the column variables for the decomposition chart, we use the following method. Consider the binary decision tree of the classification function with row variables X_1. Then, the size of a leaf of the decision tree corresponds to the number of non-zero elements in a row. We used a heuristic using **impurity measure** [16] to construct a balanced tree, which made the number of non-zero elements in rows nearly equal in the decomposition chart.

To minimize the total memory size, H and G should have similar sizes. Thus, n_1 should be larger than n_2 to minimize the total memory size. In the experiment, we computed the total memory sizes for various n_1, and selected the value of n_1 that minimized the total memory size.

V. Experimental Results

A. Outline of the Experiment

To investigate the effect of row-shift decomposition, we decomposed various benchmark functions for machine learning. Details of the functions can be found in [16], [17], [20]. Table 5.1 shows the results. The first column shows the function name; the second column shows the number of the variables in the original function[5] N; the third column shows the number of classes m; the fourth column shows k, the number of the registered vectors; the fifth column shows n, the number of variables after linear decomposition; the sixth column shows k_L, the number of distinct registered vectors after linear decomposition; the seventh column shows n_1, the number of row variables; the eighth column shows n_r, the number of variables to show the shift bits; the ninth column shows n_2, the number of column variables; the tenth column shows n_m, the number of inputs to G; the eleventh column shows $ex = n_m - n_2$, the number of extra inputs to G; the last two columns show the total memory size. The column $Total$, shows the total number of bits to implement the function; and the column $Ratio$, shows the reduction ratio by the row-shift decomposition.

The experiment was done on a PC using an Intel Core i9-10900, 2.8 GHz CPU, on Windows 11.

B. Detailed Result for MNIST 2-Class Function.

The original function has $N = 28 \times 28 = 784$ variables and $m = 2$ outputs [3], [16] . The number of distinct registered vectors is $k = 59981$.

After linear decomposition, the number of variables is reduced to $n = 24$, and the number of distinct registered vectors is reduced to $k_L = 52823$. Since $n = 24$, the input variables were partitioned into X_1 and X_2, where each block contains 12 variables. Thus, the numbers of rows and columns are both $2^{12} = 4096$. The number of rows with non-zero elements is 4036, and the number of columns with non-zero elements is 4095. The maximum number of non-zero elements in a row is 67. After applying Algorithm 3.1, we obtained the following results. The maximum number of shift bits is $maxbits = 12770$. Thus, the number of outputs for H is $n_r = \lceil \log_2 maxbits \rceil = 14$. And, the number of inputs to G is also $n_m = 14$. We need $ex = 14 - 12 = 2$ extra variables to represent G.

The size of memory after the linear decomposition is $m2^n = 33554432$. The size of memory for H is $n_r 2^{n_1} = 14 \times 2^{12} = 57344$. The size of memory for G is $m2^{n_m} = 2 \times 2^{14} = 32768$. The total memory size is 57344+32768=90112. Thus, the reduction ratio by the row-shift decomposition is 0.0027. The CPU time to find the row shift decomposition for this function is 0.23 seconds.

C. Detailed Result for Spam Mail Filter

The original function has $N = 128$ variables and $m = 2$ outputs. The number of distinct registered vectors is $k = 20000$. Note that this function was generated randomly, and the numbers of vectors for two classes are the same.

After linear decomposition, the number of variables is reduced to $n = 22$, and the number of distinct registered vector is reduced to $k_L = 19977$.

Since $n = 22$, the input variables were partitioned into X_1 and X_2, where each block contains 11 variables. Thus, the numbers of rows and columns are both $2^{11} = 2048$. The number of rows with non-zero elements is 2023, and the number of columns with non-zero elements is 2048. The maximum number of non-zeros elements in a row is 22. Algorithm 3.1 yielded the following results. The maximum number of shift bits is $maxbits = 3272$. Thus, the number

[4]With 1-hot encoding, we can use **ensemble method** [16] to improve the **generalization ability**.

[5]Since two type of decompositions are used in this section, N denotes the number of variables before linear decomposition, while n denotes the number of variables before row-shift decomposition, which was denoted by p in Fig. 1.1.

TABLE 5.1
RESULTS OF ROW-SHIFT DECOMPOSITION.

Benchmark Data				After Linear Decomp.		After Row-Shift Decomposition						
						H		G			Total	
Function Name	N	m	k	n	k_L	n_1	n_r	n_2	n_m	ex	$Total$	$Ratio$
CHEMICAL-ELEMENTS	60	7	118	9	111	6	6	3	6	3	832	0.1806
CHESS3196	75	2	3196	15	2960	8	10	7	10	3	4608	0.0703
CIFAR32 $\times$ 32	1024	2	9930	19	9521	11	9	8	10	2	20480	0.0195
COMPANIES	30	9	3700	18	3691	10	10	8	10	2	18432	0.0078
CONNECT-4	126	3	67557	22	67334	13	14	9	14	5	163840	0.0130
COUNTRIES	60	6	197	10	192	6	7	4	7	3	1216	0.1979
LETTER-RECOGNITION	256	26	20000	21	16912	13	13	8	13	5	319488	0.0059
MNIST14 $\times$ 14	196	10	58191	25	53672	14	15	11	15	4	573440	0.0017
MNIST28 $\times$ 28	784	10	59981	25	53288	14	15	11	15	4	573440	0.0017
MNIST 2-CLASS	784	2	59984	24	52823	12	14	12	14	2	90112	0.0027
NOBEL-LAUREATES	176	3	632	13	620	7	8	6	8	2	1792	0.0729
POKER HAND	85	10	25010	21	24799	13	11	8	11	3	110592	0.0053
RANDOM4000	30	4	4000	18	3990	11	9	7	9	2	20480	0.0195
SEMEION	256	10	1593	16	1459	9	10	7	10	3	15360	0.0234
SPAM MAIL FILTER	128	2	20000	22	19977	11	12	11	12	1	32768	0.0039
SPLICE	240	3	3174	15	2492	9	9	6	10	4	7680	0.0781

of outputs for H is $n_r = \lceil \log_2 maxbits \rceil = 12$. And, the number of inputs to G is also $n_m = 12$. We need $ex = 12 - 11 = 1$ extra variable to represent G.

The size of memory after linear decomposition is $m2^n = 8388608$. The size of memory for H is $n_r 2^{n_1} = 12 \times 2^{11} = 24576$. The size of memory for G is $m2^{n_m} = 2 \times 2^{12} = 8192$. The total memory size is 24576+8192=32768. Thus, the reduction ratio by the row-shift decomposition is 0.0039.

Algorithm 3.1 found shift values where only one extra variable is necessary to avoid collisions.

D. Observation

- Functions with large k and m tend to require large extra variables ex.
- In Connect-4, when $n_1 = n_2 = 11$, the total memory size is 428032, while when $n_1 = 13$ and $n_2 = 9$, the total memory size is 163840.
- For MNIST14$\times$14, and MNIST28$\times$28, when $n_1 > n_2$. the total memory sizes are reduced.
- Let n_m be the number of inputs to G, then $n_m < \lceil \log_2 k_L \rceil$, in Table 5.1. This result was obtained experimentally. So, it may not be true for other benchmark functions.
- CONNECT-4 took 0.33 seconds to perform row-shift decomposition. It required five extra variables, and took the longest CPU time among the benchmark functions in Table 5.1.
- When n is small (e.g., CHEMICAL_ELEMENTS and COUNTRIES), $Ratio$ are large, while when n is large (e.g., MNIST14$\times$14 and MNIST28$\times$28), $Ratio$ are small. Thus, for the functions with large n, the row-shift decompositions are more effective.

VI. SCALABILITY

We assume that the classification function is given by a registered vector table such as Table 2.1. Let n be the number of variables in an original function, and k be the number of registered vectors. The most time-consuming step is the linear decomposition. Various heuristic algorithms for linear decompositions are known. The algorithm using the difference vectors [16] (**ALG3**) produces very good solutions, but takes $O(nk^2)$ time, so when k is large, it is slow. However, by combining a reduction algorithm for primitive variables (**ALG1**) [16] which takes $O(nk \log k)$ time, and a reduction algorithm **3-MIN** [17] which takes $O(n^3 k \log k)$ time, we can obtain a good solution in a shorter time.

For example, in the case of MNIST 2-class, where $n = 784$ and $k = 59984$. the total computation time can be reduced as shown in Fig. 6.1. First, we apply ALG1 to obtain an $n_1 = 35$-variable solution in 75.7 seconds. Then, we apply ALG3 to obtain an $n_3 = 24$-variable solution in 1085 seconds, which is very time-consuming. However, we can reduce the total CPU time by using 3-MIN as shown in Fig. 6.1. First, we apply ALG1 to obtain an $n_1 = 35$-variable solution. Second, we apply 3-MIN to obtain an $n_2 = 27$ variable solution in 25.1 seconds. And, finally we apply ALG3 to obtain an $n_3 = 24$-variable solution in 318.6 seconds.

To find row-shift decompositions, we used sparse matrix technique: only non-zero elements are stored to represent a decomposition chart. The computation complexity is $O(k^2)$. For example, the computation time for MNIST 2-class function is 0.235 seconds, and is much shorter than the time for the linear decomposition. So it can be neglected.

VII. RELATED WORKS

The row-shift decomposition was introduced for index generation functions [13]. Index generation functions form a special class of classification functions, where $m = k$. In many applications of index generation functions, the output of the function f is 0 when the input mismatches the registered vectors. Thus, an AUX memory [15] is used to verify that the input matches a registered vector. However, in

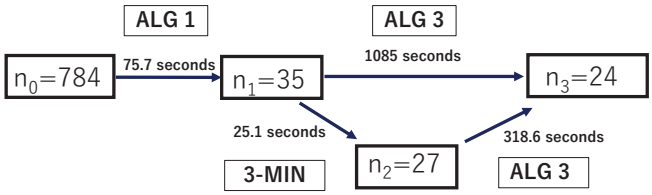

Fig. 6.1. Reduction of CPU time for MNIST 2-Class.

the case of classification functions, the output of the function f can be any value when the input mismatches the registered vectors.

In [13] and [14], only the case where k is small is considered. In such a case, the size of G is equal to $\lceil \log_2 k \rceil 2^{n_2}$. However, in the case of classification functions, we assume that k can be larger and $m < k$, and in many cases, the size of G is greater than $m2^{n_2}$. An explanation as to why the row-shift decomposition does so well is shown in [2].

Various realizations of index generation functions appear in [5], [6], [14], [15]. Application to a virus scanning engine is shown in [8].

VIII. Conclusion and Comments

This paper introduced a row-shift decomposition for classification functions. The contributions of this paper are:

1) Showed the algorithm of row-shift decomposition for classification functions, for the first time.
2) Demonstrated that row-shift decomposition reduced total memory size drastically for machine learning benchmark functions.
3) Showed a heuristic method using impurity measure to reduce total memory size.

Acknowledgments

This work was supported in part by a Grant-in-Aid for Scientific Research of the JSPS. The author thanks Dr. Alan Mishchenko for encouragement, and Prof. Jon T. Butler for discussion. We thank referees whose suggestions improved this paper.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *International Symposium on the Theory of Switching*,pp. 74-116, April 1957.

[2] J. T. Butler and T. Sasao, "Analysis of cyclic row-shift decompositions for index generation functions," *The 21st Workshop on Synthesis And System Integration of Mixed Information Technologies* (SASIMI 2018), pp.65-70.

[3] S. Chatterjee,"Learning and memorization," *International Conference on Machine Learning (ICML 2018)*, Stockholm, Sweden, July 10-15, 2018, pp. 754-762.

[4] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[5] Y. Matsunaga, "Synthesis algorithm of parallel index generation units," *DATE*, Dresden, March 2014, pp. 297.

[6] Y. Matsunaga, "Synthesis algorithm for parallel index generator," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E97-A, No. 12, pp.2451-2458, Dec. 2014.

[7] 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.

[8] H. Nakahara, T. Sasao, and M. Matsuura, "A virus scanning engine using an MPU and an IGU based on row-shift decomposition," *IEICE Transactions on Information and Systems*, Vol. E96-D, No. 8, Aug. 2013, pp. 1667-1675.

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

[10] T. Sasao, "Totally undecomposable functions: applications to efficient multiple-valued decompositions," *ISMVL*, Freiburg, Germany, May 20-23, 1999, pp. 59-65.

[11] T. Sasao, "On the number of variables to represent sparse logic functions," *ICCAD-2008*, San Jose, California, USA, Nov.10-13, 2008, pp. 45-51.

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

[13] T. Sasao, "Row-shift decompositions for index generation functions," *Design, Automation & Test in Europe* (DATE-2012), March 12-16, 2012, Dresden, Germany, pp.1585-1590.

[14] T. Sasao, "Cyclic row-shift decompositions for incompletely specified index generation functions," IWLS-2013.

[15] T. Sasao, *Index Generation Functions*, Morgan & Claypool, Oct. 2019.

[16] T. Sasao, *Classification Functions for Machine Learning and Data Mining*, Springer Nature, Aug. 2023.

[17] T. Sasao, "Iterative linear transformation to reduce compound variables," The 25th Workshop on Synthesis And System Integration of Mixed Information technologies, (SASIMI-2024), March 11, Taipei, Taiwan.

[18] T. Sasao, "Approximate synthesis of classification functions," *International Symposium on Multiple-Valued Logic*, (ISMVL-2024), Brno, Chech Republic, May 29, 2024, pp. 59-64.

[19] R. E. Tarjan, and A. C-C. Yao, "Storing a sparse table," *Communications of the ACM*, Vol.22, No.11, Nov. 1979, pp.606-611.

[20] https://archive.ics.uci.edu/ml/datasets.php