Classification of Boolean Functions where Affine Functions are Uniformly Distributed

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

[1]Applied Statistics Unit, Indian Statistical Institute, Kolkata-700108, India,

[2]Institute of Mathematics and Applications, Bhubaneswar-751003, India

Email. ranjeetkumarrout@gmail.com, pabitrapalchoudhury@gmail.com,

sudhakar.sahoo@gmail.com

Abstract

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/ sub-classes in such a way that the members of classes/ sub-classes satisfy certain similar properties.

Keyword. Affine Boolean Function, Truth Table, Classification, Carry Value Transformation, XOR Operation, Hamming Distance.

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 welcome 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 in number than the number 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 [2]. Further, it has also been elaborated in [1] 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 have been discussed in [5, 12, 13, and 14]. Heuristic design of cryptographically strong balanced Boolean function was envisaged in [15]. In [7], 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 [6], [7], and [8] 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) [10]. 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 [11]. The complement of f is denoted as $\bar{f}$.

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:

$f_{affine}(x_1, x_2, x_3, \ldots, x_n) = k_n x_n \oplus k_{n-1} x_{n-1} \oplus \cdots \oplus k_2 x_2 \oplus k_1 x_1 \oplus k_0$, where the co-efficients 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 function 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, first eight are linear and 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 $\bar{f}$ are denoted as ff and $f\bar{f}$ respectively. For example,

if $f = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, then $ff = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ and $f\bar{f} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$.

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

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

Apart from the above concatenations as stated in Theorem 1, all other concatenations give non-linear Boolean functions [9].

Corollary 1 *f is an affine Boolean function if and only if ff, $f\bar{f}$, $\bar{f}f$ and $\bar{f}\bar{f}$ are affine Boolean functions.*

Proof. The proof of the corollary is the consequence of Theorem 1 and using the definition of affine Boolean functions.

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 following.

$$S_1 \times S_1^{'} = \{\{\mathbf{0000}, 0010\}, \{1000, \mathbf{1010}\}, \{\mathbf{1100}, 1110\}, \{0100, \mathbf{0110}\}\} \tag{1}$$

and

$$S_1 \times S_1^{''} = \{\{\mathbf{0011}, 0001\}, \{1011, \mathbf{1001}\}, \{\mathbf{1111}, 1101\}, \{0111, \mathbf{0101}\}\} \tag{2}$$

Note that, S_1 contains four classes each containing a 1-variable Boolean functions where as, the set $\left(S_1 \times S_1^{'}\right) \cup \left(S_1 \times S_1^{''}\right)$ 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 (1) and (2). This process is repeated for the next higher variable, using the recursive formula of (3) and (4).

(i) **Base Case:(For $n = 1$)**

$$S_1^{'} = \{\{00\}, \{10\}\},\ S_1^{''} = \{\{11\}, \{01\}\} \ \textit{and}\ S_1 = (S_1^{'} \cup S_1^{''}) = \{\{00\}, \{10\}, \{11\}, \{01\}\} \tag{3}$$

(ii) **Recursion :(For $n \geq 2$)**

$$S_n^{'} = (S_{n-1} \times S_{n-1}^{'}),\ \ S_n^{''} = (S_{n-1} \times S_{n-1}^{''}) \ \textit{and}\ S_n = (S_{n-1}^{'} \cup S_{n-1}^{'''}). \tag{4}$$

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 2 *The recursive procedure of equation* (3) *and* (4)*, when repeated up to* $(n-1)$ *times, classify 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 1** of **Section 2**.

Illustration: **(from 2-variable classes to 3-variable classes)**

From equation (3) and (4) the set $S_2 = \left\{ \begin{array}{l} \{\mathbf{0000}, 0010\}, \{1000, \mathbf{1010}\}, \{\mathbf{1100}, 1110\}, \{0100, \mathbf{0110}\} \\ \{\mathbf{0011}, 0001\}, \{1011, \mathbf{1001}\}, \{\mathbf{1111}, 1101\}, \{0111, \mathbf{0101}\} \end{array} \right\}$

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 Figure 1.

$$S_3^{'} = \left\{ \begin{array}{ll} \mathbf{00000000}, & \\ 00000010, & \\ 00001000, & \\ 00001010, & \\ 00001100, & \\ 00001110, & \\ 00000100, & \\ 00000110, & Class2, \ldots, Class8 \\ 00100000, & \\ 00100010, & \\ 00101000, & \\ 00101010, & \\ 00101100, & \\ 00101110, & \\ 00100100, & \\ 00100110 & \end{array} \right\}, S_3^{''} = \left\{ \begin{array}{ll} 00000011, & \\ 00000001, & \\ 00001011, & \\ 00001001, & \\ \mathbf{00001111}, & \\ 00001101, & \\ 00000111, & \\ 00000101, & Class10, \ldots, Class16 \\ 00100011, & \\ 00100001, & \\ 00101011, & \\ 00101001, & \\ 00101111, & \\ 00101101, & \\ 00100111, & \\ 00100101 & \end{array} \right\}$$

[Figure 1: 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 the above figure, only the members of CLASS 1 and CLASS 13 are shown and other classes of Boolean functions are shown in **Appendix-A**].

Theorem 3 *The number of different classes in the above classification is* 2^{n+1}.

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

Theorem 4 *The classes are of equal size and the cardinality of each class equals 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 3** the cardinality of each class=$\frac{total\ number\ of\ n-variable Boolean\ functions}{total\ number\ of\ n-variable\ affine\ Boolean\ functions} = \frac{2^{2^n}}{2^{n+1}}$ $= 2^{2^n-(n+1)}$.

Theorem 5 *The least significant bit of all the Boolean functions in* $S_n^{'}$ *is* 0*, where as 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 preserve 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 6 *For each class of n-variable, the length of a Boolean function is* 2^n*, out of which* $(n+1)$ *bits are fixed and 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:

$$P_{n-1} - 2^k, \textit{ where } P_{n-1} = 2^{n-1}+1 \textit{ and } k = 0, 1, 2, \ldots, n-1. \tag{5}$$

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 are fixed and satisfies equation-(5). From Theorem 5, 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 equation-(5). 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 $k = 0, 1, 2, \ldots, n-1$,

$\{P_{n-1} - 2^k\} + 2^{n-1} = \{(2^{n-1}+1) - 2^k\} + 2^{n-1} = \{2^n+1\} - 2^k = P_n - 2^k$

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

$1 = (2^n+1) - 2^n = P_n - 2^n = P_n - 2^k$

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 sub-classes 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 2 *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 6 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.

Using the result of Theorem 6, 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, $f\ R\ g$ *iff "there exist* $(n+1)$ *bit positions calculated on using Theorem* 6 *and the calculated bit positions are same for the functions* f *and* g*".* Clearly,

1. $f\ R\ f\ \forall f$. So, R is reflexive.
2. If $f\ R\ g$ then $g\ R\ f$. So, R is symmetric.
3. If $f\ R\ g$ and $g\ R\ h$ then $f\ R\ h$. 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 1. The worst case time complexity of the Algorithm is $O(n)$.

Algorithm 1

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.

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, shall be a handle to find out equivalence classes of equal cardinality.
2. 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 $0 \leq l \leq 2^n - k$, 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 rest of the positions are arbitrarily filled up by 0/1(Dont care). Any Boolean function generated through this procedure, 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 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 sub-classes 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 Sub-classification

Hamming Distance(HD) between two Boolean functions, 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 belongs to the same class of n-variable. Further, Boolean functions in a class having $HD = k$ with respect to the corresponding affine Boolean function forms sub-classes, whose cardinality are ***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 sub-classes 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 sub-classes are given in **Appendix A**.

Table 1: Shows different Sub-classes 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

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 the 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 belong 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**.

Table 2: Shows 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

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 [10] 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 [10]. 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 below in Table 3 and others are shown in **Appendix C**.

Table 3: Shows 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

5 Conclusion

The novelty of this article 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 article 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 article for easy reference and clear comprehension showing varied sub-classes, patterns and values of different classes.

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

References

[1] S. W Golomb (1959), "On the classification of Boolean functions", IRE transactions on circuit theory, Vol. 06, Issue. 05, pp.176- 186 .

[2] D. Slepian (1954), "On the number of symmetry types of Boolean functions of n variables", Society for industrial and Mathematics", Vol. 5, No. 2, pp. 185-193.

[3] M. A. Harrison (1964), "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.

[4] V. P. Correia, A. I. Reis (2001), "Classifying n-Input Boolean Functions.", IBERCHIP, pp.58-66.

[5] An. Braeken, Y. Borissov, S. Nikova, B. Preneel (2005), "Classification of Boolean Functions of 6 Variables or Less with Respect to Cryptographic Properties", ICALP'05 Proceedings of the 32nd international conference on Automata, Languages and Programming, Springer-Verlag Berlin, Heidelberg , pp. 324-334.

[6] P. Pal Choudhury, S. Sahoo, M. Chakarborty, S. K Bhandari, A. Pal(2009), "Investigation of the Global Dynamics of Cellular Automata Using Boolean Derivatives ", Computers and Mathematics with Applications, Elsevier, 57,pp. 1337-1351.

[7] P. Pal Choudhury, S. Sahoo, M. Chakraborty (2011), "Characterization of the evolution of Non-linear Uniform Cellular Automata in the light of Deviant States", Mathematics and Mathematical Sciences, Vol. 2011, Article ID 605098.

[8] S. Sahoo, P. Pal Choudhury, M. Chakraborty (2010), "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.

[9] B. K Nayak, S. Sahoo, S. Biswal, "Cellular Automata Rules and Linear Numbers", arxiv.org/pdf /1204.3999.

[10] P. Pal Choudhury, S. Sahoo, B. K Nayak, Sk. Sarif Hassan(2010), "Theory of Carry Value Transformation (CVT) and its Application in Fractal formation", Global Journal of Computer Science and Technology, Vol. 10, Issue 14, Version 1.0, pp. 98-107.

[11] S. WolForm(2002), "A New Kind of Science", Wolfram Media, Inc., 2002.

[12] P. Stanica, S.H. Sung(2004), "Boolean Functions with Five Controllable Cryptogrpahic Properties, Designs, Codes and Cryptography", Vol. 31, pp. 147-157.

[13] Y.V. Taranikov(2000), "On Resilient Functions with Maximum Possible Nonlinearity", Indocrypt 2000, LNCS 1977, Springer-Verlag, pp. 19-30.

[14] Y. Zheng, X. M. Zhang(1997), "Cryptographically Resilient Functions", IEEE Transactions on Information Theory, Vol. 43 (5), pp. 1740-1747.

[15] W. Millan, A. Clark, E. Dawson(1998), "Heuristic Design of Cryptographically Strong Balanced Boolean Functions", Eurocrypt 98, LNCS 1403, Springer-Verlag, pp. 489-499.

A Sub-Classification

Following table shows the Classes and Sub-classes of 3-variable Boolean Functions.

Abbreviations: BF-Boolean Function, DV-Decimal Value, HD-Hamming Distance, No.BF-Number of Boolean Functions.

Class 1				Class 2				Class 3				Class 4			
BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF
00000000	0	0	1	**10101010**	170	0	1	**11001100**	204	0	1	**01100110**	102	0	1
00000010	2			10100010	162			11001000	200			01100010	98		
00100000	32	1	4	10101000	168	1	4	11001110	206	1	4	01101110	110	1	4
00001000	8			10001010	138			11101100	236			01100100	100		
00000100	4			10101110	174			11000100	196			01000110	70		
00100010	34			10100000	160			11000000	192			01100000	96		
00001010	10			10000010	130			11001010	202			01000010	66		
00101000	40	2	6	10001000	136	2	6	11101000	232	2	6	01101010	106	2	6
00001100	12			10101100	172			11101110	238			01101100	108		
00000110	6			10001110	142			11000110	198			01001110	78		
00100100	36			10100110	166			11100100	228			01000100	68		
00101010	42			10000000	128			11000010	194			01000000	64		
00001110	14	3	4	10001100	140	3	4	11100000	224	3	4	01101000	104	3	4
00101100	44			10100100	164			11101010	234			01001010	74		
00100110	38			10000110	134			11100110	230			01001100	76		
00101110	46	4	1	10000100	132	4	1	11100010	226	4	1	01001000	72	4	1
Class 5				Class 6				Class 7				Class 8			
BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF
11110000	240	0	1	**01011010**	90	0	1	**00111100**	60	0	1	**10010110**	150	0	1
11110010	242			01010010	82			00111000	56			10010010	146		
11010000	208	1	4	01011000	88	1	4	00111110	62	1	4	10011110	158	1	4
11111000	248			01111010	122			00011100	28			10010100	148		
11110100	244			01011110	94			00110100	52			10110110	182		
11010010	210			01010000	80			00110000	48			10010000	144		
11111010	250			01110010	114			00111010	58			10110010	178		
11011000	216	2	6	01111000	120	2	6	00011000	24	2	6	10011010	154	2	6
11111100	252			01011100	92			00011110	30			10011100	156		
11110110	246			01111110	126			00110110	54			10111110	190		
11010100	212			01010110	86			00010100	20			10110100	180		
11011010	218			01110000	112			00110010	50			10110000	176		
11111110	254	3	4	01111100	124	3	4	00010000	16	3	4	10011000	152	3	4
11011100	220			01010100	84			00011010	26			10111010	186		
11010110	214			01110110	118			00010110	22			10111100	188		
11011110	222	4	1	01110100	116	4	1	00010010	18	4	1	10111000	184	4	1
Class 9				Class 10				Class 11				Class 12			
BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF
11111111	255	0	1	**01010101**	85	0	1	**00110011**	51	0	1	**10011001**	153	0	1
11111101	253			01011101	93			00110111	55			10011101	157		
11011111	223	1	4	01010111	87	1	4	00110001	49	1	4	10010001	145	1	4
11110111	247			01110101	117			00010011	19			10011011	155		
11111011	251			01010001	81			00111011	59			10111001	185		
11011101	221			01011111	95			00111111	63			10011111	159		
11110101	245			01111101	125			00110101	53			10111101	189		
11010111	215	2	6	01110111	119	2	6	00010111	23	2	6	10010101	149	2	6
11110011	243			01010011	83			00010001	17			10010011	147		
11111001	249			01110001	113			00111001	57			10110001	177		
11011011	219			01011001	89			00011011	27			10111011	187		
11010101	213			01111111	127			00111101	61			10111111	191		
11110001	241	3	4	01110011	115	3	4	00011111	31	3	4	10010111	151	3	4
11010011	211			01011011	91			00010101	21			10110101	181		
11011001	217			01111001	121			00011001	25			10110011	179		
11010001	209	4	1	01111011	123	4	1	00011101	29	4	1	10110111	183	4	1
Class 13				Class 14				Class 15				Class 16			
BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF	BF	DV	HD	No. Of BF
00001111	15	0	1	**10100101**	165	0	1	**11000011**	195	0	1	**01101001**	105	0	1
00001101	13			10101101	173			11000111	199			01101101	109		
00101111	47	1	4	10100111	167	1	4	11000001	193	1	4	01100001	97	1	4
00000111	7			10000101	133			11100011	227			01101011	107		
00001011	11			10100001	161			11001011	203			01001001	73		
00101101	45			10101111	175			11001111	207			01101111	111		
00000101	5			10001101	141			11000101	197			01001101	77		
00100111	39	2	6	10000111	135	2	6	11100111	231	2	6	01100101	101	2	6
00000011	3			10100011	163			11100001	225			01100011	99		
00001001	9			10000001	129			11001001	201			01000001	65		
00101011	43			10101001	169			11101011	235			01001011	75		
00100101	37			10001111	143			11001101	205			01001111	79		
00000001	1	3	4	10000011	131	3	4	11101111	239	3	4	01100111	103	3	4
00100011	35			10101011	171			11100101	229			01000101	69		
00101001	41			10001001	137			11101001	233			01000011	67		
00100001	33	4	1	10001011	139	4	1	11101101	237	4	1	01000111	71	4	1

B XOR Operations in Classes

Following table shows the XOR operation values of Class-1,Class-2 and Class-3 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

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

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
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

C CVT Operations in Classes

Following tables shows the CVT(Carry Value Transformation) patterns of Class-1, Class-2 and Class-3 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

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

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																