Computing Affine Equivalence Classes of Boolean Functions by Group Isomorphism

Yan Zhang, Guowu Yang, William N.N. Hung, and Juling Zhang

Abstract—Affine equivalence classification of Boolean functions has significant applications in logic synthesis and cryptography. Previous studies for classification have been limited by the large set of Boolean functions and the complex operations on the affine group. Although there are many research on affine equivalence classification for parts of Boolean functions in recent years, there are very few results for the entire set of Boolean functions. The best existing result has been achieved by Harrison with 15768919 affine equivalence classes for 6-variable Boolean functions. This paper presents a concise formula for affine equivalence classification of the entire set of Boolean functions as well as a formula for affine classification of Boolean functions with distinct ON-set size respectively. The method outlined in this paper greatly simplifies the affine group's action by constructing an isomorphism mapping from the affine group to a permutation group. By this method, we can compute the affine equivalence classes for up to 10 variables. Experiment results indicate that our scheme for calculating the affine equivalence classes for more than 6 variables is a significant advancement over previous published methods.

Index Terms—Logic synthesis, Boolean functions, affine equivalence classification, group isomorphism

1 Introduction

Boolean functions play a critical role in science and technology. As an elementary equivalence among Boolean functions, affine equivalence has significant applications in logic synthesis for combinational logic circuit and Field Programmable Gate Array (FPGA) [1]. The affine equivalence of Boolean functions also has critical applications in cryptography such as S-box [2] and Reed-Muller codes [3], [4], [5], [6], [7].

Two Boolean functions $f(x_1, x_2, \ldots, x_n)$ and $h(x_1, x_2, \ldots, x_n)$ are said to be affine equivalent if $h(x_1, x_2, \ldots, x_n)$ can be written as $h(X) = f(\mathbf{A}X + \mathbf{b})$, where $\mathbf{A}$ is an $n \times n$ nonsingular matrix over the Boolean field $\mathbb{F}_2 = \{0, 1\}$, X is a column vector whose transpose is $X^T = [x_1, x_2, \ldots, x_n]$, and $\mathbf{b}$ is an n-dimensional vector over $\mathbb{F}_2$. The addition $\mathbf{A}X + \mathbf{b}$ is also over $\mathbb{F}_2$. We say that $(\mathbf{A}, \mathbf{b})$ is a nonsingular affine transformation from f to h. All possible nonsingular affine transformations form the affine group $AGL(n, 2)$ [8]. The affine equivalence classification is the classification of Boolean functions under the affine group $AGL(n, 2)$.

Note that all of the vectors in this paper represent column vectors. Uppercase bold letters express matrixes. Uppercase un-bolded letters and lowercase bold letters represent vectors. Lowercase un-bolded letters represent scalars.

In the design of a combinational logic circuit, each physical circuit with n inputs and a single output can be designed based on a Boolean function with n variables. Specifically, if the values of variables $x_1, x_2, \ldots, x_n$ represent the states of the n inputs (each x_i takes value 0 or 1), then the state of the output can be given by the value of $f(x_1, x_2, \ldots, x_n)$. Any Boolean function obtained from f by an affine transformation of variables can be treated as corresponding to the same physical circuit as f. It is necessary to define two Boolean functions as being of the same affine equivalence class if one of the functions can be obtained from the other by an affine transformation of variables. So there are as many different physical circuits of n inputs as the number of affine equivalence classes with n-variable Boolean functions. The aim of this paper is to enumerate affine equivalence classes of Boolean functions.

Due to many similar cryptographic properties owned by affine equivalent Boolean functions, such as correlation immunity, resiliency and propagation characteristics [9], the affine equivalence classification of Boolean functions can be applied to the study of cryptography, such as the classification of Reed-Muller cosets. The affine equivalence relation between two Reed-Muller cosets $f(X), h(X) \in R(r, n)/R(s, n)$ was expressed as $h(X) = f(\mathbf{A}X + \mathbf{b}) \bmod R(s, n)$ [10]. When comparing the affine equivalence relation between two Boolean functions, we can conclude that if two Boolean functions f and h are affine equivalent, then they are also in the same affine equivalence class of Reed-Muller cosets. The affine equivalence classification of the cosets has been widely studied by many scholars such as Braeken et al. [9], Hou [11] and [12]. Owing to the fact that affine equivalent Boolean functions are also equivalent in the Reed-Muller cosets, the results of classification for Reed-Muller cosets can be achieved by the classification of Boolean functions.

1.1 Related Work

In recent years, many scholars have studied the affine equivalence classification of a special case of Boolean functions, known as monomial rotation symmetric (MRS) functions

- *Y. Zhang, G.W. Yang, and J.L. Zhang are with the Big Data Research Center, University of Electronic Science and Technology of China, Chengdu 611731, China. E-mail: yixianqianzy@gmail.com, guowu@uestc.edu.cn, zjlgj@163.com.*
- *W.N.N. Hung is with Synopsys Inc., Mountain View, CA 94043. E-mail: William.Hung@synopsys.com.*

Manuscript received 1 Oct. 2015; revised 7 Apr. 2016; accepted 8 Apr. 2016. Date of publication 24 Apr. 2016; date of current version 14 Nov. 2016. Recommended for acceptance by P. Eles. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2016.2557329

[13], [14], because MRS functions have important applications in cryptography. Among them, Cusick gave a detailed description of the affine equivalence classification for quadratic, cubic and quartic MRS functions respectively in [14]. In addition, there are many theoretical results regarding symmetric Boolean functions [15], [16], [17]. Besides, there is another study about affine equivalence [18] of S-boxes, but it is different from the affine equivalence of Boolean functions discussed in this paper.

However, there is little study of affine equivalence classification for the entire set of Boolean functions. Previous researchers calculated the affine equivalence classes via the idea of Burnside's lemma [19]. Burnside's lemma offers a method to calculate orbits of a set Ω under the action of a group G.

In the study of computing the affine classes of Boolean functions under the affine group, we will replace G with the affine group $AGL(n,2)$ and replace Ω with F_n (the entire set of Boolean functions with n variables). As a matter of fact, the number of orbits denoted by N_n is the number of affine equivalence classes of F_n. Limited by the computational capability of present computers, Burnside's lemma cannot work for Boolean functions with more than six variables. Early in 1959 Nechiporuk [20] calculated N_n by using the idea of Burnside's lemma for $n \leq 5$ variables. The numbers N_n for $n \leq 6$ were achieved by Herrison [21] who divided the six-variable Boolean functions into 15,768,919 affine classes. In 1997, Strazdins [22] discovered a rule for the affine equivalence classification of a small group of Boolean functions for $n \geq 6$. Additionally, there has been a great deal of research [23], [24] about decomposition of Boolean functions. The decomposition of Boolean functions is helpful to classify Boolean functions into affine equivalence classes when n is large.

In 1953, Slepian obtained the numbers of symmetry equivalence classes of Boolean functions by a recursion method [25]. The symmetry classification of Boolean functions was applied to Logic Synthesis and technology mapping for FPGAs [26], [27] and Programmable Logic Blocks (PLBs) [28]. But the size of symmetry classes is much larger than the affine classes for the same n. For example, the number of symmetry classes for $n = 5$ is 1,228,158 [25] while the number of affine classes is 382 [21]. In the design of FPGA, affine classification is more important than symmetry classification since we will obtain fewer representative Boolean functions which lead us to design fewer PLBs to achieve the same features. This motivated us to pursue a deeper study of affine equivalence classification.

1.2 Organization of This Paper

The research progress of affine equivalence classification for the complete set of Boolean functions has been restricted by the size of F_n and the complex operations on the affine group. This paper will provide advancement based on isomorphism mapping to overcome the limit of complex group action. In Section 2, some basic theories related to this paper are presented. In Section 3, an analysis of the isomorphism between $AGL(n,2)$ and a permutation group denoted by P_n is shown in detail. Then we discuss the transformation from the affine equivalence classification under $AGL(n,2)$ to the classification under P_n. In Section 4, we present the details of our classification procedure under P_n. Motivated by the method in [25], we first classify P_n into disjoint conjugate classes and find a representative from each conjugate class. We then compute the number of Boolean functions fixed by these representatives. In this way, the computation of N_n is greatly simplified. Finally, we compute the affine equivalence classes of Boolean functions with $n \leq 10$ variables using GAP [29].

2 Preliminaries

This section provides a brief introduction of the theory associated with this paper. As a theory closely related to this paper, Burnside's lemma states that the number of orbits N can be determined by the following formula when a group G acts on a set Ω,

$$N = \frac{1}{|G|}\sum_{g\in G}|Fix(g)|, \tag{1}$$

where

$$Fix(g) = \{\omega \in \Omega | g\omega = \omega\}. \tag{2}$$

A Boolean function f is a mapping from $\mathbb{F}_2^n = \{0,1\}^n$ to $\mathbb{F}_2 = \{0,1\}$ and it can be written as $f(X) : \mathbb{F}_2^n \to \mathbb{F}_2$, where $X^T = [x_1, \ldots, x_n]$. The common representation of a Boolean function is the Truth Table. It can also be expressed as

$$T_f = (f(0), f(1), \ldots, f(2^n - 1)), \tag{3}$$

where each element $f(i)$ represents the output of $f(X)$ when the corresponding input $(x_1, \ldots, x_n)$ is the n-bit binary representation of integer i. Only the terms of (3) that take value 1 will be considered in designing a circuit. So in this paper we use another representation named ON-set representation defined as follows.

Definition 2.1. *Suppose f is an n-variable Boolean function. The ON-set of f is a set denoted by S_f which contains all possible inputs whose corresponding outputs take value 1. The expression of S_f can be written as*

$$S_f = \{i | f(i) = 1, 0 \leq i \leq 2^n - 1\}.$$

Let S denote the set of all possible inputs, then S can be written as

$$S = \{0, 1, \ldots, 2^n - 1\}. \tag{4}$$

It's clear that S_f is a subset of S. We can find that a Boolean function can be determined by the ON-set and it can also be regarded as another representation of the Truth Table.

Since equivalent Boolean functions have ON-set of the same size (the result will be shown in Lemma 4.6), we have the following definition.

Definition 2.2. *Consider the entire set of n-variable Boolean functions denoted as F_n, where we define F_n^m as the set of Boolean functions with ON-set size equal to m and define N_n^m as the number of affine equivalence classes of F_n^m. Then F_n^m can be represented as*

$$F_n^m = \{f \in F_n | |S_f| = m\}.$$

Obviously the size of F_n^m is

$$|F_n^m| = \binom{2^n}{m}.$$

N_n^m can be calculated by the method of Section 4.2.

As introduced in Section 1, the affine equivalence classification is the classification of Boolean functions under $AGL(n,2)$. The general linear group [4] denoted as $GL(n,2)$ is composed of all non-singular matrices on $\mathbb{F}_2$ and $AGL(n,2)$ [8] is expressed as

$$AGL(n,2) = \{(\mathbf{A},\mathbf{b}) | \mathbf{A} \in GL(n,2), \mathbf{b} \in \mathbb{F}_2^n\}.$$

From [4], the size of the linear group is

$$|GL(n,2)| = \prod_{i=0}^{n-1} (2^n - 2^i).$$

It's then simple to derive that the size of the affine group is

$$|AGL(n,2)| = 2^n \prod_{i=0}^{n-1} (2^n - 2^i). \tag{5}$$

The operation defined in [8] between any two elements $(\mathbf{A},\mathbf{b}),\ (\mathbf{C},\mathbf{d}) \in AGL(n,2)$ is

$$(\mathbf{A},\mathbf{b})(\mathbf{C},\mathbf{d}) = (\mathbf{A}\cdot\mathbf{C}, \mathbf{A}\cdot\mathbf{d} + \mathbf{b}). \tag{6}$$

The inverse of any element $(\mathbf{A},\mathbf{u})$ in $AGL(n,2)$ is

$$(\mathbf{A},\mathbf{u})^{-1} = \left(\mathbf{A}^{-1}, \mathbf{A}^{-1}\mathbf{u}\right).$$

We can easily attain that the identity of $AGL(n,2)$ is $\mathbf{e} = (\mathbf{E},\mathbf{0})$, where $\mathbf{E}$ is the identity matrix in $\mathbb{F}_2$ and $\mathbf{0}$ is the zero vector in $\mathbb{F}_2^n$.

As introduced in Section 1, the affine group acts on the Boolean functions by taking operation on X. Generally, if $g = (\mathbf{A},\mathbf{b})$ is in $AGL(n,2)$, X is an n-dimensional vector of $\mathbb{F}_2^n$. Then the action of g on X can be expressed as

$$g(X) = \mathbf{A}X + \mathbf{b}. \tag{7}$$

Elements of isomorphic groups have the same algebraic properties [19]. Two groups G and G' are isomorphic [19] if and only if there exists a bijection ϕ from G to G' that satisfies

$$\phi(a\cdot b) = \phi(a) * \phi(b), \quad \forall a,b \in G,$$

where the operation of each group works separately. The isomorphism between G and G' is generally expressed as $G \cong G'$.

Since elements in the same conjugate class have similar algebraic properties, we need to classify $AGL(n,2)$ into disjoint conjugate classes in this paper.

Definition 2.3. *[19] Two elements g_1 and g_2 of G are conjugate elements if and only if there exists an element a in G such that*

$$g_1 = a g_2 a^{-1}.$$

The conjugate relation offers a method to divide G into disjoint conjugate classes. In addition, the conjugate classification of G can also be looked as an orbital division of G under the action of itself [19]. Let p be in G. Then the conjugate class of p is represented as

$$O(p) = \{q \in G | \exists a \in G, q = apa^{-1}\}.$$

Assume G can be divided into k disjoint orbits (or conjugate classes) and p_i is the representative of the ith orbit. Then G can be written as a union of these orbits

$$G = \bigcup_{i=1}^{k} O(p_i). \tag{8}$$

Specifically, if G is a permutation group on $\{1,2,\ldots,n\}$, each element of G is a permutation and it can be represented as a multiplication of some disjoint cycles that is also called complete factorization [19].

Lemma 2.1. *[19] Let G be a permutation group whose elements are permutations of $\{1,2,\ldots,n\}$, g_1 and g_2 be conjugate elements of G. Then g_1 and g_2 have the same cycle structure.*

In more detail, suppose that the complete factorizations of g_1 and g_2 have λ_{1r} and λ_{2r} r-cycles (cycles with length r) respectively as follows,

$$g_1 = \underbrace{(a_1)\cdots(a_{\lambda_{11}})}_{\lambda_{11}} \underbrace{(b_{11},b_{12})\cdots(b_{\lambda_{12}1},b_{\lambda_{12}2})}_{\lambda_{12}} \cdots \underbrace{\overbrace{l_{11},\ldots,l_{1n}}^{n}\cdots\overbrace{(l_{\lambda_{1n}1},\ldots,l_{\lambda_{1n}n})}^{n}}_{\lambda_{1n}},$$

$$g_2 = \underbrace{(c_1)\cdots(c_{\lambda_{21}})}_{\lambda_{21}} \underbrace{(d_{11},d_{12})\cdots(d_{\lambda_{22}1},d_{\lambda_{22}2})}_{\lambda_{22}} \cdots \underbrace{\overbrace{(m_{11},\ldots,m_{1n})}^{n}\cdots\overbrace{(m_{\lambda_{2n}1},\ldots,m_{\lambda_{2n}n})}^{n}}_{\lambda_{2n}},$$

where $r \in \{1,2,\ldots,n\}$ and $0 \le \lambda_{1r}, \lambda_{2r} \le n$ for each r. The corresponding cycle structures for g_1 and g_2 are presented as follows,

$$g_1 = 1^{\lambda_{11}} 2^{\lambda_{12}} \cdots n^{\lambda_{1n}},\ g_2 = 1^{\lambda_{21}} 2^{\lambda_{22}} \cdots n^{\lambda_{2n}}. \tag{9}$$

Then g_1 and g_2 have the same cycle structure, that is to say

$$\lambda_{1r} = \lambda_{2r}, \qquad \forall r \in \{1,2,\ldots,n\}.$$

3 Isomorphism between $AGL(n,2)$ and P_n

In Section 2, formula (1) provides a method for dividing Boolean functions into disjoint affine classes under the affine group. However the study of affine equivalence classification is extremely constrained by the complex operations on the affine group.

This section provides a technique to considerably reduce the complexity by constructing isomorphism mapping from the affine group to a permutation group. Furthermore, we will analyze the conjugate classification of the permutation group in Section 3.2.

3.1 Isomorphism Mapping from $AGL(n,2)$ to P_n

Before the discussion of isomorphism mapping, we will define two operations between integer and binary vector as follows.

Definition 3.1. *Let integer i be expressed as $i = \sum_{k=1}^{n} b_k 2^{n-k}$. Then the n-dimensional binary vector of an integer i denoted by $B(i)$ is defined as an arrangement of the n-bit binary of i from high to low. It's evident that $B(i) \in \mathbb{F}_2{}^n$ and $B(i)$ can be written by*

$$B(i) = [b_1, b_2, \ldots, b_n]^T.$$

Definition 3.2. *Let an n-dimensional vector B be in $\mathbb{F}_2^n$. If $B = [b_1, b_2, \ldots, b_n]^T$, then the integer of B denoted by $I(B)$ is determined by*

$$I(B) = \sum_{i=1}^{n} b_i 2^{n-i}.$$

Definition 3.3. *Let $AGL(n, 2)$ be the affine group on $\mathbb{F}_2^n$ and $S = \{0, 1, \ldots, 2^n - 1\}$. A permutation group denoted by P_n is defined as an image of $AGL(n, 2)$ under the mapping ϕ defined as*

$$\begin{aligned} \phi : AGL(n,2) &\to P_n \\ g &\mapsto \sigma_g, \end{aligned}$$

where $g = (\mathbf{A}, \mathbf{b})$,

$$\phi(g) = \sigma_g = \begin{bmatrix} 0 & 1 & \cdots & 2^n - 1 \\ \sigma_g(0) & \sigma_g(1) & \cdots & \sigma_g(2^n - 1) \end{bmatrix}, \tag{10}$$

and

$$\sigma_g(i) = I(g(B(i))), \quad i \in S. \tag{11}$$

From (7) we get

$$\sigma_g(i) = I(\mathbf{A} \cdot B(i) + \mathbf{b}), \quad i \in S. \tag{12}$$

It is evident that P_n is a permutation group on S.

Example 1. Let $n = 3$, mapping ϕ be defined as Definition 3.3, g be element of $AGL(3, 2)$,

$$g = (\mathbf{A}, \mathbf{b}) = \left(\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right).$$

From the expression (4), we get $S = \{0, 1, \ldots, 7\}$. For any $i \in S$, if the corresponding binary vector of i is $[a_1, a_2, a_3]^T$, i.e., $B(i) = [a_1, a_2, a_3]^T$, then we can substitute g and $B(i)$ into the equation (12) and obtain

$$\begin{aligned} \sigma_g(i) &= I\left(\begin{bmatrix} 1\,0\,0 \\ 1\,1\,0 \\ 0\,0\,1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \\ &= I\begin{pmatrix} a_1 \\ a_1 + a_2 \\ a_3 + 1 \end{pmatrix}. \end{aligned}$$

Based on the above equation and the expression (10), we can get

$$\phi(g) = \sigma_g = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 7 & 6 & 5 & 4 \end{pmatrix}.$$

It can also be represented by the cycle form $\sigma_g = (0, 1)(2, 3)(4, 7)(5, 6)$. Images of the other elements in $AGL(n, 2)$ under ϕ can be calculated by the same steps.

Theorem 3.1. *Let P_n be the image of $AGL(n, 2)$ under ϕ. Then the affine group $AGL(n, 2)$ is isomorphic to P_n.*

Proof. According to the definition of isomorphic groups [19], we need to prove that $AGL(n, 2)$ and P_n satisfy the following three steps.

Step a. Guarantee ϕ is injective.

For any two elements $g_1 = (\mathbf{A}_1, \mathbf{b}_1)$, $g_2 = (\mathbf{A}_2, \mathbf{b}_2) \in AGL(n, 2)$ and $g_1 \neq g_2$, we need to assure $\phi(g_1) \neq \phi(g_2)$.

From formula (10), we have

$$\phi(g_1) = \sigma_{g_1} = \begin{bmatrix} 0 & 1 & \cdots & 2^n - 1 \\ \sigma_{g_1}(0) & \sigma_{g_1}(1) & \cdots & \sigma_{g_1}(2^n - 1) \end{bmatrix},$$

$$\phi(g_2) = \sigma_{g_2} = \begin{bmatrix} 0 & 1 & \cdots & 2^n - 1 \\ \sigma_{g_2}(0) & \sigma_{g_2}(1) & \cdots & \sigma_{g_2}(2^n - 1) \end{bmatrix}.$$

From (11), for each $i \in S$ we get

$$\begin{aligned} \sigma_{g_1}(i) &= I(g_1(B(i))), \\ \sigma_{g_2}(i) &= I(g_2(B(i))). \end{aligned}$$

If $\sigma_{g_1}(i) \neq \sigma_{g_2}(i)$ for some i, then we get the conclusion of $\phi(g_1) \neq \phi(g_2)$.

Below we prove it by absurdity. Suppose $\phi(g_1) = \phi(g_2)$, then $\sigma_{g_1}(i) = \sigma_{g_2}(i)$ for arbitrary i, which means

$$I(g_1(B(i))) = I(g_2(B(i))).$$

From Definition 3.2 we get

$$g_1(B(i)) = g_2(B(i)).$$

From (12) we get

$$\mathbf{A}_1 \cdot B(i) + \mathbf{b}_1 = \mathbf{A}_2 \cdot B(i) + \mathbf{b}_2,$$

so

$$(\mathbf{A}_1 - \mathbf{A}_2) \cdot B(i) = \mathbf{b}_2 - \mathbf{b}_1.$$

The above equation is true for any i, which means that the equation is true for any n-dimensional binary vector $B(i)$. So we can get

$$\mathbf{A}_1 - \mathbf{A}_2 = \mathbf{O}, \quad \mathbf{b}_2 - \mathbf{b}_1 = \mathbf{0}.$$

Therefore

$$(\mathbf{A}_1, \mathbf{b}_1) = (\mathbf{A}_2, \ \mathbf{b}_2).$$

That means

$$g_1 = g_2.$$

But it contradicts to the premise $g_1 \neq g_2$. Therefore $\phi(g_1) \neq \phi(g_2)$ and ϕ is injective.

Step b. Confirm ϕ is surjective.

Owing to the fact that P_n is the image of $AGL(n, 2)$ under ϕ, ϕ is surjective from $AGL(n, 2)$ to P_n.

Step c. Closure of the operations.

For any $g_1, g_2 \in AGL(n, 2)$, we need to prove

$$\phi(g_1 \cdot g_2) = \phi(g_1) * \phi(g_2). \tag{13}$$

From Definition 3.3, the left of equation (13) can be written by

$$\phi(g_1 \cdot g_2) = \sigma_{g_1 \cdot g_2}.$$

From (6), (7), (11) and (12), for each $i \in S$ we have

$$\begin{aligned}\sigma_{g_1 \cdot g_2}(i) &= I((g_1 \cdot g_2)(B(i))) \\ &= I(\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot B(i) + \mathbf{A}_1 \cdot \mathbf{b}_2 + \mathbf{b}_1). \end{aligned} \quad (14)$$

On the other hand, from Definition 3.3, the right hand side of (13) can be written as

$$\phi(g_1) * \phi(g_2) = \sigma_{g_1} * \sigma_{g_2}.$$

Based on the multiplication of permutations [19], for each i we have

$$\sigma_{g_1} * \sigma_{g_2}(i) = \sigma_{g_1}\big(\sigma_{g_2}(i)\big).$$

Similarly, from (6), (7), (11) and (12) we get

$$\begin{aligned}\sigma_{g_1}\big(\sigma_{g_2}(i)\big) &= \sigma_{g_1}(I(\mathbf{A}_2 \cdot B(i) + \mathbf{b}_2)) \\ &= I(\mathbf{A}_1 \cdot (\mathbf{A}_2 \cdot B(i) + \mathbf{b}_2) + \mathbf{b}_1) \\ &= I(\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot B(i) + \mathbf{A}_1 \cdot \mathbf{b}_2 + \mathbf{b}_1). \end{aligned} \quad (15)$$

Since the right hand sides of equations (14) and (15) are the same, we know their left hand sides are also equivalent:

$$\phi(g_1 \cdot g_2) = \phi(g_1) * \phi(g_2).$$

Thus we conclude that $AGL(n,2)$ and P_n are isomorphic groups. □

We now discuss the equivalence between two Boolean functions under the action of P_n.

Theorem 3.2. *Let $g = (\mathbf{A}, \mathbf{b})$ be in G and f, h be affine equivalent Boolean functions under the action of g, i.e., $h(X) = f(\mathbf{A}X + \mathbf{b})$. Then the ON-sets of f and h satisfies*

$$S_f = \sigma_g S_h.$$

Proof. From Definition 2.1, suppose the ON-set of h is $S_h = \{s_1, \ldots, s_k\}$, which means

$$h(s_1) = 1, \ldots, h(s_k) = 1.$$

Applying (7) and Definition 3.1, the above equations can be written as

$$\begin{aligned} h(s_1) &= f(\mathbf{A} \cdot B(s_1) + \mathbf{b}) = 1, \\ &\cdots \\ h(s_k) &= f(\mathbf{A} \cdot B(s_k) + \mathbf{b}) = 1. \end{aligned}$$

From Definition 3.2, we get the ON-set of f,

$$S_f = \{I(\mathbf{A} \cdot B(s_1) + \mathbf{b}), \ldots, I(\mathbf{A} \cdot B(s_k) + \mathbf{b})\},$$

while from Definition 3.3, we get

$$\begin{aligned}\sigma_g S_h &= \sigma_g\{s_1, \ldots, s_k\} \\ &= \{I(\mathbf{A} \cdot B(s_1) + \mathbf{b}), \ldots I(\mathbf{A} \cdot B(s_k) + \mathbf{b})\},\end{aligned}$$

therefore

$$S_f = \sigma_g S_h.$$

□

Since isomorphic groups have the same algebraic properties, from Theorem 3.2 we conclude that the equivalence of Boolean functions under the affine group can be converted to the equivalence under the permutation group P_n. So the classification of F_n under $AGL(n,2)$ is equivalent to the classification of F_n under P_n. Later we will discuss the classification of F_n under the action of P_n in Section 4.

3.2 The Conjugate Classification of P_n

In order to compute the number of orbits for F_n under P_n from formula (1), a necessary task is to count the Boolean functions fixed by each element of P_n. On the other hand, elements coming from a conjugate class maintain the same number of Boolean functions, which we will prove in Lemma 4.3. That promotes us to compute the conjugate classes of P_n.

From Section 3.1, we know that $AGL(n,2)$ is isomorphic to P_n. So the size of P_n is equal to the size of $AGL(n,2)$. From (5), we obtain

$$|P_n| = |AGL(n,2)| = 2^n \prod_{i=0}^{n-1}(2^n - 2^i).$$

The size of P_n increases exponentially with n. In the past, conjugate classes of permutation group were computed through exhaustive search. It is impractical for large sized permutation groups. In 2000, Hulpke conducted a study in [30] for conjugate classification of finite permutation groups by homomorphism mapping. He also participated in the design of GAP. In this paper we mainly utilize GAP to calculate the conjugate classes of P_n and determine a representative for each conjugate class.

Assume P_n has k conjugate classes and p_i is a representative of the ith conjugate class. Assume the orbit of p_i is denoted by $O(p_i)$. From (8), the permutation group P_n can be written as a union of these conjugate orbits

$$P_n = \bigcup_{i=1}^{k} O(p_i).$$

Based on a conclusion of [19], the centralizer of p_i is denoted as $C(p_i) = \{a | a p_i a^{-1} = p_i, a \in P_n\}$. Then the size of $O(p_i)$ can be computed by

$$|O(p_i)| = \frac{|P_n|}{|C(p_i)|}. \quad (16)$$

Let the complete factorization of p_i be

$$\begin{aligned} p_i = &\underbrace{(a_1) \cdots (a_{\lambda_{i1}})}_{\lambda_{i1}} \underbrace{(b_{11}, b_{12}) \cdots (b_{\lambda_{i2}1}, b_{\lambda_{i2}2})}_{\lambda_{i2}} \cdots \\ &\underbrace{\overbrace{(l_{11}, \ldots, l_{1\mu})}^{\mu} \cdots \overbrace{(l_{\lambda_{i\mu}1}, \ldots, l_{\lambda_{i\mu}\mu})}^{\mu}}_{\lambda_{i\mu}}, \end{aligned} \quad (17)$$

where $1 \le \mu \le 2^n$ is the largest length of all these cycles. Each element of p_i is in $S = \{0, 1, \ldots, 2^n - 1\}$ and there are

TABLE 1
N_n for $1 \le n \le 6$

n	1	2	3	4	5	6
N_n	3	5	10	32	382	15768919

λ_{ir} r-cycles in p_i. From expression (9), the cycle structure of p_i can be written as

$$p_i = 1^{\lambda_{i1}} 2^{\lambda_{i2}} \cdots \mu^{\lambda_{i\mu}}. \tag{18}$$

Owing to the fact that p_i is a permutation of S, we obtain

$$\sum_{r=1}^{\mu} r \cdot \lambda_{ir} = 2^n.$$

4 Affine Equivalence Classification of Boolean Functions

Theorem 3.1 makes a significant contribution to calculate the affine classes of Boolean functions, which makes it possible to study the classification under P_n instead of $AGL(n,2)$. In this section, we present detailed steps for the classification of Boolean functions under P_n. We first compute the number of affine classes for F_n in Section 4.1. We define a special kind of Boolean functions F_{sn}, which we denote as "severe n-variable Boolean functions", and compute the number of affine equivalence classes for the special Boolean functions in Section 4.2. Then we independently compute affine equivalence classes for F_n^m in Section 4.3.

4.1 Affine Equivalence Classification of F_n

The results of affine equivalence classification of F_n for $n \le 5$ have been worked out by Nechiporuk [20] and for $n \le 6$ by Herrison [21]. It suffices here to list the previous results of N_n in Table 1.

As discussed in Section 3.1, the classification of F_n under $AGL(n,2)$ is equivalent to the classification under P_n. Now we concentrate on counting the orbits of F_n under the action of P_n. Formula (1) will be rewritten as follows,

$$N_n = \frac{1}{|P_n|} \sum_{\sigma \in P_n} |Fix(\sigma)|. \tag{19}$$

On the basis of expression (2) and Definition 2.1, the Boolean functions fixed by permutation σ are given by

$$Fix(\sigma) = \{f \in F_n | \sigma S_f = S_f\}. \tag{20}$$

Owing to the fact that $|P_n|$ is obtained from Section 3.2, we only need to calculate $|Fix(\sigma)|$ for computing N_n. So we take the Boolean functions fixed by each permutation σ of P_n into consideration. But the size of P_n is also large when $n \ge 6$. We could not compute $|Fix(\sigma)|$ for each σ of P_n. In order to simplify the steps of computing the Boolean functions fixed by all the elements of P_n, we first illustrate through an example.

Example 2. Let $n = 3$. From expression (4), we get $S = \{0, 1, \ldots, 7\}$. If permutation $\sigma = (1,2)(3,4,5)(6,7)$ is in P_3, the cycles of σ separately will be $(0), (1,2), (3,4,5)$ and $(6,7)$. If the ON-sets of two Boolean functions f and h respectively are $S_f = \{0,1,2\}$ and $S_h = \{1,2,3\}$, we can easily get that $\sigma S_f = \{0,1,2\} = S_f$ while $\sigma S_h = \{1,2,4\} \ne S_h$.

From the above example, we obtain a rule that Boolean functions whose ON-sets are completely constructed by some cycles respectively will be invariant under the corresponding permutation. In the example, we can find that S_f can be completely constructed by cycles (0) and $(1,2)$. Nevertheless S_h cannot be completely constructed by some cycles. So we get the following lemma.

Lemma 4.1. *Suppose σ is a permutation of P_n and σ fixes a Boolean function f. Then the ON-set of f can be completely constructed by some cycles of σ.*

Proof. Assume $\sigma = 1^{\lambda_1} 2^{\lambda_2} \cdots \mu^{\lambda_\mu}$. From expression (17) the complete factorization of σ can be written as

$$\sigma = \underbrace{(a_1)\cdots(a_{\lambda_1})}_{\lambda_1} \underbrace{(b_{11}, b_{12})\cdots(b_{\lambda_2 1}, b_{\lambda_2 2})}_{\lambda_2} \cdots \underbrace{\overbrace{(l_{11}, \ldots, l_{1\mu})}^{\mu} \cdots \overbrace{(l_{\lambda_\mu 1}, \ldots, l_{\lambda_\mu \mu})}^{\mu}}_{\lambda_\mu}, \tag{21}$$

where μ is the maximum length of cycles in σ.

Suppose the ON-set of f is $S_f = \{s_1, \ldots, s_k\}$ and f is invariant under the permutation σ. i.e.,

$$\sigma S_f = S_f.$$

That is to say

$$S_f = (a_1)\cdots(a_{\lambda_1})(b_{11}, b_{12})\cdots(b_{\lambda_2 1}, b_{\lambda_2 2})\cdots (l_{11}, \ldots, l_{1\mu})\cdots(l_{\lambda_\mu 1}, \ldots, l_{\lambda_\mu \mu}) S_f.$$

From the operation of permutation [19], we derive that only elements in S_f that come from the same cycle of σ will be invariant under the permutation σ, so we conclude that elements of S_f can be completely constructed by some cycles. □

Theorem 4.2. *Suppose σ is an element of P_n and the complete factorization of σ is expressed as (21). Let $Fix(\sigma)$ denote the Boolean functions in F_n fixed by σ. Then the size of $Fix(\sigma)$ can be computed by*

$$|Fix(\sigma)| = 2^{\sum_{r=1}^{\mu} \lambda_r}. \tag{22}$$

Proof. As Lemma 4.1 states, only Boolean functions whose ON-sets are completely composed of some cycles of σ will be invariant under σ. Therefore, we only need to calculate the number of combinations of all cycles in σ.

The total number of cycles in σ is $\sum_{r=1}^{\mu} \lambda_r$. In a particular ON-set, we may choose a particular cycle or not, so there are $2^{\sum_{r=1}^{\mu} \lambda_r}$ combinations according to combinatorics, and the conclusion is attained. □

Lemma 4.3. *Suppose σ_1 and σ_2 belong to the same conjugate class of P_n, then*

$$|Fix(\sigma_1)| = |Fix(\sigma_2)|. \tag{23}$$

Proof. From Lemma 2.1 we know that the number of cycles in σ_1 is equal to the number of cycles in σ_2. From Theorem 4.2 we get the conclusion. □

In order to calculate $\sum_{\sigma \in P_n} |Fix(\sigma)|$, we only need to compute the number of Boolean functions fixed by a representative of each conjugate class and the size of each conjugate class based on the result of Lemma 4.3. Therefore, we deduce the following formula to compute N_n.

Theorem 4.4. *Suppose P_n is the permutation group that is isomorphic to $AGL(n, 2)$. P_n can be divided into k disjoint conjugate classes and p_i expressed as (18) is a representative of the ith conjugate class of P_n. Then N_n can be computed by the following formula,*

$$N_n = \sum_{i=1}^{k} \frac{1}{|C(p_i)|} \cdot 2^{\sum_{r=1}^{\mu} \lambda_{ir}}. \tag{24}$$

Proof. Suppose the size of ith conjugate class is $|O(p_i)|$. From formula (16), we get

$$|P_n| = |O(p_i)| \cdot |C(p_i)|. \tag{25}$$

Based on Lemma 4.3 we know that the numbers of Boolean functions fixed by elements of the same conjugate class are equal, so we get

$$\sum_{\sigma \in P_n} |Fix(\sigma)| = \sum_{i=1}^{k} |Fix(p_i)| \cdot |O(p_i)|. \tag{26}$$

Replace $|P_n|$ and $\sum_{\sigma \in P_n} |Fix(\sigma)|$ in (19) with the expression (25) and (26), respectively. Then N_n can be rewritten as

$$N_n = \sum_{i=1}^{k} \frac{1}{|C(p_i)|} \cdot |Fix(p_i)|.$$

On the basis of the conclusion from Theorem 4.2, we get

$$N_n = \sum_{i=1}^{k} \frac{1}{|C(p_i)|} \cdot 2^{\sum_{r=1}^{\mu} \lambda_{ir}}.$$

□

As an experiment, we count affine classes for Boolean functions with $n \leq 10$ variables. We first transform the affine group to the permutation group P_n through Definition 3.3. Then we compute the conjugate classes of P_n by GAP. Specifically, we obtain the representatives $\{p_1, p_2, \ldots, p_k\}$ of P_n and the size of their centralizers. Finally we compute the value of N_n using formula (24). The results of N_n for $1 \leq n \leq 6$ are in accordance with Table 1 and for $7 \leq n \leq 10$ are listed in Table 2.

TABLE 2
N_n for $7 \leq n \leq 10$

n	N_n
7	16224999167506438730294
8	84575066435667906978109556031081616704183639810103015118
9	37430909215886319975901838102920717810428460679428092304 28008257901850006463109702659786369725993180193348426018 5559925420090026
10	47908499187603292265858004314876730755390913279968674449 51164293957943603412316251836932288361525930004308707339 73647565874325953529940377811130038575926765372370447759 62710960690298997198405370570217712314109935763875520593 03730030873644839300959028776787896277250546257356 36

need to make a linear combination of circuit inputs to implement other functions in the same equivalence class.

This section aims to count the number of different physical circuits for severe n-variable Boolean functions that will be defined in Definition 4.1. If an n-variable Boolean function f that can be converted by an affine transformation to another Boolean function g that can be represented by less than n variables, then f would not be considered as a severe n-variable Boolean function in this section. In fact, we only need to make a linear combination of the circuit inputs of g to realize f. So f and g can share the same physical circuit. We will discuss the number of affine equivalence classes for the severe n-variable Boolean functions.

Definition 4.1. *Let F_n be the entire set of n-variable Boolean functions and f be a Boolean function of F_n. If there does not exist an affine transformation that transfers f to any Boolean function that can be represented by less than n variables, then f is a severe n-variable Boolean function. We define F_{sn} as the set of all severe n-variable Boolean functions in F_n.*

Now we discuss the affine equivalence classification of F_{sn}. It is obvious that the number of affine equivalence classes of F_{sn} is less than the number of affine equivalence classes of F_n.

Theorem 4.5. *Let N_n be the number of affine equivalence classes of the entire set of Boolean functions F_n. If we define N_{sn} as the number of affine equivalence classes of F_{sn}, then*

$$N_{sn} = N_n - N_{n-1}.$$

Proof. According to Definition 4.1, F_{sn} contains all severe n-variable Boolean functions in F_n. The entire set of n-variable Boolean functions F_n can be divided into two disjoint parts, F_{sn} and $F_n - F_{sn}$. Suppose the set of Boolean functions $F_n - F_{sn}$ can be divided into K disjoint affine equivalence classes, then the number of affine equivalence classes for F_{sn} is equal to $N_n - K$, because any Boolean function in F_{sn} is not affine equivalent to any Boolean function in $F_n - F_{sn}$. Otherwise, F_{sn} contains Boolean functions that are affine equivalent with Boolean functions represented by less than n variables. However, this would contradict the definition of F_{sn}. So the number of affine equivalence classes, N_{sn}, is equal to $N_n - K$.

Now we only need to determine the value of K. Let the entire set of $(n-1)$-variable Boolean functions F_{n-1}

4.2 Affine Equivalence Classification of F_{sn}

As introduced in Section 1, the affine classification of Boolean functions is useful in a wide range of fields such as logic synthesis. Affine equivalent Boolean functions can be implemented by the same physical circuit. If we have designed a combinational logic circuit based on a representative Boolean function of an equivalence class, we only

TABLE 3
N_{sn} for $1 \le n \le 6$

n	1	2	3	4	5	6
N_{sn}	3	2	5	22	350	15768537

be divided into N_{n-1} disjoint affine equivalence classes, and the representatives of these equivalence classes are $f_1, f_2, \ldots, f_{N_{n-1}}$ respectively.

On the one hand, $f_1, f_2, \ldots, f_{N_{n-1}}$ are all in the set $F_n - F_{sn}$, and each of them must be in different affine equivalence classes of the set $F_n - F_{sn}$. So the number of affine equivalence classes of $F_n - F_{sn}$ is no less than N_{n-1}, i.e.

$$K \ge N_{n-1}.$$

On the other hand, each Boolean function in $F_n - F_{sn}$ can be transformed by an affine transformation to a Boolean function with less than n variables. That is to say $F_n - F_{sn}$ can be transformed into a subset of F_{n-1}. So the number of affine equivalence classes of $F_n - F_{sn}$ is no more than N_{n-1}, i.e.,

$$K \le N_{n-1}.$$

From the above, we can infer that

$$K = N_{n-1}.$$

Hence

$$N_{sn} = N_n - N_{n-1}.$$

□

Using Theorem 4.5, we can obtain the number of different physical circuits for the severe n-variable Boolean functions. The results of N_{sn} for $1 \le n \le 6$ and $7 \le n \le 10$ are listed in Tables 3 and 4 respectively.

The results in Table 4 can be especially useful from an FPGA synthesis point of view, as it presents us the number of unique functions that can be realized using larger sized look-up tables (LUTs) in FPGAs. This can be useful for architectural exploration of FPGA with larger LUT sizes.

4.3 Affine Equivalence Classification of F_n^m

In this section we will discuss the affine equivalence classification of Boolean functions with different size of ON-sets respectively.

Lemma 4.6. *If f and h belong to F_n and f is affine equivalent to h, then*

$$|S_f| = |S_h|.$$

Proof. From Section 1 there exists $g = (\mathbf{A}, \mathbf{b})$ such that $h(X) = f(\mathbf{A}X + \mathbf{b})$. From Theorem 3.1, the isomorphic image of g is denoted by σ_g. Suppose the complete factorization of σ_g is shown in (21), we obtain the following from Theorem 3.2:

$$\begin{aligned} S_f &= \sigma_g S_h \\ &= (a_1)\cdots(a_{\lambda_1})(b_{11}, b_{12})\cdots(b_{\lambda_2 1}, b_{\lambda_2 2})\cdots \\ &\quad (l_{11}, \ldots, l_{1\mu})\cdots(l_{\lambda_\mu 1}, \ldots, l_{\lambda_\mu \mu})S_h. \end{aligned}$$

TABLE 4
N_{sn} for $7 \le n \le 10$

n	N_{sn}
7	162249991675064229613 75
8	845750664356679069781095560310816004791844723036642848 24
9	374309092158863199759018381029207178104284606794280923 04 280082579018499980056030590929956719150375770851867556 00 1920115317074908
10	479084991876032922658580043148767307553909132799686744 49 511642939579436034123162518369322883615259300043087073 39 736475658743259535299403778111300385384958561564841277 83 725271503982272161555593026274084818861016778620255141 29 927597648950078667016410094419452944216512920564561 0

Because the size of a set will be invariant under a permutation, so $|\sigma_g S_h| = |S_h|$. Therefore we can get

$$|S_f| = |S_h|.$$ □

Based on the result of Lemma 4.6, the method described in this section is another way to compute the affine equivalence classes of F_n. This method contains two steps. The first step roughly classifies Boolean functions into different F_n^m according to their different ON-set size. The second step classifies each F_n^m into disjoint affine equivalence classes independently.

Definition 4.2. *Let $\sigma \in P_n$. The set of Boolean functions in F_n^m fixed by σ is determined by*

$$Fix^m(\sigma) = \{f \in F_n^m | \sigma S_f = S_f\}.$$

Based on formula (19), the formula of N_n^m can be written as

$$N_n^m = \frac{1}{|P_n|}\sum_{\sigma \in P_n} |Fix^m(\sigma)|, \tag{27}$$

which leads us to calculate $|Fix^m(\sigma)|$ for each σ. The course of computing $|Fix^m(\sigma)|$ is similar to that of $|Fix(\sigma)|$. From (26), we get

$$\sum_{\sigma \in P_n} |Fix^m(\sigma)| = \sum_{i=1}^{k} |Fix^m(p_i)| \cdot |O(p_i)|. \tag{28}$$

From formulas (25) and (27) we get

$$N_n^m = \sum_{i=1}^{k} \frac{1}{|C(p_i)|} |Fix^m(p_i)|. \tag{29}$$

The process of computing N_n^m is different from computing N_n in calculating the size of Boolean functions fixed by each p_i. $|Fix^m(p_i)|$ is the number of Boolean functions fixed by p_i in F_n^m, while $|Fix(p_i)|$ is the number of Boolean functions fixed by p_i in F_n. $|Fix(p_i)|$ can be calculated straightly by formula (22) while $|Fix^m(p_i)|$ can not be directly obtained. Hence, the following paragraphs will devote our attention to computing $|Fix^m(p_i)|$.

Lemma 4.7. *Let σ be a permutation of P_n whose complete factorization is shown in (21). If a Boolean function f of F_n^m keeps invariant under σ, then the ON-set of f can be completely*

constructed by some cycles of σ. Furthermore, the total length of these cycles is equal to m.

Proof. Owing to the fact that σ fixes f, from formula (20), we get

$$S_f = (a_1)\cdots(a_{\lambda_1})(b_{11}, b_{12})\cdots(b_{\lambda_2 1}, b_{\lambda_2 2})\cdots (l_{11}, \ldots, l_{1\mu})\cdots(l_{\lambda_\mu 1}, \ldots, l_{\lambda_\mu \mu})S_f.$$

Since f is in F_n^m, we obtain

$$|S_f| = m. \tag{30}$$

From Lemma 4.1 we know that S_f can be constructed from some cycles of σ. Using equation (30), we know that the summation of lengths of all these cycles is equal to m. □

Similar to the computation of $|Fix(p_i)|$, Lemma 4.7 supplies a method to calculate the size of $Fix^m(p_i)$ by counting the combinations of cycles with total lengths of m.

Theorem 4.8. *Let σ be a permutation of P_n and the complete factorization of σ be expressed as (18). $F_\sigma(x)$ is a polynomial associated with x as follows,*

$$F_\sigma(x) = \prod_{j=1}^{\mu} {}_\sigma(1 + x^j)^{\lambda_j}$$

$$= (1+x)^{\lambda_1}\cdot(1+x^2)^{\lambda_2}\cdot\cdots\cdot(1+x^\mu)^{\lambda_\mu} \tag{31}$$

$$= \sum_{m=0}^{2^n} c_m x^m. \tag{32}$$

Then

$$|Fix^m(\sigma)| = c_m.$$

Proof. From the viewpoint of combinatorics, in the expression of $(1 + x^j)$, x^j represents that we choose a cycle of length j and 1 represents not choosing the cycle.

Before merging terms with the same degree, each term in the expansion of (31) has the form of

$$x^{1\cdot k_1 + 2\cdot k_2 + \cdots + \mu\cdot k_\mu},$$

where $k_j (1 \le j \le \mu)$ are integers between 0 to λ_j. The above expression stands for a combination of choosing k_j cycles with length j. We can find that the first term of the expansion of (31) is always equal to 1, which means not to choose any cycle in the combination, i.e., $k_j = 0$ for $1 \le j \le \mu$. If the summation $1\cdot k_1 + 2\cdot k_2 + \cdots + \mu\cdot k_\mu$ is equal to m, then the ON-sets of Boolean functions respectively composed of all the elements in these k_j cycles with length j will be invariant under the permutation σ.

After merging terms with the same degree for the expansion of (31), $F_\sigma(x)$ is expressed as (32). The coefficient of each term c_m is equal to the number of combinations separately composed of cycles with total lengths of m. So the coefficient c_m is equal to $|Fix^m(\sigma)|$. □

Theorem 4.9. *Let p_i be the representative of the ith conjugate class of P_n and be expressed as (18). $N(x)$ is a polynomial associated with x as follows. Then N_n^m is equal to the coefficient of term with degree m in $N(x)$,*

$$N(x) = \sum_{i=1}^{k} \frac{1}{|C(p_i)|} F_{p_i}(x) = \sum_{m=0}^{2^n} N_n^m x^m. \tag{33}$$

Proof. From (29) and Theorem 4.8 we can easily find the conclusion. □

TABLE 5
Example of Computing N_n^m for $n = 3$

i	p_i	$\lvert C(p_i)\rvert$	$F_{p_i}(x)$
1	()	1,344	$(1+x)^8$
2	(0,4)(1,5)(2,6)(3,7)	192	$(1+x^2)^4$
3	(4,5)(6,7)	32	$(1+x)^4(1+x^2)^2$
4	(0,4,1,5)(2,6,3,7)	16	$(1+x^4)^2$
5	(0,2)(1,3)(4,7)(5,6)	32	$(1+x^2)^4$
6	(2,3)(4,6,5,7)	8	$(1+x)^2(1+x^2)(1+x^4)$
7	(0,4,2,7)(1,5,3,6)	8	$(1+x^4)^2$
8	(2,4,6)(3,5,7)	6	$(1+x)^2(1+x^3)^2$
9	(0,1)(2,5,6,3,4,7)	6	$(1+x^2)(1+x^6)$
10	(1,2,4,3,6,7,5)	7	$(1+x)(1+x^7)$
11	(1,2,4,5,7,3,6)	7	$(1+x)(1+x^7)$

Example 3. Let $n = 3$. We get P_3 from Definition 3.3 and it can be generated by the following four generators [19]:

$$g_1 = (4,6)(5,7),\ g_2 = (2,3)(6,7),$$
$$g_3 = (1,5)(3,7),\ g_4 = (0,4)(1,5)(2,6)(3,7).$$

We can get the size of P_3 from Section 3.2,

$$|P_3| = 2^3\prod_{i=0}^{3-1}(2^3 - 2^i) = 1{,}344.$$

Then we utilize GAP to compute the conjugate classes of P_3. We can get the representatives of these conjugate classes and their centralizers. In addition, through Theorem 4.8 we can calculate the polynomial $F_{p_i}(x)$ of each representative p_i. These results are shown in Table 5.

By substituting the above results into (33) and merging terms with the same degree of x, we will get the result of $N(x)$ as follows,

$$N(x) = 1 + x + x^2 + x^3 + 2x^4 + x^5 + x^6 + x^7 + x^8.$$

According to Theorem 4.9, the numbers of affine equivalence classes for $m = 0, 1, \ldots, 8$ respectively are

$$N_n^m = [\ 1,\ 1,\ 1,\ 1,\ 2,\ 1,\ 1,\ 1,\ 1\ ].$$

Specifically for Boolean functions with On-set size $m = 4$, there are two affine equivalence classes. For other values of m, there is only one affine equivalence class.

Because the experimental results of N_n^m are large for $n > 7$, we simply list the results of N_n^m for $n = 7$ in Table 6 and for $n = 8, 9, 10$, $m = 0, \ldots, 18$ in Table 7. The entries of N_n^m for $n = 7$ are only for $m = 0, \ldots, 64$, as the results for $m = 65, \ldots, 128$ can be computed by the following equation [22]

$$N_n^m = N_n^{2^n - m}.$$

TABLE 6
N_n^m for $n = 7$ and $m = 0, \ldots, 64$

m	0	1	2	3	4	5	6	7	8	9	10
N_n^m	1	1	1	1	2	2	4	6	12	18	38
m	11	12	13	14	15	16	17	18	19	20	21
N_n^m	70	172	401	1164	3655	13817	58012	271426	132607 0	655860 6	318677 36
m	22	23	24	25	26	27	28	29	30	31	32
N_n^m	150014 159	678124 268	293291 4889	121157 95447	477910 53341	180056 507058	648376 310406	223330 376357 5	736457 190154 9	232703 829343 06	705156 788087 31
m	33	34	35	36	37	38	39	40	41	42	43
N_n^m	205091 732325 571	572964 561823 206	153865 884928 4508	397457 810435 5111	988222 158101 2960	236644 388164 50926	546087 687572 56870	121502 108907 564372	260781 201893 095358	540183 738954 764732	108035 859439 023387 4
m	44	45	46	47	48	49	50	51	52	53	54
N_n^m	208704 333010 496978 2	389579 553402 834361 4	702934 413265 529862 6	122639 266273 516438 68	206953 291455 480050 08	337882 316808 386018 79	533853 295123 175805 81	816480 571416 933119 19	120901 818738 541321 086	173368 515662 097239 778	240789 458392 144849 379
m	55	56	57	58	59	60	61	62	63	64	
N_n^m	323971 110711 092187 340	422319 313437 888512 349	533455 800627 039641 429	653023 308506 502861 661	774773 255214 779700 776	890989 099297 538271 351	993233 630159 061645 670	107333 302687 374672 7909	112444 406821 703577 0074	114201 348833 717296 9272	

TABLE 7
N_n^m for $n = 8, 9, 10$ and $m = 0, \ldots, 18$

n \ m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
8	1	1	1	1	2	2	4	6	12	19	42	82	217	577	1987	8063	43060	290679	443907
9	1	1	1	1	2	2	4	6	12	19	43	86	234	647	2370	10545	64845	545683	6384556
10	1	1	1	1	2	2	4	6	12	19	43	87	239	668	2483	11306	72057	645,302	8468295

In addition, the experimental results are also consistent with the existence rule of N_n^m found by Strazdins in 1997 [22].

5 Conclusion

This paper offers two formulas to calculate the number of affine equivalence classes for the complete set of Boolean functions. Additionally, we conducted a study of the affine equivalence classification for Boolean functions with different ON-set size separately.

A key step in our method is creating isomorphism mapping from the affine group to a permutation group, which simplifies the process of computing greatly. Subsequently, the classification of Boolean functions under the affine group is transformed to classification under the permutation group. In the process of computing the Boolean functions fixed by elements of P_n, conjugate classification of P_n also greatly simplifies the calculation. This scheme provides great progress on the affine equivalence classification of Boolean functions and makes it possible to calculate N_n and N_n^m for $n \leq 10$ respectively.

In addition, the affine equivalence classification of Boolean functions has wide applications in the design of circuits [31] and cryptography [32] with excellent properties. One can design PLBs based on the representatives of affine equivalence classes mentioned in this paper. Owing to the fact that affine equivalent Boolean functions have many similar cryptography properties, the techniques given for counting the numbers of affine equivalence classes under the affine group $AGL(n, 2)$ may be applied to counting the numbers of affine equivalence classes of Reed-Muller codes under the affine group.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61272175, 61572109.

References

[1] P. Wilson, *Design Recipes for FPGAs: Using Verilog VHDL*, Newnes, Linacre House, Jordan Hill, Oxford OX2 8DP, 2011.

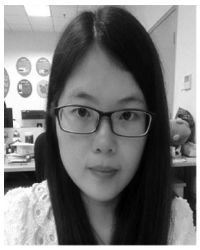
[2] L. Qu, Y. Tan, C. Tan, and C. Li, "Constructing differentially 4-uniform permutations over via the switching method," *IEEE Trans. Inf. Theory*, vol. 59, no. 7, pp. 4675–4686, Jul. 2013.

[3] D. Janković, R. S. Stanković, and C. Moraga, "Optimization of polynomial expressions by using the extended dual polarity," *IEEE Trans. Comput.*, vol. 58, no. 12, pp. 1710–1725, Dec. 2009.

[4] X. D. Hou, "$GL(m, 2)$ acting on $R(r, m)/R(r-1, m)$," *Discr. Math.*, vol. 149, no. 1, pp. 99–122, 1996.

[5] C.-C. Tsai and M. Marek-Sadowska, "Boolean functions classification via fixed polarity Reed-Muller forms," *IEEE Trans. Comput.*, vol. 46, no. 2, pp. 173–186, Feb. 1997.

[6] B. J. Falkowski, "A comment on "Boolean functions classification via fixed polarity Reed-Muller forms"," *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 1067–1069, Aug. 2006.
[7] C. Fontaine, "On some cosets of the first-order Reed-Muller code with high minimum weight," *IEEE Trans. Inf. Theory*, vol. 45, no. 4, pp. 1237–1243, May 1999.
[8] J. A. Maiorana, "A classification of the cosets of the Reed-Muller code $R(1,6)$," *Math. Comput.*, vol. 57, no. 195, pp. 403–414, 1991.
[9] 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 *Proc. 32nd Int. Colloq. Automata, Lang. Program.*, 2005, pp. 324–334.
[10] Q. Meng, M. Yang, H. Zhang, and Y. Liu, "Analysis of affinely equivalent Boolean functions," in *Proc. 1st Int. Workshop, BFCA*, 2005, vol. 5, pp. 105–114.
[11] X. Hou, "Classification of cosets of the Reed-Muller code $R(m-3,m)$," *Discr. Math.*, vol. 128, no. 1, pp. 203–224, 1994.
[12] E. R. Berlekamp and L. R. Welch, "Weight distributions of the cosets of the $(32,6)$ Reed-Muller code," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 203–207, Jan. 1972.
[13] P. Stănică, "Affine equivalence of quartic monomial rotation symmetric Boolean functions in prime power dimension," *Inform. Sci.*, vol. 314, pp. 212–224, 2015.
[14] T. W. Cusick and Y. Cheon, "Affine equivalence of quartic homogeneous rotation symmetric Boolean functions," *Inform. Sci.*, vol. 259, pp. 192–211, 2014.
[15] N. N. Biswas, "On identification of totally symmetric Boolean functions," *IEEE Trans. Comput.*, vol. C-19, no. 7, pp. 645–648, Jul. 1970.
[16] B. M. E. Moret, M. G. Thomason, and R. C. Gonzalez, "Symmetric and threshold Boolean functions are exhaustive," *IEEE Trans. Comput.*, vol. C-32, no. 12, pp. 1211–1212, Dec. 1983.
[17] G. Gao, X. Zhang, W. Liu, and C. Carlet, "Constructions of quadratic and cubic rotation symmetric bent functions," *IEEE Trans. Inf. Theory*, vol. 58, no. 7, pp. 4908–4913, Jul. 2012.
[18] A. Biryukov, C. De Canniere, A. Braeken, and B. Preneel, "A toolbox for cryptanalysis: Linear and affine equivalence algorithms," in *Proc. Adv. CryptologyEUROCRYPT*, 2003, pp. 33–50.
[19] J. J. Rotman, *Advanced Modern Algebra*. Providence, RI, USA: American Mathematical Soc., 2010, vol. 114.
[20] E. Nechiporuk, "On the synthesis of networks using linear transformations of argument variables," *Nauk SSSR*, vol. 123, pp. 610–612, Apr. 1958.
[21] M. A. Harrison, "On the classification of Boolean functions by the general linear and affine groups," *J. Soc. Ind. Appl. Math.*, vol. 12, no. 2, pp. 285–299, 1964.
[22] I. Strazdins, "Universal affine classification of Boolean functions," *Acta Appl. Math.*, vol. 46, no. 2, pp. 147–167, 1997.
[23] H. A. Curtis, "Simplified decomposition of Boolean functions," *IEEE Trans. Comput.*, vol. C-25, no. 10, pp. 1033–1044, Oct. 1976.
[24] D. Cheng and X. Xu, "Bi-decomposition of multi-valued logical functions and its applications," *Automatica*, vol. 49, no. 7, pp. 1979–1985, 2013.
[25] D. Slepian, "On the number of symmetry types of Boolean functions of n variables," *Canadian J. Math*, vol. 5, no. 2, pp. 185–193, 1953.
[26] Y. Hu, V. Shih, R. Majumdar, and L. He, "Exploiting symmetries to speed up SAT-based Boolean matching for logic synthesis of FPGAs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1751–1760, Oct. 2008.
[27] C. Lai, J. R. Jiang, and K. Wang, "Boolean matching of function vectors with strengthened learning," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Des.*, Nov. 2010, pp. 596–601.
[28] J. Cong and Y. Y. Hwang, "Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1077–1090, Sep. 2001.
[29] GAP – Groups, Algorithms, Program., Version 4.7.8, The GAP Group, 2015. [Online]. Available: http://www.gap-system.org
[30] A. Hulpke, "Conjugacy classes in finite permutation groups via homomorphic images," *Math. Comput.*, vol. 69, no. 232, pp. 1633–1651, 2000.
[31] C. R. Edwards, "The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis," *IEEE Trans. Comput.*, vol. C-24, no. 1, pp. 48–62, Jan. 1975.
[32] F. Luccio and L. Pagli, "On a new Boolean function with applications," *IEEE Trans. Comput.*, vol. 48, no. 3, pp. 296–310, Mar. 1999.



Yan Zhang received the BS degree in mathematics and applied mathematics from Sichuan Normal University, China. She is currently working toward the PhD degree in the School of Computer Science and Engineering, University of Electronic Science and Technology of China, China. Her research interests include logic synthesis, optimization problem, and machine learning.

Guowu Yang received the BS degree from the University of Science and Technology of China in 1989, the MS degree from the Wuhan University of Technology in 1994, and the PhD degree in electrical and computer engineering from Portland State University in 2005. He was at the Wuhan University of Technology from 1989 to 2001 and at Portland State University from 2005 to 2006. He is currently a full professor at the University of Electronic Science and Technology of China. His research interests include verification, logic synthesis, quantum computing, and machine learning. He has published more than 100 journal and conference papers.

William N. N. Hung received the BS and MS degrees from the University of Texas at Austin in 1994 and 1997, respectively, and the PhD degree from Portland State University in 2002, all in electrical and computer engineering. He is currently a principal engineer at Synopsys in Mountain View, California, leading technological innovations on constraint-based verification and hardware accelerated verification such as emulation and prototyping. He was with several high-tech companies including Intel, Synplicity, and Synopsys. He has more than 19 years of industrial R&D experience, published more than 80 journal and conference papers, and is the inventor of numerous patents. He is currently an associate editor of *IEEE Transactions on Computer-Aided Design*. He also served on the technical program committees of many conferences such as DAC, DATE, ICCD, CAV, FMCAD, CEC, WCCI, etc. He was the chair of Quantum Computing Task Force under the Emergent Technologies Technical Committee of the IEEE Computational Intelligence Society. He also served as a cochair of the Logic and Circuit Track in the technical program committee of ICCD.

Juling Zhang is currently working toward the PhD degree at the University of Electronic Science and Technology of China. Her research interests include Boolean matching and information security risk assessment.