Hindawi Publishing Corporation
Journal of Discrete Mathematics
Volume 2013, Article ID 270424, 12 pages
http://dx.doi.org/10.1155/2013/270424

Research Article

Classification of Boolean Functions Where Affine Functions Are Uniformly Distributed

Ranjeet Kumar Rout,[1] Pabitra Pal Choudhury,[1] and Sudhakar Sahoo[2]

[1] *Applied Statistics Unit, Indian Statistical Institute, Kolkata 700108, India*
[2] *Institute of Mathematics and Applications, Bhubaneswar 751003, India*

Correspondence should be addressed to Ranjeet Kumar Rout; ranjeetkumarrout@gmail.com

Received 17 May 2013; Revised 22 August 2013; Accepted 11 September 2013

Academic Editor: Pantelimon Stănică

The present paper on classification of n-variable Boolean functions highlights the process of classification in a coherent way such that each class contains a single affine Boolean function. Two unique and different methods have been devised for this classification. The first one is a recursive procedure that uses the Cartesian product of sets starting from the set of one variable Boolean functions. In the second method, the classification is done by changing some predefined bit positions with respect to the affine function belonging to that class. The bit positions which are changing also provide us information concerning the size and symmetry properties of the classes/subclasses in such a way that the members of classes/subclasses satisfy certain similar properties.

1. Introduction

Classification of non-linear Boolean functions has been a long standing problem in the field of theoretical computer science. A systematic classification of Boolean functions with n-variable having a representative in each class is a welcomed step in this area of study. It has been very accurately considered as vital and meaningful because of two important well-defined reasons: (a) equivalent functions in each class possess similar properties and (b) the number of representatives in each class is much less than that of Boolean functions.

Earlier, when two Boolean functions of n-variable differ only by permutation or complementation of their variables, they fall into equivalence classes. The formula for counting the number of such equivalence classes is given in [1]. Further, it has also been elaborated in [2] about the procedures of selection of a *representative assembly*, with one member from each equivalence class. In [3], the linear group and the affine Boolean function group of transformations have been defined and an algorithm has been proposed for counting the number of classes under both groups. The classification of the set of n-input functions is specifically based on three criteria: the number of functions, the number of P classes, and the number of NPN classes, which are first introduced in [4]. Classification of the affine equivalence classes of cosets of the first order Reed-Muller code with respect to cryptographic properties such as correlation immunity, resiliency, and propagation characteristics has been discussed in [5–8]. Heuristic design of cryptographically strong balanced Boolean function was envisaged in [9]. In [10], three variable Boolean functions in the name of 3-neighborhood cellular automata rules have been classified on the basis of hamming distance with respect to linear rules. The characterization of 3-variable non-linear Boolean functions has been undertaken in three different ways, by Boolean derivatives, by deviant states, and by matrices as elaborated in the papers [10–12], respectively.

In this paper, two methods have been proposed for generating equivalence classes of Boolean functions with a specific objective in our mind that, in each class, exactly one affine Boolean function is present. The first method is a recursive approach to classify n-variable Boolean functions starting from 1-variable to higher variables. In the second method, the classification is done through changing some variable bit positions with respect to the affine function belonging to that class.

In the following sections, the paper is organized in a precise methodical manner. In Section 2, the literature of Boolean functions of different variables relevant to our work is reviewed. In Section 3, the method of recursive classification of n-variable Boolean functions is introduced and the properties of these classes are discussed. Based on these properties another efficient method has also been proposed for generating the same classes of n-variable Boolean functions. In Section 4, we have studied the behavior of those classes by using different binary operations such as Hamming distance (HD), XOR operation, and Carry value transformation (CVT) [13]. Section 5 deals with concluding remarks emphasizing the key factors of the entire analysis.

2. Relevant Review

An n-variable Boolean function f is a mapping from the set of all possible n-bit strings $\{0,1\}^n$ into $\{0,1\}$. The number of different n-variable Boolean functions is 2^{2^n}, where each function can be represented by a truth table output as a binary string of length 2^n. The decimal equivalent of the binary string starting from bottom to top (least significant bit) in the truth table is called the rule number of that function [14]. The complement of f is denoted as $\overline{f}$.

A Boolean function with algebraic expression, where the degree is at most one is called an affine Boolean function. The general form for n-variable affine function is

$$f_{\text{affine}}(x_1, x_2, x_3, \dots, x_n) = k_n x_n \oplus k_{n-1} x_{n-1} \oplus \cdots \oplus k_2 x_2 \oplus k_1 x_1 \oplus k_0, \quad (1)$$

where the coefficients are either zero or one.

If the constant term k_0 of an affine function is zero then the function is called a *linear Boolean function*. Thus, affine Boolean functions are either linear Boolean functions or their complements. The number of different n-variable affine Boolean functions is 2^{n+1} out of which 2^n are linear. As an example, the 16 affine Boolean functions in 3-variables are 0, 60, 90, 102, 150, 170, 204, 240, 15, 51, 85, 105, 153, 165, 195, and 255 out of which the first eight are linear and the remaining Boolean functions are their corresponding complements [3].

The concatenation of the Boolean function f with itself and the concatenation of f with its complement $\overline{f}$ are denoted as ff and $f\overline{f}$, respectively. For example,

$$\text{if } f = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \text{then } ff = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad f\overline{f} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}. \quad (2)$$

Note that if f is a Boolean function of n-variable, then ff and $f\overline{f}$ are Boolean functions of $(n+1)$-variable.

Theorem 1. *f is linear if and only if ff and $f\overline{f}$ are linear.*

Apart from the above concatenations as stated in Theorem 1, all other concatenations give non-linear Boolean functions [15].

Corollary 2. *f is an affine Boolean function if and only if ff, $f\overline{f}$, $\overline{f}f$, and $\overline{f}\,\overline{f}$ are affine Boolean functions.*

Proof. The proof of the corollary easily follows from Theorem 1 as affine Boolean functions are either linear Boolean functions or their complements. □

3. Proposed Methods for Classification of Boolean Functions

In this section, two different methods have been proposed to classify the set of all possible n-variable Boolean functions such that each class is of equal cardinality and contains only a single affine function.

3.1. A Recursive Procedure to Classify n-Variable Boolean Functions. Let $S_1 = \{\{00\},\{10\},\{11\},\{01\}\}$ be a set of all 1-variable Boolean functions. Here all the Boolean functions are affine. Let $S_1' = \{\{00\},\{10\}\}$ be a set containing all linear Boolean functions of 1-variable, and $S_1'' = \{\{11\},\{01\}\}$ is the complement of the set S_1'. The Cartesian product of the sets S_1 with S_1' and S_1'' is defined successively as follows:

$$\begin{aligned} S_1 \times S_1' &= \{\{\mathbf{0000}, 0010\}, \{1000, \mathbf{1010}\}, \\ &\quad \{\mathbf{1100}, 1110\}, \{0100, \mathbf{0110}\}\}, \\ S_1 \times S_1'' &= \{\{\mathbf{0011}, 0001\}, \{1011, \mathbf{1001}\}, \\ &\quad \{\mathbf{1111}, 1101\}, \{0111, \mathbf{0101}\}\}. \end{aligned} \quad (3)$$

Note that, S_1 contains four classes each containing a 1-variable Boolean functions whereas, the set $(S_1 \times S_1') \cup (S_1 \times S_1'')$ contains eight disjoint classes of all 2-variable Boolean functions. Here, each class contains exactly one 2-variable affine Boolean function as highlighted above in (3). This process is repeated for the next higher variable, using the recursive formula of the following.

(i) Base case: (for $n = 1$)

$$\begin{aligned} S_1' &= \{\{00\},\{10\}\}, \qquad S_1'' = \{\{11\},\{01\}\}, \\ S_1 &= \left(S_1' \cup S_1''\right) = \{\{00\},\{10\},\{11\},\{01\}\}. \end{aligned} \quad (4)$$

(ii) Recursion: (for $n \geq 2$)

$$\begin{aligned} S_n' &= \left(S_{n-1} \times S_{n-1}'\right), \qquad S_n'' = \left(S_{n-1} \times S_{n-1}''\right), \\ S_n &= \left(S_{n-1}' \cup S_{n-1}''\right), \end{aligned} \quad (5)$$

where S_n contains the classes of all n-variable Boolean functions, where each class contains exactly one n-variable affine function. Here both the sets S_n' and S_n'' are complement to each other.

Theorem 3. *The recursive procedure of* (4) *and* (5)*, when repeated up to* $(n-1)$ *times, classifies the set of all n-variable Boolean functions into* 2^{n+1} *number of disjoint classes. such that each class contains exactly one n-variable affine Boolean function along with some n-variable non-linear Boolean functions.*

Proof. The result follows because of the fact that $(S_{n-1}\times S'_{n-1})\cup(S_{n-1}\times S'_{n-1}) = S_{n-1}\times(S'_{n-1}\cup S''_{n-1}) = S_{n-1}\times S_{n-1} = S_n$ and $(S_{n-1}\times S'_{n-1})\cap(S_{n-1}\times S''_{n-1}) = S_{n-1}\times(S'_{n-1}\cap S''_{n-1}) = S_{n-1}\times\phi = \phi$. And the property that each class contains exactly one n-variable affine Boolean function can be ascertained on using Corollary 2 of Section 2. □

Illustration (from 2-variable classes to 3-variable classes). From (4) and (5) the set

$$S_2 = \left\{\begin{matrix}\{\mathbf{0000},0010\},\{1000,\mathbf{1010}\},\{\mathbf{1100},1110\},\{0100,\mathbf{0110}\} \\ \{\mathbf{0011},0001\},\{1011,\mathbf{1001}\},\{\mathbf{1111},1101\},\{0111,\mathbf{0101}\}\end{matrix}\right\}, \quad (6)$$

and this set contains the classes of all 2-variable Boolean functions. The set $S'_2 = \{\{\mathbf{0000},0010\},\{1000,\mathbf{1010}\},\{\mathbf{1100},1110\},\{0100,\mathbf{0110}\}\}$ is the first four classes of S_2 and $S''_2 = \{\{\mathbf{0011},0001\},\{1011,\mathbf{1001}\},\{\mathbf{1111},1101\},\{0111,\mathbf{0101}\}\}$ is the set containing the remaining classes of S_2 and complement of the set S'_2. Now, the classes of 3-variables are generated using the formula as $S'_3 = (S_2\times S'_2)$, $S''_3 = (S_2\times S''_2)$, and $S_3 = (S'_3\cup S''_3)$. Some of the class members are shown in the following:

$$S'_3 = \left\{\begin{matrix}\mathbf{00000000}, \\ 00000010, \\ 00001000, \\ 00001010, \\ 00001100, \\ 00001110, \\ 00000100, \\ 00000110, \\ 00100000, \\ 00100010, \\ 00101000, \\ 00101010, \\ 00101100, \\ 00101110, \\ 00100100, \\ 00100110\end{matrix}\quad \text{Class } 2,\ldots,\text{Class } 8\right\},$$

$$S''_3 = \left\{\begin{matrix}00000011, \\ 00000001, \\ 00001011, \\ 00001001, \\ \mathbf{00001111}, \\ 00001101, \\ 00000111, \\ 00000101, \\ 00100011, \\ 00100001, \\ 00101011, \\ 00101001, \\ 00101111, \\ 00101101, \\ 00100111, \\ 00100101\end{matrix}\quad \text{Class } 10,\ldots,\text{Class } 16\right\}. \quad (7)$$

The naming of the classes is given as class 1, class 2, . . ., class 2^{n+1} such that the complement of class k is the class $(2^n + k)$ where $k = 1, 2, 3, \ldots, 2^n$. In (7), only the members of 1 and 13 are shown and other classes of Boolean functions are shown in Appendix A.

Theorem 4. *The number of different classes in the above classification is* 2^{n+1}.

Proof. As each class contains exactly one affine Boolean function, the number of classes of n-variable is the same as the number of affine Boolean functions and equals to 2^{n+1}. □

Theorem 5. *The classes are of equal size and the cardinality of each class is equal to* $2^{2^n-(n+1)}$.

Proof. The equal size of the classes easily follows from the cardinality of the two sets S'_n and S''_n. On using Theorem 4, the cardinality of each class = (total number of n − variable Boolean functions)/(total number of n − variable affine Boolean functions) = $(2^{2^n})/(2^{n+1}) = 2^{2^n-(n+1)}$. □

Theorem 6. *The least significant bit of all the Boolean functions in* S'_n *is 0, whereas in* S''_n *it is 1.*

Proof. When $n = 1$, that is for the base case of the recursion, the least significant bit position of all the Boolean functions in the set S'_1 is 0 and for the set S''_1 it is 1. Therefore, the recursive procedure using the Cartesian product also preserves the same property for the next higher variable. □

Interestingly, the relation defined in the recursive procedure is operating on the set of $(n-1)$-variable Boolean functions, but the partition is obtained in the set of n-variable Boolean functions. Therefore, an equivalence relation must exist on the set of n-variable Boolean functions, which divides the set into disjoint equivalence classes.

Theorem 7. *For each class of n-variable, the length of a Boolean function is* 2^n*, out of which* $(n+1)$ *bits are fixed and the remaining* $(2^n-(n+1))$ *bits are changing with respect to the affine Boolean function of that class. The* $(n+1)$ *bit positions of a Boolean function which are fixed in a class are calculated using the formula* $P_n - 2^k$*, where* $P_n = (2^n + 1)$ *and the values of* $k = 0, 1, 2, \ldots, n$.

Proof (using mathematical induction).

Basis. For $n = 1$, each class contains a single Boolean function of length 2. Hence both the first and second bit positions are fixed and it satisfies the formula $P_1 - 2^k = (2^1 + 1) - 2^k$ for $k = 0$ and 1. So, the bit positions are $3 - 2^0 = 2$ and $3 - 2^1 = 1$. Hence the formula is valid for $n = 1$.

Induction Hypothesis. Assume that the formula is valid for the classes of $(n-1)$-variable Boolean functions, S_{n-1}. From recursive definition, the formula is also valid for all the classes of S'_{n-1} and S''_{n-1}. Thus, by induction hypothesis, the invariant bit positions of a class of S_{n-1} is calculated using the formula as given below:

$$P_{n-1} - 2^k, \quad \text{where } P_{n-1} = 2^{n-1} + 1, \; k = 0, 1, 2, \ldots, n-1. \tag{8}$$

Induction. Here we have to prove that the formula is true for all classes in S_n. According to the recursive formula $S_n = (S'_n \cup S''_n)$ where $S'_n = (S_{n-1} \times S'_{n-1})$ and $S''_n = (S_{n-1} \times S''_{n-1})$. Consider a particular class of S_{n-1} and let it be C_1. The corresponding classes of S'_n which will be generated using $(C_1 \times S'_{n-1})$ must contain the Boolean functions of length 2^n, where the first 2^{n-1} (starting from most significant bit) bit positions are from a single class C_1. And hence by induction hypothesis, n number of bit positions is fixed and satisfies (9). From Theorem 6, the least significant bit position of the remaining string of length 2^{n-1} is 0 for all the members of the classes of S'_n. Therefore, the bit positions of a Boolean function, which are fixed in a class of S'_n is calculated by adding 2^{n-1} to all the numbers generated from (9). Along with this, we have to include the least significant bit position (or the first position) in the formula, which gives $(n+1)$ invariant positions of a class in S_n. Thus for S_n, the formula is calculated as follows:

for $k = 0, 1, 2, \ldots, n-1$,

$$\begin{aligned} \left\{P_{n-1} - 2^k\right\} + 2^{n-1} &= \left\{\left(2^{n-1} + 1\right) - 2^k\right\} + 2^{n-1} \\ &= \left\{2^n + 1\right\} - 2^k = P_n - 2^k; \end{aligned} \tag{9}$$

for $k = n$, the value is 1:

$$1 = \left(2^n + 1\right) - 2^n = P_n - 2^n = P_n - 2^k. \tag{10}$$

So the formula is true for all the values of $k = 0, 1, 2, \ldots, n$. The above formula is also true for all the classes of S_n, as any class in S_n is either generated using the formula $(S_{n-1} \times S'_{n-1})$ or $(S_{n-1} \times S''_{n-1})$. Hence, by the principle of mathematical induction, we conclude that $P_n - 2^k$ is true for all positive integers n.

Illustration. For every 1-variable Boolean function, all the bit positions are fixed and the bit positions are $(2^1 + 1) - 2^0 = 2$ and $(2^1 + 1) - 2^1 = 1$. For every 2-variable Boolean function, three bit positions are fixed and the bit positions are $(2^2 + 1) - 2^0 = 4$, $(2^2 + 1) - 2^1 = 3$, and $(2^2 + 1) - 2^2 = 1$. Similarly, for every 3-variable Boolean function, four bit positions are fixed and the bit positions are $(2^3 + 1) - 2^0 = 8$, $(2^3 + 1) - 2^1 = 7$, $(2^3 + 1) - 2^2 = 5$, and $(2^3 + 1) - 2^3 = 1$. For 3-variable functions, all classes and their subclasses are given in Appendix A.

The set of bit positions which are changing in a class can be calculated by subtracting the set of invariant bit positions from the set $\{1, 2, 3, \ldots, 2^n\}$. □

Corollary 8. *The bit positions which are fixed or changing are invariant for all classes with respect to the concerned affine function of that class.*

Proof. The formula given in Theorem 7 is used to calculate the bit positions which are fixed or changing and valid for an arbitrary class. Hence, it is also valid for all classes. □

Table 1: Different subclasses of class 1.

Boolean functions	Decimal value	HD wrt affine Boolean function	No. of Boolean function
00000000 (Affine)	0	0	1
00000010	2	1	4
00100000	32		
00001000	8		
00000100	4		
00100010	34	2	6
00001010	10		
00101000	40		
00001100	12		
00000110	4		
00100100	36		
00101010	42	3	4
00001110	14		
00101100	44		
00001100	12		
00100110	38		
00101110	36	4	1

Using the result of Theorem 7, an equivalence relation has been defined on the set of all possible n-variable Boolean functions by which the same class or classes can be generated without using recursion.

Let f and g be two n-variable Boolean functions, and R is a binary relation on the set of n-variable Boolean functions defined as fRg if and only if "there exist $(n+1)$ bit positions calculated on using Theorem 7 and the calculated bit positions are the same for the functions f and g." Clearly,

(1) $fRf \, \forall f$. So, R is reflexive.

(2) If fRg then gRf. So, R is symmetric.

(3) If fRg and gRh then fRh. So, R is transitive.

Hence, R is an equivalence relation. The next procedure uses the above equivalence relation and can efficiently generate the same class or classes without using the recursive procedure.

3.2. Procedure to Generate the Same Class without Using Recursion. Let f be an n-variable affine Boolean function. Let B be an array which is used to store the bit positions, which are fixed for a Boolean function with respect to the affine function. Array B can be calculated using Algorithm 9. The worst case time complexity of the Algorithm is $O(n)$.

Algorithm 9 (fixed-bit positions (f)).

(1) Initialize $X = 2^n$

(2) for $(i = 0$ to $n)$

(3) {

(4) $B[i] = X$

(5) $X = B[i] - 2^i$

(6) }

(7) return B.

Table 2: XOR values of class 1 of 3-variable Boolean functions.

XOR	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
0	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
2	2	0	6	4	10	8	14	12	34	32	38	36	42	40	46	44
4	4	6	0	2	12	14	8	10	36	38	32	34	44	46	40	42
6	6	4	2	0	14	12	10	8	38	36	34	32	46	44	42	40
8	8	10	12	14	0	2	4	6	40	42	44	46	32	34	36	38
10	10	8	14	12	2	0	6	4	42	40	46	44	34	32	38	36
12	12	14	8	10	4	6	0	2	44	46	40	42	36	38	32	34
14	14	12	10	8	6	4	2	0	46	44	42	40	38	36	34	32
32	32	34	36	38	40	42	44	46	0	2	4	6	8	10	12	14
34	34	32	38	36	42	40	46	44	2	0	6	4	10	8	14	12
36	36	38	32	34	44	46	40	42	4	6	0	2	12	14	8	10
38	38	36	34	32	46	44	42	40	6	4	2	0	14	12	10	8
40	40	42	44	46	32	34	36	38	8	10	12	14	0	2	4	6
42	42	40	46	44	34	32	38	36	10	8	14	12	2	0	6	4
44	44	46	40	42	36	38	32	34	12	14	8	10	4	6	0	2
46	46	44	42	40	38	36	34	32	14	12	10	8	6	4	2	0

Table 3: CVT patterns of class 1 of 3-variable Boolean functions.

CVT	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4
4	0	0	8	8	0	0	8	8	0	0	8	8	0	0	8	8
6	0	4	8	12	0	4	8	12	0	4	8	12	0	4	8	12
8	0	0	0	0	16	16	16	16	0	0	0	0	16	16	16	16
10	0	4	0	4	16	20	16	20	0	4	0	4	16	20	16	20
12	0	0	8	8	16	16	24	24	0	0	8	8	16	16	24	24
14	0	4	8	12	16	20	24	28	0	4	8	12	16	20	24	28
32	0	0	0	0	0	0	0	0	64	64	64	64	64	64	64	64
34	0	4	0	4	0	4	0	4	64	68	64	68	64	68	64	68
36	0	0	8	8	0	0	8	8	64	64	72	72	64	64	72	72
38	0	4	8	12	0	4	8	12	64	68	72	76	64	68	72	76
40	0	0	0	0	16	16	16	16	64	64	64	64	80	80	80	80
42	0	4	0	4	16	20	16	20	64	68	64	68	80	84	80	84
44	0	0	8	8	16	16	24	24	64	64	8	8	80	80	72	72
46	0	4	8	12	16	20	24	28	64	68	72	76	80	84	88	92

By invoking the above function in an algorithm, we can get other non-linear functions in a class. For this purpose, one has to put all possible binary sequences of length $2^n - (n+1)$, except those fixed bit positions of f. Taking different affine functions as input, different classes can be generated.

3.3. *List of Inferences Drawn from the above Classification Method*

(1) The method of keeping some of the bit positions fixed and varying other bit positions with respect to a Boolean function will be a handle to find out equivalence classes of equal cardinality.

(2) The number of equivalence classes is equal to 2^k, where k is the number of fixed positions.

(3) Different set of fixed positions generates different classes of Boolean functions.

(4) The number of members in a particular class is 2^l for $0 \le l \le 2^n - k$, where l is the number of changing bit positions.

(5) How to select the set of representative functions that generate disjoint equivalence classes of equal cardinality? The generators are all possible k bit sequences in the fixed positions and the rest of the positions are arbitrarily filled up by 0/1. Any Boolean function generated through this procedure

Table 4

BF	DV	HD	No. of BF
Class 1			
BF	DV	HD	No. of BF
00000000	0	0	1
00000010	2	1	4
00100000	32		
00001000	8		
00000100	4		
00100010	34	2	6
00001010	10		
00101000	40		
00001100	12		
00000110	6		
00100100	36		
00101010	42	3	4
00001110	14		
00101100	44		
00100110	38		
00101110	46	4	1
Class 2			
BF	DV	HD	No. of BF
10101010	170	0	1
10100010	162	1	4
10101000	168		
10001010	138		
10101110	174		
10100000	160	2	6
10000010	130		
10001000	136		
10101100	172		
10001110	142		
10100110	166		
10000000	128	3	4
10001100	140		
10100100	164		
10000110	134		
10000100	132	4	1
Class 3			
BF	DV	HD	No. of BF
11001100	204	0	1
11001000	200	1	4
11001110	206		
11101100	236		
11000100	196		
11000000	192	2	6
11001010	202		
11101000	232		
11101110	238		
11000110	198		
11100100	228		

Table 4: Continued.

BF	DV	HD	No. of BF
11000010	194	3	4
11100000	224		
11101010	234		
11100110	230		
11100010	226	4	1
Class 4			
BF	DV	HD	No. of BF
01100110	102	0	1
01100010	98	1	4
01101110	110		
01100100	100		
01000110	70		
01100000	96	2	6
01000010	66		
01101010	106		
01101100	108		
01001110	78		
01000100	68		
01000000	64	3	4
01101000	104		
01001010	74		
01001100	76		
01001000	72	4	1
Class 5			
BF	DV	HD	No. of BF
11110000	240	0	1
11110010	242	1	4
11010000	208		
11111000	248		
11110100	244		
11010010	210	2	6
11111010	250		
11011000	216		
11111100	252		
11110110	246		
11010100	212		
11011010	218	3	4
11111110	254		
11011100	220		
11010110	214		
11011110	222	4	1
Class 6			
BF	DV	HD	No. of BF
01011010	90	0	1
01010010	82	1	4
01011000	88		
01111010	122		
01011110	94		

Table 4: Continued.

01010000	80	2	6
01110010	114		
01111000	120		
01011100	92		
01111110	126		
01010110	86		
01110000	112	3	4
01111100	124		
01010100	84		
01110110	118		
01110100	116	4	1
Class 7			
BF	DV	HD	No. of BF
00111100	60	0	1
00111000	56	1	4
00111110	62		
00011100	28		
00110100	52		
00110000	48	2	6
00111010	58		
00011000	24		
00011110	30		
00110110	54		
00010100	20		
00110010	50	3	4
00010000	16		
00011010	26		
00010110	22		
00010010	18	4	1
Class 8			
BF	DV	HD	No. of BF
10010110	150	0	1
10010010	146	1	4
10011110	158		
10010100	148		
10110110	182		
10010000	144	2	6
10110010	178		
10011010	154		
10011100	156		
10111110	190		
10110100	180		
10110000	176	3	4
10011000	152		
10111010	186		
10111100	188		
10111000	184	4	1

Table 4: Continued.

Class 9			
BF	DV	HD	No. of BF
11111111	255	0	1
11111101	253	1	4
11011111	223		
11110111	247		
11111011	251		
11011101	221	2	6
11110101	245		
11010111	215		
11110011	243		
11111001	249		
11011011	219		
11010101	213	3	4
11110001	241		
11010011	211		
11011001	217		
11010001	209	4	1
Class 10			
BF	DV	HD	No. of BF
01010101	85	0	1
01011101	93	1	4
01010111	87		
01110101	117		
01010001	81		
01011111	95	2	6
01111101	125		
01110111	119		
01010011	83		
01110001	113		
01011001	89		
01111111	127	3	4
01110011	115		
01011011	91		
01111001	121		
01111011	123	4	1
Class 11			
BF	DV	HD	No. of BF
00110011	51	0	1
00110111	55	1	4
00110001	49		
00010011	19		
00111011	59		
00111111	63	2	6
00110101	53		
00010111	23		
00010001	17		
00111001	57		
00011011	27		

Table 4: Continued.

00111101	61	3	4
00011111	31		
00010101	21		
00011001	25		
00011101	29	4	1
Class 12			
BF	DV	HD	No. of BF
10011001	153	0	1
10011101	157	1	4
10010001	145		
10011011	155		
10111001	185		
10011111	159	2	6
10111101	189		
10010101	149		
10010011	147		
10110001	177		
10111011	187		
10111111	191	3	4
10010111	151		
10110101	181		
10110011	179		
10110111	183	4	1
Class 13			
BF	DV	HD	No. of BF
00001111	15	0	1
00001101	13	1	4
00101111	47		
00000111	7		
00001011	11		
00101101	45	2	6
00000101	5		
00100111	39		
00000011	3		
00001001	9		
00101011	43		
00100101	37	3	4
00000001	1		
00100011	35		
00101001	41		
00100001	33	4	1
Class 14			
BF	DV	HD	No. of BF
10100101	165	0	1
10101101	173	1	4
10100111	167		
10000101	133		
10100001	161		

Table 4: Continued.

10101111	175	2	6
10001101	141		
10000111	135		
10100011	163		
10000001	129		
10101001	169		
10001111	143	3	4
10000011	131		
10101011	171		
10001001	137		
10001011	139	4	1
Class 15			
BF	DV	HD	No. of BF
11000011	195	0	1
11000111	199	1	4
11000001	193		
11100011	227		
11001011	203		
11001111	207	2	6
11000101	197		
11100111	231		
11100001	225		
11001001	201		
11101011	235		
11001101	205	3	4
11101111	239		
11100101	229		
11101001	233		
11101101	237	4	1
Class 16			
BF	DV	HD	No. of BF
01101001	105	0	1
01101101	109	1	4
01100001	97		
01101011	107		
01001001	73		
01101111	111	2	6
01001101	77		
01100101	101		
01100011	99		
01000001	65		
01001011	75		
01001111	79	3	4
01100111	103		
01000101	69		
01000011	67		
01000111	71	4	1

BF: Boolean function, DV: decimal value, HD: Hamming distance, and No. BF: number of Boolean functions.

Table 5

(a)

XOR	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
0	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
2	2	0	6	4	10	8	14	12	34	32	38	36	42	40	46	44
4	4	6	0	2	12	14	8	10	36	38	32	34	44	46	40	42
6	6	4	2	0	14	12	10	8	38	36	34	32	46	44	42	40
8	8	10	12	14	0	2	4	6	40	42	44	46	32	34	36	38
10	10	8	14	12	2	0	6	4	42	40	46	44	34	32	38	36
12	12	14	8	10	4	6	0	2	44	46	40	42	36	38	32	34
14	14	12	10	8	6	4	2	0	46	44	42	40	38	36	34	32
32	32	34	36	38	40	42	44	46	0	2	4	6	8	10	12	14
34	34	32	38	36	42	40	46	44	2	0	6	4	10	8	14	12
36	36	38	32	34	44	46	40	42	4	6	0	2	12	14	8	10
38	38	36	34	32	46	44	42	40	6	4	2	0	14	12	10	8
40	40	42	44	46	32	34	36	38	8	10	12	14	0	2	4	6
42	42	40	46	44	34	32	38	36	10	8	14	12	2	0	6	4
44	44	46	40	42	36	38	32	34	12	14	8	10	4	6	0	2
46	46	44	42	40	38	36	34	32	14	12	10	8	6	4	2	0

(b)

XOR	128	130	132	134	136	138	140	142	160	162	164	166	168	170	172	174
128	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
130	2	0	6	4	10	8	14	12	34	32	38	36	42	40	46	44
132	4	6	0	2	12	14	8	10	36	38	32	34	44	46	40	42
134	6	4	2	0	14	12	10	8	38	36	34	32	46	44	42	40
136	8	10	12	14	0	2	4	6	40	42	44	46	32	34	36	38
138	10	8	14	12	2	0	6	4	42	40	46	44	34	32	38	36
140	12	14	8	10	4	6	0	2	44	46	40	42	36	38	32	34
142	14	12	10	8	6	4	2	0	46	44	42	40	38	36	34	32
160	32	34	36	38	40	42	44	46	0	2	4	6	8	10	12	14
162	34	32	38	36	42	40	46	44	2	0	6	4	10	8	14	12
164	36	38	32	34	44	46	40	42	4	6	0	2	12	14	8	10
166	38	36	34	32	46	44	42	40	6	4	2	0	14	12	10	8
168	40	42	44	46	32	34	36	38	8	10	12	14	0	2	4	6
170	42	40	46	44	34	32	38	36	10	8	14	12	2	0	6	4
172	44	46	40	42	36	38	32	34	12	14	8	10	4	6	0	2
174	46	44	42	40	38	36	34	32	14	12	10	8	6	4	2	0

(c)

XOR	192	194	196	198	200	202	204	206	224	226	228	230	232	234	236	238
192	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
194	2	0	6	4	10	8	14	12	34	32	38	36	42	40	46	44
196	4	6	0	2	12	14	8	10	36	38	32	34	44	46	40	42
198	6	4	2	0	14	12	10	8	38	36	34	32	46	44	42	40
200	8	10	12	14	0	2	4	6	40	42	44	46	32	34	36	38
202	10	8	14	12	2	0	6	4	42	40	46	44	34	32	38	36
204	12	14	8	10	4	6	0	2	44	46	40	42	36	38	32	34
206	14	12	10	8	6	4	2	0	46	44	42	40	38	36	34	32
224	32	34	36	38	40	42	44	46	0	2	4	6	8	10	12	14
226	34	32	38	36	42	40	46	44	2	0	6	4	10	8	14	12
228	36	38	32	34	44	46	40	42	4	6	0	2	12	14	8	10

(c) Continued.

XOR	192	194	196	198	200	202	204	206	224	226	228	230	232	234	236	238
230	38	36	34	32	46	44	42	40	6	4	2	0	14	12	10	8
232	40	42	44	46	32	34	36	38	8	10	12	14	0	2	4	6
234	42	40	46	44	34	32	38	36	10	8	14	12	2	0	6	4
236	44	46	40	42	36	38	32	34	12	14	8	10	4	6	0	2
138	46	44	42	40	38	36	34	32	14	12	10	8	6	4	2	0

can be a representative for the class. The number of generators for the proposed classification is 2^k.

(6) Any Boolean function of a class can be a representative of that class. In fact, taking affine function as the representative of a class will provide us with the guarantee of the inclusion of that affine function in that class.

4. Different Operations in Classes

In this section, classes are divided into several subclasses on using the Hamming distance (HD) between the Boolean functions and the affine function in that class. Also, the classes are analyzed on performing XOR and CVT operations among the functions of a class.

4.1. Subclassification. Hamming distance (HD) between two Boolean functions is denoted as $HD(f, g) = k$, where k can be $0, 1, 2, \ldots, 2^n - (n+1)$ where f is a Boolean function and g is an affine Boolean function and both belong to the same class of n-variable. Further, Boolean functions in a class having $HD = k$ with respect to the corresponding affine Boolean function form subclasses whose cardinality is *binomial coefficients* of the form ${}^{2^n-(n+1)}C_k$, where $k = 0, 1, 2, \ldots, 2^n - (n+1)$.

Illustration. Table 1 shows the 3-variable Boolean functions belonging to class 1, where the affine Boolean function is 0 = (00000000). There are five subclasses having cardinality 1, 4, 6, 4, and 1 with Hamming distance (HD) 0, 1, 2, 3, and 4, respectively. For 3-variables all classes and their subclasses are given in Appendix A.

4.2. XOR Operation in Classes. Let $a = (a_{2^n}, a_{2^n-1}, \ldots, a_1)$ and $b = (b_{2^n}, b_{2^n-1}, \ldots, b_1)$ be two n-variable Boolean functions belonging to a particular class. The XOR operation of all the classes when arranged in a table only gives those entries given by class 1 functions, as $(a+k) \oplus (b+k) = (a \oplus b) + (k \oplus k) = (a \oplus b)$, where, the XOR operation of a and b is defined as $a \oplus b = (a_{2^n} \oplus b_{2^n}, a_{2^n-1} \oplus b_{2^n-1}, \ldots, a_1 \oplus b_1)$.

Illustration. Suppose we want the XOR operation of $(44)_{10} = (00101100)_2$ and $(34)_{10} = (00100010)_2$ both belonging to class 1 of 3-variables. And $44 \oplus 34 = (00101100) \oplus (00100010) = (00001110) = 14$. Table 2 is constructed for all classes of n-variable Boolean functions that contain only the XOR values of all the functions in a class. The functions are arranged in ascending order in both rows and columns of the table. It can be proved that the content of each table remain invariant under the XOR operation and the decimal values of the content in the table are same as in class 1. For 3-variables the XOR operation of other classes are given in Appendix B.

4.3. CVT Operation in Classes. Let $a = (a_k, a_{k+1}, \ldots, a_1)$ and $b = (b_k, b_{k+1}, \ldots, b_1)$ be two Boolean functions in a Class. Then the Carry Value Transform (CVT) of a and b is defined in [13] as $CVT(a, b) = (a_k \wedge b_k, a_{k-1} \wedge b_{k-1}, \ldots, a_1 \wedge b_1, 0)$. Carry Value Transformation (CVT) is a kind of representation of n-variable Boolean functions and is used to produce many interesting patterns [13]. Under the CVT operation, we have observed some interesting self-similar fractal patterns which are invariant for all classes of n-variable Boolean functions.

Illustration. The CVT operation of $(44)_{10} = (00101100)_2$ and $(34)_{10} = (00100010)_2$ is 64. The patterns for class 1 functions using CVT operation is shown in Table 3 and others are shown in Appendix C.

5. Conclusion

The novelty of this paper lies in its systematic classification of Boolean functions with focal emphasis on the prominent binary operations like Hamming distance, XOR, and CVT. The present analytical study introduces a new way towards the formulation of an universal classifier of arbitrary length which is being actively pursued. The procedures followed in this paper are very handy and useful even for our future experimental research in this domain of theoretical computer science. A number of tables have been incorporated in this paper for easy reference and clear comprehension showing varied subclasses, patterns, and values of different classes.

Appendices

A. Subclassification

Table 4 shows the classes and subclasses of 3-variable Boolean functions.

B. XOR Operations in Classes

Table 5 shows the XOR operation values of class-1, class-2 and class-3 of 3-variable Boolean functions.

Table 6

(a)

CVT	0	2	4	6	8	10	12	14	32	34	36	38	40	42	44	46
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4
4	0	0	8	8	0	0	8	8	0	0	8	8	0	0	8	8
6	0	4	8	12	0	4	8	12	0	4	8	12	0	4	8	12
8	0	0	0	0	16	16	16	16	0	0	0	0	16	16	16	16
10	0	4	0	4	16	20	16	20	0	4	0	4	16	20	16	20
12	0	0	8	8	16	16	24	24	0	0	8	8	16	16	24	24
14	0	4	8	12	16	20	24	28	0	4	8	12	16	20	24	28
32	0	0	0	0	0	0	0	0	64	64	64	64	64	64	64	64
34	0	4	0	4	0	4	0	4	64	68	64	68	64	68	64	68
36	0	0	8	8	0	0	8	8	64	64	72	72	64	64	72	72
38	0	4	8	12	0	4	8	12	64	68	72	76	64	68	72	76
40	0	0	0	0	16	16	16	16	64	64	64	64	80	80	80	80
42	0	4	0	4	16	20	16	20	64	68	64	68	80	84	80	84
44	0	0	8	8	16	16	24	24	64	64	8	8	80	80	72	72
46	0	4	8	12	16	20	24	28	64	68	72	76	80	84	88	92

(b)

CVT	128	130	132	134	136	138	140	142	160	162	164	168	170	172	174	176
128	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
130	256	260	256	260	256	260	256	260	256	260	256	260	256	260	256	260
132	256	256	264	264	256	256	264	264	256	256	264	264	256	256	264	264
134	256	260	264	268	256	260	264	268	256	260	264	268	256	260	264	268
136	256	256	256	256	272	272	272	272	256	256	256	256	272	272	272	272
138	256	260	256	260	272	276	272	276	256	260	256	260	272	276	272	276
140	256	256	264	264	272	272	280	280	256	256	264	264	272	272	280	280
142	256	260	264	268	272	276	280	284	256	260	264	268	272	276	280	284
160	256	256	256	256	256	256	256	256	320	320	320	320	320	320	320	320
162	256	260	256	260	256	260	256	260	320	324	320	324	320	324	320	324
164	256	256	264	264	256	256	264	264	320	320	328	328	320	320	328	328
166	256	260	264	268	272	276	280	284	320	324	328	323	320	324	328	323
168	256	256	256	256	272	272	272	272	320	320	320	320	336	336	336	336
170	256	260	256	260	272	276	272	276	320	324	320	324	336	340	336	340
172	256	256	264	264	272	272	280	280	320	320	328	328	336	336	344	344
174	256	260	264	268	272	276	280	284	320	324	328	332	336	340	344	348

(c)

CVT	192	194	196	198	200	202	204	206	224	226	228	230	232	234	236	238
192																
194																
196																
198																
200																
202																
204																
206																
224																
226																
228																
230																
232																
234																
136																
138																

C. CVT Operations in Classes

Table 6 shows the CVT (Carry Value Transformation) patterns of class 1, class 2 and class 3 of 3-variable Boolean Functions.

Acknowledgment

The authors are grateful to Professor Birendra Kumar Nayak of Utkal University and Mr Sk. Sarif Hassan of Institute of Mathematics and Applications, Bhubaneswar, for their valuable suggestions.

References

[1] D. Slepian, "On the number of symmetry types of Boolean functions of n variables," *Canadian Journal of Mathematics*, vol. 5, no. 2, pp. 185–193, 1953.

[2] S. W. Golomb, "On the classification of Boolean functions," *IRE Transactions on Circuit Theory*, vol. 6, no. 5, pp. 176–186, 1959.

[3] M. A. Harrison, "On the classification of Boolean functions by the general linear and affine groups," *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 2, pp. 285–299, 1964.

[4] V. P. Correia and A. I. Reis, "Classifying n-input Boolean functions," in *Proceedings of the 7th Workshop IBERCHIP (IWS '01)*, pp. 58–66, Montevideo, Uruguay, March 2001.

[5] A. Braeken, Y. Borissov, S. Nikova, and B. Preneel, "Classification of Boolean functions of 6 variables or less with respect to some cryptographic properties," in *Automata, Languages and Programming*, L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds., vol. 3580 of *Lecture Notes in Computer Science*, pp. 324–334, Springer, Berlin, Germany, 2005.

[6] P. Stănică and S. H. Sung, "Boolean functions with five controllable cryptographic properties," *Designs, Codes and Cryptography*, vol. 31, no. 2, pp. 147–157, 2004.

[7] Y. V. Taranikov, "On resilient functions with maximum possible nonlinearity," in *Progress in Cryptology—INDOCRYPT 2000*, B. Roy and E. Okamoto, Eds., vol. 1977 of *Lecture Notes in Computer Science*, pp. 19–30, Springer, 2000.

[8] X.-M. Zhang and Y. Zheng, "Cryptographically resilient functions," *IEEE Transactions on Information Theory*, vol. 43, no. 5, pp. 1740–1747, 1997.

[9] W. Millan, A. Clark, and E. Dawson, "Heuristic design of cryptographically strong balanced Boolean functions," in *Advances in Cryptology—EUROCRYPT'98*, K. Nyberg, Ed., vol. 1403 of *Lecture Notes in Computer Science*, pp. 489–499, Springer, 1998.

[10] P. P. Choudhury, S. Sahoo, and M. Chakraborty, "Characterization of the evolution of nonlinear uniform cellular automata in the light of deviant states," *International Journal of Mathematics and Mathematical Sciences*, vol. 2011, Article ID 605098, 16 pages, 2011.

[11] P. P. Choudhury, S. Sahoo, M. Chakraborty, S. K. Bhandari, and A. Pal, "Investigation of the global dynamics of cellular automata using Boolean derivatives," *Computers and Mathematics with Applications*, vol. 57, no. 8, pp. 1337–1351, 2009.

[12] S. Sahoo, P. P. Choudhury, and M. Chakraborty, "Characterization of any non-linear Boolean function using a set of linear operators," *Journal of Orissa Mathematical Society*, vol. 2, no. 1-2, pp. 111–133, 2010.

[13] P. P. Choudhury, S. Sahoo, B. K. Nayak, and Sk. S. Hassan, "Theory of Carry Value Transformation (CVT) and its application in fractal formation," *Global Journal of Computer Science and Technology*, vol. 10, no. 14, pp. 98–107, 2010.

[14] S. Wolfram, *A New Kind of Science*, Wolfram Media, Champaign, Ill, USA, 2002.

[15] B. K. Nayak, S. Sahoo, and S. Biswal, "Cellular automata rules and linear numbers," http://arxiv.org/abs/1204.3999.