Approximate Synthesis for Classification Functions

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. Many variables in such a function are redundant, and can be eliminated to reduce the circuit size to implement the function. One can further reduce the number of variables by using linear transformation. However, by relaxing the requirement for full accuracy, we can further reduce the number of variables. This paper derives expected number of errors for approximate logic synthesis. Experimental results for various functions are shown.**

***Index Terms*—functional decomposition, linear transformation, minimization of variables, MNIST, partially defined function, approximate logic synthesis, dimension reduction, multi-valued logic, logic synthesis, confidence interval.**

I. Introduction

Classification is one of the fundamental problems in machine learning. For most input vectors, the values of the function are undefined. In such cases, the function can be represented with fewer variables than the original function.

Fig. 1.1 shows the circuit to realize a classification function, where L realizes linear functions, while G realizes general functions of p variables, where $p < n$. We assume that G is implemented by a memory.

To further reduce the memory size, we use **approximate logic synthesis** [7], [14]. That is, to permit a small number of errors to reduce the number of variables. This paper considers the number of errors for approximate logic synthesis. Especially, this paper shows that the expected number of errors can be derived from p, the number of the variables to the memory, and k_i, the number of vectors such that $f(\vec{a}) = i$. The approach is tested for various functions.

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 motivating example illustrating the effect of approximate memory-based synthesis. Section IV derives the expected numbers of errors when the number of inputs to the memory are limited to $p-1$ and $p-2$. Section V shows experimental results; Section VI describes the data sets used in the experiment; 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, $m \geq 2$ be the number of classes, $B = \{0, 1\}$, $D \subset B^n$, and $M = \{0, 1, 2, \ldots, m-1\}$. 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 $|D| = 10$.

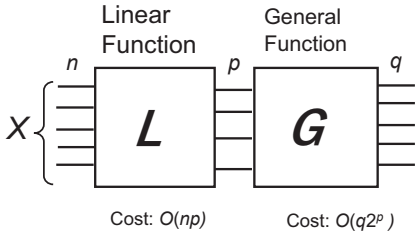

Fig. 1.1. Linear decomposition.

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	0
1	1	0	0	1	1	0
0	1	1	0	1	1	0
0	1	0	1	0	0	0
0	0	0	0	1	0	0
1	1	1	0	1	1	1
1	0	1	1	1	1	1
1	0	0	0	1	1	1
0	0	1	0	1	0	1
0	0	0	1	0	1	1

Each vector in the table is a **registered vector**. Since $n = 6$, there are 2^6 possible input combinations, but the function is defined for only 10 combinations. For the other $64 - 10 = 54$ 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 sufficient to represent the function. As shown in Table 2.2, all the bit patterns of the first four bits are distinct. A similar statement is not true of any set of three variables. However, in Table 2.3, we show a method to reduce the number of variables to three. ■

Consider the decomposition of the function shown in Fig. 1.1 [6], where L contains a linear function, while G contains a general function. The cost of L is $O(np)$, while the cost of G is $O(q2^p)$. We assume that L is implemented by a circuit consisting of 2-input EXOR gates, while G is implemented by a memory (look-up table). 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:

$$y_i = a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n,$$

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

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

TABLE 2.3
f REPRESENTED WITH y_1, y_2 AND y_3.

y_1	y_2	y_3	f	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	0	1	0	*
0	0	0	0	
1	1	1	1	
0	1	0	1	
0	0	1	1	
0	1	0	1	*
0	0	1	1	*

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

Example 2.3: When we use the linear transformation:

$$\begin{aligned} y_1 &= x_2, \\ y_2 &= x_3, \\ y_3 &= x_1 \oplus x_4, \end{aligned}$$

the function in Table 2.2 can be represented by only three variables as shown in Table 2.3. * denotes duplicated vectors. ■

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**.

The function in Table 2.3 is irreducible. That is, we cannot reduce the number of variables by linear transformation.

As for the number of compound variables p in an irreducible classification function, we have the following result [13].

Theorem 2.1: 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

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

In an approximate logic synthesis, a small number of errors are permitted to reduce the circuit size. The error ratio shows how much error occurs when some of input variables are eliminated.

Definition 2.3: The **error ratio** of the classification function f is

$$ER(f) = \frac{N\_Errors}{k},$$

where $k = \sum_{i=0}^{m-1} k_i$ is the total number of the registered vectors, and N_Errors is the number of errors when the registered vectors are applied to f.

III. MOTIVATING EXAMPLE

Example 3.1: In the MNIST [25] handwritten digit recognition data set, consider the 784-variable two-class function f, where the images are binarized by the most significant bit, and

$f(\vec{a}) = 0$ if $\vec{a}$ corresponds to a digit 0, 1, 2, 3, or 4,
$f(\vec{a}) = 1$ if $\vec{a}$ corresponds to a digit 5, 6, 7, 8, or 9.

TABLE 3.1
NUMBER OF VARIABLES VS NUMBER OF ERRORS IN MNIST-2CLASS FUNCTION.

Number of Variables	Number of Errors	Error Ratio
$p - 4 = 20$	524	9.92×10^{-3}
$p - 3 = 21$	268	5.07×10^{-3}
$p - 2 = 22$	117	2.21×10^{-3}
$p - 1 = 23$	39	7.38×10^{-4}
$p = 24$	0	0

By using a linear transformation, we have an irreducible function with $p = 24$ variables. Thus, the circuit requires a linear circuit and a memory with 24 inputs. After removing duplicated vectors, we have $k_0 = 25021$, $k_1 = 27802$, and $k = k_1 + k_2 = 52823$.

Table 3.1 shows the relation among the number of variables, the number of errors, and the error ratio. As expected, with the increase of the variables, the number of errors decreased. The method to compute the number of errors will be illustrated in Example 4.1. When $p-4 = 20$ variables are used to represent f, the number of errors is 524, and the error ratio is 9.92×10^{-3}. When $p - 1 = 23$ variables are used to represent f, the number of errors is 39, and the error ratio is 7.38×10^{-4}. When $p - 4 = 20$ variables are used to represent f, the size of the memory can be reduced to $1/16$ of the exact (100% accurate) realization. ■

In the next section, we derive the expected number of errors and the variance from p, k_0, and k_1.

IV. EXPECTED NUMBER OF ERRORS

In this part, we derive the expected number of errors, when the number of variables to the memory is less than p, where p is the number of the variables of the irreducible classification function. To derive the expected value using statistics, we need the following:

Definition 4.1: In probability theory and statistics, a collection of random variables is **independent and identically distributed** (**IID**)[22], [13] if each random variable has the same probability distribution as the others and all are mutually independent.

IID is often used in machine learning [2]. To derive the expected number of errors, we assume that the values of the truth table of the function are IID[1]

Theorem 4.1: In a p-variable irreducible classification function f, if the values of functions are IID, then the expected number of errors in an approximate logic synthesis for f with $p - 1$ variables is

$$E_1(f) = \frac{\sum_{(i<j)} k_i k_j}{2^p},$$

where k_i is the number of registered vectors $\vec{a}$ such that $f(\vec{a}) = i$.

[1]Unfortunately, for most functions used for experiments, this assumption do not hold.

(Proof) Assume that x_p is not used to represent the function f. In this case, an error occurs when both of $f(a_1, a_2, \ldots, a_{p-1}, 0)$ and $f(a_1, a_2, \ldots, a_{p-1}, 1)$ are specified, and their values are different, where $a_i \in \{0, 1\}$.

Let A be the probability of an error. Then we have

$$A = \sum_{(i \neq j)} \alpha_i \alpha_j,$$

where $\alpha_i = \frac{k_i}{2^p}$. The expected number of errors is

$$E_1(f) = 2^{p-1} \times A = \frac{\sum_{(i<j)} k_i k_j}{2^p},$$

where we used the relation

$$\sum_{(i \neq j)} \alpha_i \alpha_j = 2 \sum_{(i<j)} \alpha_i \alpha_j.$$

□

Theorem 4.2: In a p-variable irreducible classification function f, if the values of functions are IID, then the expected number of errors in an approximate logic synthesis for f with $p-2$ variables is

$$E_2(f) \simeq \frac{3 \sum_{(i<j)} k_i k_j}{2^p},$$

where k_i is the number of registered vectors $\vec{a}$ such that $f(\vec{a}) = i$.

(Proof) Assume that x_{p-1} and x_p are not used to represent the function f. In this case, an error occurs when at least two of

$$f(a_1, a_2, \ldots, a_{p-2}, 0, 0), \qquad f(a_1, a_2, \ldots, a_{p-2}, 0, 1),$$
$$f(a_1, a_2, \ldots, a_{p-2}, 1, 0), \qquad f(a_1, a_2, \ldots, a_{p-2}, 1, 1).$$

are specified, and their values are different.

We use a similar assumption to the proof for Theorem 4.1. Let B be the probability of an error, Then we have

$$B \simeq 6 \sum_{(i \neq j)} \alpha_i \alpha_j,$$

where $\alpha_i = \frac{k_i}{2^p}$. This can be proved as follows: Since, $k_i \ll 2^p$, α_i is very small. There are four rows in the decomposition chart (x_{p-1}, x_{p-2}): (0,0), (0,1), (1,0), and (1,1). There are $\binom{4}{2} = 6$ ways an error can occur. Two errors occur when the four values are specified, and two of them are 1, and the others are 0. For example, in the column with $(x_1, x_2, x_3) = (1, 1, 1)$ of Fig. 4.2, two errors can occur. However, the probability of the occurrence of more than two inconsistencies is $6\alpha_i^2 \alpha_j^2$ which is very small, and can be neglected.

Hence, the expected number of errors is

$$E_2(f) = 2^{p-2} \times B \simeq \frac{3 \sum_{(i<j)} k_i k_j}{2^p}.$$

□

Theorem 4.3: The variance of errors in an approximate logic synthesis for f with $p-1$ variables is

$$V_1(f) \simeq \frac{\sum_{(i<j)} k_i k_j}{2^p},$$

where k_i is the number of registered vectors $\vec{a}$ such that $f(\vec{a}) = i$.

(Proof) The variance can be obtained as

$$V_1(f) = 2^{p-1} A(1 - A).$$

Since A is very small, $1 - A$ can be approximated by 1. Thus, we have $V_1(f) \simeq E_1(f)$. □

Errors produced by an approximate synthesis follows a **normal distribution** in their frequency of occurrence.

Theorem 4.4: In a normal distribution, let μ be the expected number of events, and let σ^2 be the variance. Then, 95.45% of the cases, the numbers of events are within the **confidence interval** $[\mu - 2\sigma, \mu + 2\sigma]$.

TABLE 4.1
EXAMPLE CLASSIFICATION FUNCTION

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

Example 4.1: Consider the classification function in Table 4.1, where $n = 5$, $m = 2$, and $k_0 = k_1 = 9$. This function is irreducible, that is, we cannot reduce the number of variables by linear decomposition. Consider the **decomposition chart** [1], [8] in Fig. 4.1. $X_1 = (x_1, x_2, x_3, x_4)$ denotes the column variables, while $X_2 = (x_5)$ denotes the row variable. It shows the case when the function is implemented by a memory with four variables: (x_1, x_2, x_3, x_4). Blank elements are *don't cares*. Note that the last two columns have the asterisk marks. They show that there exist inconsistencies when the input values (x_1, x_2, x_3, x_4) are $(1, 1, 1, 0)$ and $(1, 1, 1, 1)$. This means that an error can occur when the input values are (1,1,1,0) or (1,1,1,1). Other columns are consistent. In this case, the number of errors is two.

Consider the decomposition chart in Fig. 4.2. $X_1 = (x_1, x_2, x_3)$ denotes the column variables, while $X_2 = (x_4, x_5)$ denotes the row variables. It shows the case when the function is implemented by a memory with three variables: (x_1, x_2, x_3). In this case, three columns with asterisk marks have an inconsistency. This means that an error occurs when the input values (x_1, x_2, x_3) are (0,0,1), (1,1,0), or (1,1,1). Other columns are consistent. In this case, the number of errors is three. ■

V. EXPERIMENTAL RESULTS

To investigate the number of errors in approximate synthesis, we decomposed various functions. Details of the functions

														*	*	
	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	x_1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	x_2
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	x_3
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	x_4
0		0	1	0	0		1		1				0		1	0	
1	0					0			1	1		1	0	1	0	1	
x_5																	

Fig. 4.1. Example decomposition chart for f showing two errors when (x_1, x_2, x_3, x_4) are used as inputs.

		*					*	*	
	0	0	0	0	1	1	1	1	x_1
	0	0	1	1	0	0	1	1	x_2
	0	1	0	1	0	1	0	1	x_3
00		1	0	1	1		0	1	
01	0				1		0	0	
10	0	0						0	
11			0		1	1	1	1	
x_4x_5									

Fig. 4.2. Example decomposition chart for f showing three errors when (x_1, x_2, x_3) are used as inputs.

TABLE 5.1
NUMBER OF ERRORS FOR APPROXIMATE SYNTHESIS

Function Data				Comp. Vari.	Number of Errors			
					Experimental		Expected	
Function Name	n	m	k	p	$p-1$	$p-2$	$p-1$	$p-2$
4350WORDS	75	14	4,350	18	40	87	23.2	69.5
CHESS3196	75	2	3,196	15	13	36	66.7	200.2
CIFAR32 × 32	1024	2	9,930	19	39	120	43.2	129.6
COMPANIES	30	9	3,700	18	13	40	16.5	49.6
CONNECT-4	126	3	67,557	22	357	974	269.0	806.9
FASHION-MNIST	784	10	59,954	25	74	189	44.5	133.4
LETTER-RECOGNITION	256	26	20,000	21	58	204	65.5	196.6
MNIST-2CLASS	784	2	59,984	24	39	117	41.5	124.4
MNIST14 × 14	196	10	58,191	25	44	129	38.5	115.5
MNIST28 × 28	784	10	59,981	25	51	127	37.9	113.8
POKER HAND	85	10	25,010	21	51	243	83.4	250.2
RANDOM4000	30	4	4,000	18	19	59	22.8	68.3
SPAM MAIL FILTER	128	2	20,000	22	17	51	23.8	71.3
SPLICE	240	3	3,174	15	19	49	56.0	168.0

TABLE 5.2
NUMBER OF ERRORS WHEN VARIABLE x_i IS ELIMINATED.

i	Errors	i	Errors
1	45	13	42
2	50	14	57
3	59	15	40
4	68	16	61
5	67	17	51
6	48	18	60
7	49	19	57
8	53	20	64
9	50	21	76
10	80	22	60
11	47	23	67
12	39	24	56

are shown in Section VI. Table 5.1 shows the results. The first column shows the function name; the second column shows the number of binary variables n; the third column shows the number of classes m; the fourth column shows k, the total number of the registered vectors; the fifth column shows p, the number of compound variables in an irreducible function, which was obtained by Algorithm 6.1.1 in p.54 of [13].

The sixth column shows the number of errors when the number of variables is limited to $p-1$; the seventh column shows the number of errors when the number of variables is limited to $p-2$; the eighth column shows the expected number of errors when the number of variables is limited to $p-1$ derived by Theorem 4.1; the last column shows the expected number of errors when the number of variables is limited to $p-2$ derived by Theorem 4.2.

When the number of the variables is limited to $p-1$, the size of the memory is reduced to a half of the exact (100% accurate) realization. Also, when the number of the variables is limited to $p-2$, the size of the memory is reduced to a quarter of the exact realization. From Table 5.1, we can observe that the number of errors for $p-2$ is approximately three times that for $p-1$. Note that the number of errors depends on the variable x_i to eliminate. In Table 5.1, we used a heuristic method to find a variable to eliminate.

Example 5.1: Consider MNIST 2-Class function that appeared in Example 3.1. In this case $p = 24$. After removing duplicated vectors, we have $k_0 = 25021$ and $k_1 = 27802$. Table 5.2 shows the distribution of the number of errors, when variable x_i is eliminated for $1 \leq i \leq 12$. In this case, the algorithm found the variable x_{12} that causes the number of errors to be minimum. The average number of errors is 56.08, while the standard deviation is 10.75. The expected number of errors derived from Theorem 4.1 is $E_1(f) = 41.46$. From Theorem 4.4, the 95.45% confidence interval is $[28.58, 54.34]$. Note that the number of errors obtained by the experiment is 39, which is in the confidence interval. ■

RANDOM4000 and SPAM MAIL FILTER were generated randomly. For these functions, experimental results are in the 95.45% confidence intervals. Especially in the case of SPAM MAIL FILTER, the range of errors resulting from the elimination of a single variable falls within a narrow interval [14, 30].

Other functions are not random. For many of them, experimental results were in the 95.45% confidence intervals. However, for 4350WORDS, CONNECT-4, and F-MNIST, experimental results were larger than $\mu + 2\sigma$. Especially, in the case of CONNECT-4, the range of errors resulting from the elimination of a single variable falls within a large interval [357, 1307]. Thus, the IID model does not apply to this function.

VI. Data Set

In this part, we briefly describe the data set used for the experiment. k is the number of registered vectors prior to linear decomposition[2].

A. 4350WORDS

This function shows **parts of speech**[3] of English words. The original list [28] contained 5000 words. Upper case letters were converted into lower case letters. Only the fields for word and the part of speech were used. Some words have more than one part of speech, but only one is selected. For example, the parts of speech of 'access' are *n* (noun) and *v* (verb), but we selected only *n*. In this way, we removed duplicated words, and had a list of $k = 4350$ unique words. Most words consist of at most 14 characters. Only the first 15 characters were used. Each character was represented by the **minimum-length code** (5 bits). Thus, the number of variables is $n = 5 \times 15 = 75$.

B. CHESS3196

This function has 36 variables showing a board-descriptions for the chess endgame, and the output showing the class: "win" or "nowin" [21]. All the variables are binary, except for two variables; one takes 3 values, and the other takes 4 values. All the variables were represented by a **1-hot code**[4], and we had a function with $n = 75$, $m = 2$, and $k = 3196$.

C. CIFAR32×32

CIFAR-10 consists of colored images [26] of 32×32 pixels. Only the blue component of the training set was binarized using the most significant bits. Also, only the first two classes were used: airplane and automobile. In this way, we had a classification function with $n = 32 \times 32 = 1024$, $m = 2$, and $k = 9930$.

D. COMPANIES

This function maps the telephone numbers of $k = 3700$ Japanese companies [23] into the stock exchange. There are $m = 9$ different stock exchanges. 1) Tokyo 1st; 2) Tokyo 2nd; 3) Tokyo Mothers; 4) Sapporo; 5) Nagoya; 6) Fukuoka; 7) JASDAQ; 8) REIT; and 9) Foreign. The telephone numbers are represented by 9-digit decimal numbers. These numbers were converted into binary numbers of 30-bits. Thus, the number of inputs is $n = 30$.

E. CONNECT-4

This function classifies the positions of the game CONNECT-4 into three categories: (1) player X win, (2) player Y wins, or (3) draw [16]. Each variable takes 3 values: empty, X, or Y. There are $6 \times 7 = 42$ positions. Each variable is represented by a 1-hot code. Thus, in this function, $n = 3 \times 42 = 126$, $m = 3$ and $k = 67557$. We assume that each player plays optimally.

F. FASHION-MNIST

This data set shows sample images of Zalando's [24] fashion items. Each image is a bit map of $n = 28 \times 28 = 784$ pixels. Originally each pixel was represented by an 8-bit number (i.e., 256-valued grayscale). To reduce the size of data, each pixel was represented by a binary bit, where threshold=32. Similar to the case of MNIST, the data set consists of about $60,000$ training images, about 6000 images for each article. There are $m = 10$ items. In this case, $n = 784$, $m = 10$, and $k = 59954$.

G. LETTER-RECOGNITION

This function identifies each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet [17]. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of $k = 20,000$ unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. To represent the attributes, a 1-hot code was used, so the number of the binary variables is $n = 16 \times 16 = 256$. The number of the classes is $m = 26$.

H. MNIST-2CLASS

This function appeared in Example 3.1 [3], [13].

I. MNIST 14×14

This data is derived from MNIST 28×28. These bitmaps are divided into non-overlapping blocks of 2×2 and the maximum value of pixels are stored in each block. Finally, the values of pixels were binarized. In this case, $n = 14 \times 14 = 196$, $m = 10$, and $k = 58191$ [27].

J. MNIST 28×28

MNIST [25] is a data set of handwritten digits. The training set consist of 6×10^4 images. Each image is a bit map of 28×28 pixels. Originally, each pixel is represented by an 8-bit number (i.e., a grayscale image of 256 values), but we converted it into binary number by setting the threshold 96[5]. In this case, $n = 28 \times 28 = 784$, $m = 10$, and $k = 59981$. In this process, we removed duplicated data.

K. POKER HAND

This function produces a poker hand. Each hand consists of five playing cards drawn from a standard deck of 52. Each card is described by two attributes (suit and rank). Thus, the number of variables is $N = 10$: five of which take four values (Hearts, Spades, Diamonds, Clubs); and five of which take 13 values (Ace, 2, 3, ... , Queen, King). Thus, the total number of binary variables is $n = (5 \times 4) + (5 \times 13) = 85$. The output denotes a poker hand, and the number of classes is $m = 10$: (1) Nothing, (2) One pair, (3) Two pairs, (4) Three of a kind, (5) Straight, (6) Flush, (7) Full house, (8) Four of a kind, (9) Straight flush, and (10) Royal flush. [20].

[2]A linear decomposition reduces the number of variables, and often produces duplicated registered vectors.

[3]One of the grammatical groups, such as article, noun, pronoun, verb, auxiliary verb, adjective, adverb, preposition, conjunction, and interjection, into which words are divided according to their use.

[4]Experimental results show that when the value of a variable is small, encoding by a 1-hot code results in fewer variables [12].

[5]We found this threshold by experiments.

L. RANDOM4000

This function was generated from 4000 distinct random vectors of 30 bits. They are partitioned into $m = 4$ sets, each consists of 1000 vectors. From this, a classification function with $n = 30$ inputs, four outputs, and $k = 4000$ registered vectors was generated.

M. SPAM MAIL FILTER

This function shows a SPAM E-mail filter with the following specification:

- The number of IP addresses in the white list: 10000.
- The number of IP addresses in the black list: 10000.
- The number of bits to represent IP addresses: 128.
- The function takes two values: spam ($f = 1$) or ($f = 0$) not.
- The bit patterns of IP addresses are random.

N. SPLICE

This function classifies the gene sequences (DNA) into three classes: (1) donor, (2) acceptor, (3) neither. Originally, it contained 3191 instances, but 16 ambiguous instances were removed. Each variable takes 4 values: (1) A, (2) G, (3) T, and (4) C. Finally, conflicting data was removed to produce $k = 3,174$ instances. In this case, we use $n = 4 \times 60 = 240$ variables to represent the function. [19].

VII. Related Works

An approximate memory-based synthesis for **index generation functions** [10] was considered in [5]. Note that an index generation function is a special case of a classification function, where $k = m$. This method tries to eliminate a set of registered vectors to reduce the number of variables. Experimental results for up to 8-variable irreducible index generation functions are shown.

VIII. Conclusion and Comments

This paper considered an approximate logic synthesis for classification functions. Let p be the number of variables as the result of linear decomposition. To further reduce the memory size, an approximate logic synthesis is considered. We derived a method to estimate the number of errors, when the number of variables is $p - 1$ and $p - 2$. The main findings are as follows:

1) With p, the number of variables in an irreducible classification function, and k_i the number of registered vectors $\vec{a}$ such that $f(\vec{a}) = i$, we can estimate the number of errors for approximate logic synthesis.
2) The expected number of errors for the $(p - 2)$-variable realization is about three times of that for the $(p - 1)$-variable realization.
3) For most functions, the experimental results are within the 95.45% confidence intervals.
4) By sacrificing a little accuracy, the circuit size can be greatly reduced.

This method is useful to find a tradeoff between the number of errors and the circuit size.

In this paper, we assumed that the part G in Fig. 1.1 is implemented by a memory. However, a multi-level LUT circuit can be also used to implement G.

Acknowledgments

This work was supported in part by a Grant-in-Aid for Scientific Research of the JSPS. The author thanks Dr. Alan Mishchenko and Prof. Jon T. Butler for discussion. Reviewers's comments improved the presentation of the paper.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *International Symposium on the Theory of Switching*, pp. 74-116, April 1957.
[2] C. M. Bishop, *Pattern Recognition and Machine Learning,* Springer, 2006.
[3] S. Chatterjee,"Learning and memorization," *International Conference on Machine Learning (ICML 2018)*, 2018, pp. 754-762.
[4] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, " Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278-2324, November 1998.
[5] T. Mazurkiewicz, "Approximate memory-based logic synthesis of index generation functions using linear decomposition," *ISMVL*, 2022, online, pp. 145-150.
[6] E. I. Nechiporuk, "On the synthesis of networks using linear transformations of variables," *Dokl. AN SSSR*, vol. 123, no. 4, December 1958, pp. 610-612 (in Russian).
[7] S. Rai et al., "Logic synthesis meets machine learning: Trading exactness for generalization," *DATE2021*, pp. 1026-1031.
[8] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.
[9] T. Sasao, *Memory-Based Logic Synthesis*, Springer, 2011.
[10] T. Sasao, *Index Generation Functions*, Morgan & Claypool, October 2019.
[11] T. Sasao, "Reduction methods of variables for large-scale classification functions," *International Workshop on Logic and Synthesis* (IWLS-2020), July 27-29, 2020, pp. 82-87.
[12] T. Sasao, and J. T. Butler, "Linear Decompositions for multi-valued input classification functions," *ISMVL-2021*. May 2021, pp. 13-18.
[13] T. Sasao, *Classification Functions for Machine Learning and Data Mining*, Springer Nature, August 2023.
[14] I. Scarabottolo, et. al., "Approximate logic synthesis: A survey," *Proceedings of the IEEE*, Vol. 108, No. 12, pp. 2195-2213, December 2020.
[15] https://archive.ics.uci.edu/datasets
[16] https://archive.ics.uci.edu/dataset/26/connect+4
[17] https://archive.ics.uci.edu/dataset/59/letter+recognition
[18] https://archive.ics.uci.edu/dataset/80/optical+recognition+of +handwritten+digits
[19] https://archive.ics.uci.edu/dataset/69/molecular+biology+splice +junction+gene+sequences
[20] https://archive.ics.uci.edu/dataset/158/poker+hand
[21] https://archive.ics.uci.edu/ml/datasets/Chess+
[22] https://en.wikipedia.org/wiki/Independent_and_identically _distributed_random_variables
[23] https://str.toyokeizai.net/magazine/shikiho_cd/
[24] https://tech.zalando.com
[25] http://yann.lecun.com/exdb/mnist/
[26] https://www.cs.toronto.edu/ kriz/cifar.html
[27] https://www.cs.waikato.ac.nz/ml/index.html
[28] http://www.wordfrequency.info