2023 Design, Automation & Test in Europe Conference (DATE 2023)

Rethinking NPN Classification from Face and Point Characteristics of Boolean Functions

Jiaxi Zhang[1*], Shenggen Zheng[2*], Liwei Ni[2,3*], Huawei Li[3,4] and Guojie Luo[1]
[1]Center for Energy-Efficient Computing and Applications, Peking University, Beijing, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]University of Chinese Academy of Sciences, Beijing, China
[4]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
Email: zhangjiaxi@pku.edu.cn, zhengshg@pcl.ac.cn, nlwmode@gmail.com, lihuawei@ict.ac.cn, gluo@pku.edu.cn

***Abstract*—NPN classification is an essential problem in the design and verification of digital circuits. Most existing works explored variable symmetries and cofactor signatures to develop their classification methods. However, cofactor signatures only consider the face characteristics of Boolean functions. In this paper, we propose a new NPN classifier using both face and point characteristics of Boolean functions, including cofactor, influence, and sensitivity. The new method brings a new perspective to the classification of Boolean functions. The classifier only needs to compute some signatures, and the equality of corresponding signatures is a prerequisite for NPN equivalence. Therefore, these signatures can be directly used for NPN classification, thus avoiding the exhaustive transformation enumeration. The experiments show that the proposed NPN classifier gains better NPN classification accuracy with comparable speed.**

***Index Terms*—NPN classifier, cofactor, influence, sensitivity**

I. Introduction

Classification of Boolean functions groups a set of Boolean functions into equivalent classes. Negation-Permutation-Negation (NPN) equivalence is the most frequently used one regarding the transformations of input negation, input permutation, and output negation. It has significant applications in logic synthesis, technology mapping, and verification. Boolean functions that are NPN equivalent define an NPN equivalence class.

Boolean matching and classification have been widely studied in the past decades. Previous methods could be classified into three categories: algorithms based on Boolean satisfiability (SAT), algorithms utilizing search with signature pruning, and algorithms based on canonical forms. SAT-based methods can handle Boolean functions with a large number of input variables, but they are slow because of the NP-completeness [1]. Algorithms utilizing search with signature pruning are usually used for pair-wise matching. A signature of a Boolean function is a compact representation that characterizes some intrinsic structures and serves as a necessary condition for Boolean matching. Signatures derived from row sums [2] and cofactor [2]–[5] are two common types that have been explored extensively. Zhang *et al.* [6] explore sensitivity signatures for further pruning. Spectra like Walsh [7] and Haar [8] have also been used as signatures for Boolean matching.

Algorithms based on canonical forms are best manifested in NPN classification. A canonical form is a representative of NPN equivalent Boolean functions, and two functions match if and only if their canonical forms are identical. They work by designing a complete and unique canonical form of the Boolean functions and then try computing the canonical form for each Boolean function to check for NPN equivalence. Many works construct canonical form considering phase assignment [9], variable symmetries [10], [11] and high-order symmetries [5], [12], [13]. Abdollahi *et al.* [3] utilize cofactor signatures to define signature-based canonical forms. Zhou *et al.* [14] combine cofactor signatures and different types of symmetries to design hybrid canonical forms.

From the hypercube view of the Boolean functions, the cofactors only include the face characteristics of the hypercube. When designing an NPN classifier, it is challenging to ensure classification accuracy if only the face characteristics are considered. Exhaustive transformation enumerations are always required to improve classification accuracy or achieve exact classification. In this paper, we develop a new NPN classification method that considers both face characteristics and point characteristics of Boolean functions, which are sensitivity [15] and influence [16]. Intuitively, a cofactor considers a face of the hypercube and counts how many points take the same value in the face, while sensitivity considers a point of the hypercube and counts up how many adjacent points take a different value with the point. The main contributions are summarized as the following:

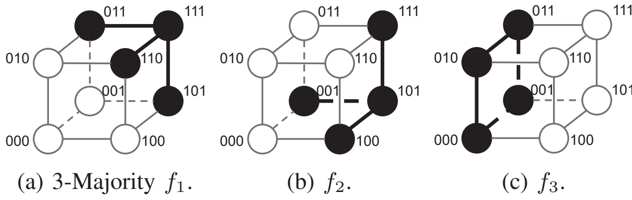
- We introduce Boolean sensitivity and influence into NPN classification. We deeply analyze the relationship between these two characteristics and the cofactor to illustrate their different properties. They bring a new perspective to the NPN classification.
- We design some signature vectors based on these two characteristics and give some proofs for these vectors to guide NPN equivalence checking.
- We develop a new NPN classifier based on these signature vectors. After signatures computation, the classifier can directly get NPN classes without transformation enumeration. Meanwhile, the classifier has stable runtime; it does not suffer from variance for different Boolean function sets.

II. Face and Point Characteristics

This section shows some commonly used notations and concepts and then introduces the three characteristics used in our NPN classifier.

*These authors contributed equally to this work.

978-3-9819263-7-8/DATE23/© 2023 EDAA



(a) 3-Majority f_1. (b) f_2. (c) f_3.

Fig. 1: Hypercubes of three 3-variable Boolean functions. 3-majority logic f_1 and f_2 are NPN equivalent, and their induced subgraphs(bolded) are isomorphic. f_2 and f_3 are not NPN equivalent, and their induced subgraphs are non-isomorphic.

A. Notations and Basic Concepts

An n-variable Boolean function, $f(X) : \{0,1\}^n \to \{0,1\}$, maps a binary word $X = (x_1, x_2, \cdots, x_n)$ of width n into a single binary value. A variable x_i or its complement $\bar{x_i}$ in f is called a literal, and i denoted the index. A *minterm* is a conjunction of n literals of different variables.

Truth table $T(f)$, a binary string of 2^n bits, is a commonly-used representation of Boolean function f. The i-th bit of $T(f)$ is equal to $f((i)_2)$, where $(i)_2$ is the little-endian binary code of integer i. A subgraph of a hypercube can also represent a Boolean function. The hypercube Q_n is a graph of order 2^n, whose vertices include all minterms and whose edges connect vertices that differ in exactly one variable. Boolean function f can be represented as the induced subgraph of Q_n. Fig. 1a shows the hypercube Q_3, and the $\bullet$ nodes, as well as edges between these nodes, construct the induced subgraph of a 3-Majority logic.

An NP transformation of a Boolean function is composed of variables negations and permutations. Negation, denoted as $\neg$, replaces a variable by its complement (e.g., $x_1 \to \neg x_1$). Besides, we denote $(\neg)$ as a selective negation. For simplicity, we denote $(\neg)X = (\neg)x_1(\neg)x_2 \cdots (\neg)x_n$ to describe the selective negation of the word X (e.g., for $(\neg)(x_1x_2) = x_1\overline{x_2}$, we have $(\neg)x_1 = x_1$ and $(\neg)x_2 = \overline{x_2}$). Permutation, denoted as π, is a reorder of variables (e.g., $\pi(x_1x_2) = x_2x_1$). Besides, we denote $X_{(i)}$ as the i-th minterm, and X^i as negating the i-th variable in X.

The *satisfy count* of a function f is the number of nodes in the induced subgraph, denoted as $|f|$. An n-input f is called *balanced* if $|f| = |\bar{f}| = 2^{n-1}$. Fig. 1 shows three balanced 3-inputs Boolean functions.

B. Face Characteristic – Cofactor

The cofactor is derived from a Boolean function by substituting constant values for some input variables, and it has been well explored in Boolean matching and classification [2]–[5], [11] in the past decades.

Definition 1. *(cofactor)*. The *cofactor* of f with respect to literal x_i and $\overline{x_i}$ are denoted as $f_{x_i=1}$ and $f_{x_i=0}$, respectively.

Definition 2. *(cofactor signatures)*. The *cofactor signatures* are the satisfy count and the satisfy counts of the cofactors. Particularly, the satisfy count of a Boolean function is called the *0-ary cofactor signature*. The *1-ary cofactor signatures* are a set of satisfy counts of the cofactors with respect to each literal. The *higher-ary cofactor signatures* (a.k.a higher-order cofactor signatures) are composed of the satisfy counts of the cofactors with respect to a set of variables.

A face in the hypercube represents a cofactor with respect to one variable or multiple variables of a Boolean function. In Fig. 2a, the blue face is $f_{x_1=0}$. Moreover, the blue face in Fig. 2b represents the cofactor $f_{x_1x_2=00}$. Therefore, cofactor signatures are the number of 1-minterms on the face in the hypercube. They contain the **face characteristics** of Boolean functions. Higher-ary cofactor signatures correspond to lower-ary faces. Symmetry can be deduced by cofactor, which can also be seen as a property of faces in a Boolean function.

C. Point Characteristic – Sensitivity

Sensitivity considers a hypercube point and counts how many adjacent points take a different value from this point.

Definition 3. (*sensitive*). Given a word X, Boolean function f is *sensitive* at literal x_i for the word X, if the output flips when the x_i flips (i.e., $f(X) \neq f(X^i)$).

Take the word *X=100* for the 3-Majority logic f_1 in Fig. 1a as an example. If the second bit flips, f_1 will also flip. Thus, f_1 is sensitive at x_2 for the word *100*.

Definition 4. (*sensitivity*). The *sensitivity* of f on the given word X, a.k.a. *local sensitivity*, is the number of input literals that are sensitive for X: $sen(f, X) = |i : f(X) \neq f(X^i)|$. Further we have the *sensitivity* of f as $sen(f) = \max\{sen(f, X) : X \in \{0,1\}^n\}$. The *0-sensitivity* of f as $sen^0(f) = \max\{sen(f, X) : X \in \{0,1\}^n, f(X) = 0\}$ and the *1-sensitivity* of f as $sen^1(f) = \max\{sen(f, X) : X \in \{0,1\}^n, f(X) = 1\}$.

Sensitivity reflects the relation between neighboring points. In Fig. 2c, the local sensitivity $sen(f, 110)$ indicates the property between the blue points and their neighboring points. Therefore, sensitivity signatures contain the **point characteristics** of a Boolean function.

D. Point-Face Characteristic – Influence

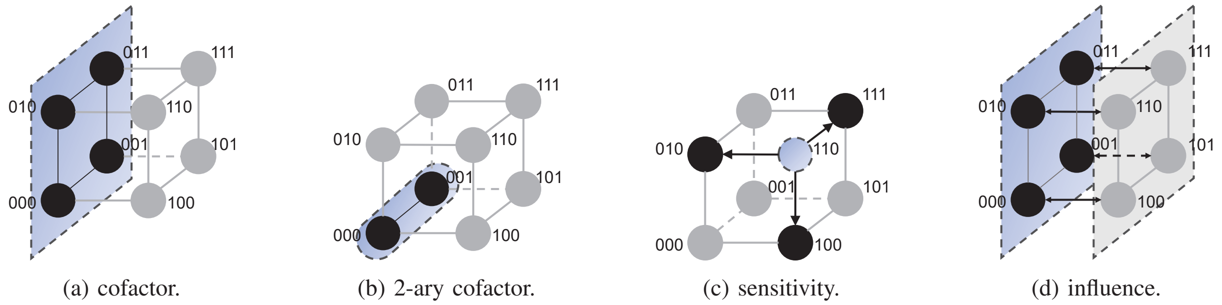
Influence is the probability that points in one face have a different value than the corresponding points in the opposite face.

Definition 5. (*influence*). The *influence* of input x_i on Boolean function f is defined to be the probability that f is sensitive at x_i for a word X: $inf(f, i) = \underset{X\in\{0,1\}^n}{Pr}[f(X) \neq f(X^i)] = \frac{1}{2^n}|f(X) \neq f(X^i) : X \in \{0,1\}^n|$ [1]. Furthermore, the *total influence* of f can be further defined as $inf(f) = \sum_{i=1}^{n} inf(f, i)$.

Boolean influence derives from the sensitive definition. The sensitive property captures the relation of neighboring points. Influence indicates the sensitive properties between two opposite faces. It calculates the number of minterms in a face with different values compared to the opposite face. In Fig. 2d, the influence of variable x_1 reflects on the blue and the grey face. Therefore, influence signatures contain the **point-face characteristics** of a Boolean function.

From the above analysis, it is evident that the structural information of a Boolean function contained in the influence

[1]For convenience, we denote in the rest of this paper that $inf(f,i) = \frac{1}{2}|f(X) \neq f(X^i) : X \in \{0,1\}^n|$. It is clear that $|f(X) \neq f(X^i) : X \in 0, 1^n|$ is an even integer. For example, if $f(000) \neq f(100)$, then $f(100) \neq f(000)$. Once the factor $\frac{1}{2^n}$ is removed, $inf(f,i)$ will be an integer. It is easier to compute integers than floating point numbers.



(a) cofactor. (b) 2-ary cofactor. (c) sensitivity. (d) influence.

Fig. 2: Cofactor, Influence and Sensitivity Signatures Visualization.

and sensitivity signatures is of a different perspective compared to the cofactor signatures. Most previous works only considered the role of cofactor signatures in constructing the NPN classification method while ignoring the influence and sensitivity features. These two characteristics have great potential to guide NPN classification. We will explore them in the following two sections.

III. Signature Vectors and NPN Equivalence

In this section, we first design several signature vectors from the point and face characteristics of Boolean functions and then give some theorems and their proofs of these signature vectors and NPN equivalence.

A. Signature Vectors

We can further define several signature vectors from the definition of the face and point characteristics.

Definition 6. *(ordered cofactor vector)*. The *1-ary ordered cofactor vector* of an n-variable Boolean function f is $OCV_1 = \{|f_{z=v}| : z \in \{x_1, x_2, ..., x_n\}, v \in \{0,1\}\}_{\leq}$, where $\{\cdot\}_{\leq}$ is the sorted multi-set (of all cofactors' satisfy counts) in non-decreasing order and $|OCV_1| = 2n$. Furthermore, the *ℓ-ary ordered cofactor vector* of an n-variable Boolean function f is the sorted multi-set $OCV_\ell = \{|f_{z=v}| : z \in \{x_1, x_2, ..., x_n\}_\ell, v \in \{0,1\}^\ell\}_{\leq}$, where $Z_\ell = \{z \subseteq Z : |z| = \ell\}$ and $|co\, f_\ell| = \binom{n}{\ell} \cdot 2^\ell$.

Definition 7. (*ordered influence vector*). The *ordered influence vector* of Boolean function f is $OIV(f) = \{inf(f,z) : z \in \{x_1, x_2, ..., x_n\}\}_{\leq}$.

Definition 8. (*ordered sensitivity vector*). For all words X in truth table $T(f)$, we denote the sorted multi-set $OSV(f) = \{sen(f,X) : X \in \{0,1\}^n\}_{\leq}$ as the *ordered sensitivity vector* of function f. Similarly, we can define $OSV^0(f) = \{sen(f,X) : X \in \{0,1\}^n, f(X) = 0\}_{\leq}$ as the *ordered 0-sensitivity vector* and $OSV^1(f) = \{sen(f,X) : X \in \{0,1\}^n, f(X) = 1\}_{\leq}$ as the *ordered 1-sensitivity vector*. Obviously, we have $OSV(f) = \{OSV^1(f), OSV^0(f)\}_{\leq}$.

Definition 9. (*sensitivity distance*). *Hamming distance* $h(X,Y)$ is a metric for comparing two binary strings X and Y. It is the number of bit positions in which X and Y differ. The *sensitivity distance* is defined as the Hamming distance of two words X and Y that have the same local sensitivity, denoted as a tuple $\{\langle sen(f,X), sen(f,Y), h(X,Y)\rangle | sen(f,X){=}sen(f,Y)\}$.

Definition 10. (*ordered sensitivity distance vector*). For a given n-variable Boolean function f, we define $OSDV(f) = (\sigma_0, \sigma_1, \cdots, \sigma_n)$, where $\sigma_i = (\delta_{i1}, \delta_{i2}, \cdots, \delta_{in})$ and $\delta_{ij} = |\{(X,Y) : sen(f,X){=}sen(f,Y){=}i,\ h(X,Y){=}j, \text{ and } X < Y\}|$. δ_{ij} is the number of pairs (X,Y) with sensitivity i and distance $h(X,Y){=}j$. Similarly, we can define $OSDV^1(f)$ and $OSDV^0(f)$ based on Definition 4.

TABLE I: Examples of different signature vectors.

Signatures	f_1 in Fig. 1a	f_3 in Fig. 1c
OCV_1	(1,1,1,3,3,3)	(0,2,2,2,2,4)
OCV_2	(0,0,0,1,1,1,1,1,1,2,2,2)	(0,0,0,0,1,1,1,1,2,2,2,2)
OIV	(2,2,2)	(0,0,4)
OSV^1	(0,2,2,2)	(1,1,1,1)
OSV^0	(0,2,2,2)	(1,1,1,1)
OSV	(0,0,2,2,2,2,2,2)	(1,1,1,1,1,1,1,1)
$OSDV^1$	(0,0,0,0,0,0,0,3,0,0,0,0)	(0,0,0,4,2,0,0,0,0,0,0,0)
$OSDV$	(0,0,1,0,0,0,6,6,3,0,0,0)	(0,0,0,12,12,4,0,0,0,0,0,0)

Table I shows some examples of different signature vectors of two 3-input Boolean functions in Fig. 1. We will use this example to explain further how to get $OSDV^1$. For f_1 in Fig. 1a, there is no word X such that $sen^1(f,X)$=1 and $sen^1(f,X)$=3. Moreover, only one word X=111 satisfies $sen^1(f,X)$=0. Thus, $\sigma_0{=}\sigma_1{=}\sigma_3{=}(0,0,0)$. For the three words that local sensitivity equal to 2, 011, 101, and 110, we can obtain δ_{21}=0, δ_{22}=3 and δ_{23}=0 according to Definition 10. In summary, $OSDV^1(f_1){=}(0,0,0,0,0,0,0,3,0,0,0,0)$.

B. Signature Vectors and NPN Equivalence

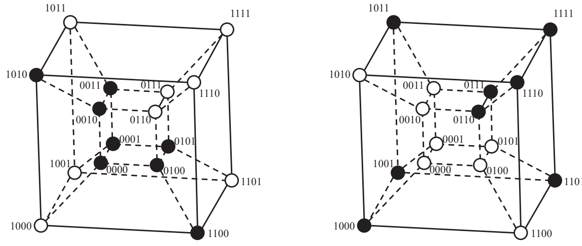
Previous work [3] has demonstrated that equality of OCV_ℓ is a prerequisite for NPN equivalence, so we only consider OIV, OSV, and $OSDV$ in this subsection. The sensitive property of Boolean functions inherently considers the polarity of the output (output negation). For unbalanced Boolean functions, we only need to consider input phase assignment and input variables order. That is to say, NPN equivalence is simplified to the PN equivalence problem. Therefore, we only need to concentrate on PN equivalence to give the following theorems.

Lemma 1. *If Boolean function f is PN-equivalent to Boolean function g, that is $f(\pi((\neg)x)) = g(x)$, then for any input i, we have $inf(f,\pi((\neg)i)) = inf(g,i)$.*

Proof. *If f is PN-equivalent to g, it is clear that $inf(g,i) = \frac{1}{2}|g(X) \neq g(X^i) : X \in \{0,1\}^n| = \frac{1}{2}|f(\pi((\neg)X) \neq f(\pi((\neg)X^i)) : X \in \{0,1\}^n)| = inf(f,\pi((\neg)i))$.*

Theorem 1. *Two PN-equivalent functions f and g have the same ordered influence vector: if f is PN-equivalent to g, then $OIV(f) = OIV(g)$.*

Proof. *According to Definition 5 and Lemma 1, it is clear that negation of a variable will not change its influence and*



(a) $OSV^1(f) = \{1, 1, 1, 1, 2, 2, 3, 3\}$ $OSV^0(f) = \{0, 1, 2, 2, 2, 2, 2, 3\}$. (b) $OSV^1(g) = \{0, 1, 2, 2, 2, 2, 2, 3\}$ $OSV^0(g) = \{1, 1, 1, 1, 2, 2, 3, 3\}$.

Fig. 3: Two NPN equivalent balanced Boolean functions. $OSV^1(f) = OSV^0(g)$ and $OSV^0(f) = OSV^1(g)$ in these two functions.

$inf(f, \pi(i)) = inf(g, i)$ under permutation of variable x_i. Therefore, the ordered OIV will not change if two functions f and g are PN-equivalent to each other.

Lemma 2. *If Boolean function f is PN-equivalent to Boolean function g, that is $f(\pi((\neg)x) = g(x)$, then for any input X, we have $sen(f, \pi((\neg)X)) = sen(g, X)$.*

Proof. *Since $f(\pi((\neg)x_1, (\neg)x_2, \cdots, (\neg)x_n)) = g(x_1, x_2, \cdots, x_n)$, it is clear that if f is sensitive at index i for input word $\pi((\neg)X)$, then g is sensitive at index j for input word X where $\pi(j)$=i. The sensitive property inherently considers negations of the variables so that flipping an input can not change anything of a Boolean function's sensitivity.*

For example, let $f(x)$ be a 4-bit Boolean function, permutation $\pi(x_1x_2x_3x_4) = x_4x_3x_2x_1$, selective negation $(\neg)x_1x_2x_3x_4 = \overline{x_1}x_2\overline{x_3}x_4$, and $f(\pi((\neg)x_1x_2x_3x_4))$ = $f(x_4, \overline{x_3}, x_2, \overline{x_1})$ = $g(x_1, x_2, x_3, x_4)$. Assume that f is sensitive at index 2 for word $(x_4, \overline{x_3}, x_2, \overline{x_1})$ (i.e., at where $\overline{x_3}$ locates), we have $g(x_1, x_2, x_3, x_4) = f(x_4, \overline{x_3}, x_2, \overline{x_1})$ = $\neg f(x_4, x_3, x_2, \overline{x_1})$ = $\neg g(x_1, x_2, \overline{x_3}, x_4)$. Function g is sensitive at index 3=$\pi(2)$ for input word (x_1, x_2, x_3, x_4).

Therefore, for any X, $sen(f, \pi((\neg)X)) = sen(g, X)$.

Theorem 2. *Two PN-equivalent unbalanced functions f and g have the same ordered sensitivity vector, ordered 0-sensitivity vector, and ordered 1-sensitivity vector: if f is PN-equivalent to g, then $(OSV, OSV^0, OSV^1)(f) = (OSV, OSV^0, OSV^1)(g)$.*

A similar theorem has been proved by Zhang *et al.* [6]. However, they ignored balanced Boolean functions. The two Boolean functions in Fig. 3 are NPN equivalent. For these two functions, $OSV^1(f)$=$OSV^0(g)$ and $OSV^0(f)$=$OSV^1(g)$. We split 1-sensitivity and 0-sensitivity to handle balanced Boolean functions. Given two NPN-equivalent unbalanced Boolean functions f and g, it is easy to check whether g is transformed from f by negation. We can use the 0-ary cofactor of the functions to find out the potential negation. However, if f and g are balanced, it cannot find out the potential negation just by using their 0-ary cofactor. In such cases, we calculate both 1-sensitivity and 0-sensitivity of the functions to deal with the potential negation.

Theorem 3. *Given two balanced Boolean functions f and g, if f is NPN-equivalent to g, then $OSV^1(f)$=$OSV^1(g)$, $OSV^0(f)$=$OSV^0(g)$ or $OSV^1(f)$=$OSV^0(g)$, $OSV^0(f)$ =$OSV^1(g)$.*

Proof. *According to Theorem 2, if f PN-equivalent to g, that is $f(\pi((\neg)x)) = g(x)$, we have $OSV^1(f) = OSV^1(g)$ and $OSV^0(f) = OSV^0(g)$. If $\neg f(\pi((\neg)x)) = g(x)$, it is clear that $OSV^0(f) = OSV^1(g)$ and $OSV^1(f) = OSV^0(g)$.*

In order to deal with balanced Boolean functions in our algorithms, if $OSV^1(f)$ is smaller than $OSV^0(f)$, we will exchange these two vectors and always put the smaller one in $OSV^0(f)$.

Lemma 3. *Given two inputs X and Y, if f is PN-equivalent to g, then $\langle sen(f, \pi((\neg)X))$, $sen(f, \pi((\neg)Y))$, $h(\pi((\neg)X), \pi((\neg)Y))\rangle$=$\langle sen(g, X)$, $sen(g, Y)$, $h(X, Y)\rangle$.*

Proof. *It is easy to see that $h(X, Y) = h(\pi((\neg)X, (\neg)Y))$. According to Lemma 2 and Definition 9, the theorem holds.*

Theorem 4. *Two PN-equivalent unbalanced functions f and g have the same ordered sensitivity distance vector, ordered 0-sensitivity distance vector, and ordered 1-sensitivity distance vector: if f is PN-equivalent to g, then $(OSDV, OSDV^0, OSDV^1)(f) = (OSDV, OSDV^0, OSDV^1)(g)$. For two balanced Boolean functions f and g, if f is NPN-equivalent to g, then $OSDV^1(f) = OSDV^1(g)$, $OSDV^0(f) = OSDV^0(g)$ or $OSDV^1(f) = OSDV^0(g)$, $OSDV^0(f) = OSDV^1(g)$.*

Proof. *Given two inputs x and y, if f is PN-equivalent to g, according to Lemma 3, we have that $\langle h(X, Y)$, $sen(f, X)$, $sen(f, Y)\rangle$=$\langle h(X', Y')$, $sen(g, X')$, $sen(g, Y')\rangle$, where X' and Y' is the NP transformation of X and Y. According to Definition 9 and Definition 10, it is clear that for unbalanced Boolean functions f and g, we have $(OSDV, OSDV^0, OSDV^1)(f)$ = $(OSDV, OSDV^0, OSDV^1)(g)$. Similar to the proof of Theorem 3, we can prove the results of the rest of the theorem for balanced Boolean functions.*

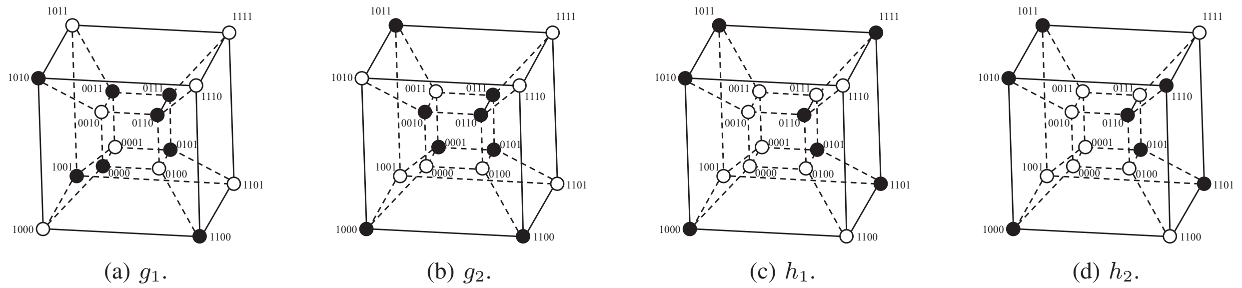
IV. Classifier

In this section, we first show the effect of influence and sensitivity signature vectors on NPN classification and then present our classification algorithm.

A. Signature Vectors Selection

All aries of cofactor signature vectors have been proved to be a canonical form [3]. However, computing all-ary cofactor signatures are time-consuming. Next, we will show that OIV, OSV, and $OSDV$ are strong discriminators of NPN non-equivalence over OCV_1 and OCV_2.

Fig. 4 shows four hypercubes of two pairs of nonequivalent 4-input Boolean functions g_1, g_2 and h_1, h_2. The $OCV_1(g_1)$=$OCV_1(g_2)$=$(3, 4, 4, 4, 4, 4, 4, 5)$ and $OCV_2(g_1)$=$OCV_2(g_2)$=$(1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3)$ are identical, respectively. However, $OIV_1(g_1)$=$(6, 6, 6, 8)$ and $OIV_2(g_2)$=$(2, 6, 6, 8)$ of these two functions are different. The ordered influence vector can distinguish nonequivalent Boolean functions that cannot be classified by OCV_1 and OCV_2. Moreover, the $OCV_1(h_1)$=$OCV_1(h_2)$=$(2, 3, 3, 3, 4, 4, 4, 5)$, $OCV_2(h_1)$=$OCV_2(h_2)$=$(0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3)$ and $OIV_1(h_1)$=$OIV_1(h_2)$=$(3, 5, 5, 5)$ are identical, respectively. But $OSV^1(h_1)$=$(2, 2, 2, 2, 3, 3, 4)$ and $OSV^1(h_2)$=$(1, 2, 3, 3, 3, 3, 3)$ of these two functions



(a) g_1. (b) g_2. (c) h_1. (d) h_2.

Fig. 4: Hypercubes of two pairs of nonequivalent 4-input Boolean functions g_1, g_2 and h_1, h_2.

Algorithm 1 NPN Classifer

Input: input variables n, truth table set tts
Output: NPN equivalent classes
1: **for** tt in tts **do**
2: get $OCV_1(tt)$ and $OCV_2(tt)$;
3: get $OIV(tt)$;
4: get $OSV(tt)$;
5: compute $OSDV(tt)$ using $OSV(tt)$;
6: construct $MSV(tt)$ using $OCV_1(tt)$, $OCV_2(tt)$, $OIV_1(tt)$, $OSV(tt)$ and $OSDV(tt)$;
7: $class \leftarrow hash(MSV(tt))$;
8: **end for**

are different. Therefore, the ordered sensitivity vector can distinguish nonequivalent Boolean functions that can not be classified by OCV_1 and OCV_2. More evaluations will be seen in Section V-B.

B. Classification Method

As the property of cofactor, influence, and sensitivity mentioned above, it is very convenient to implement the computation of the signature based on the binary string as the representation. We adopt several bitwise operation techniques in [17] for fast signatures computation. For example, to calculate the cofactor signature of a certain literal, we only need to keep the relevant bits in the truth table and count the number of 1s.

Algorithm 1 gives the overall algorithm for NPN classification. First, we compute several signature vectors as the definitions above (line 2 to line 5). Then, we construct the MSV (Mixed Signature Vector) (line 6). At last, a hash function is used to finish the classification (line 7) of this Boolean function. For runtime saving, we can use OSV^1, OSV^0 replacing OSV and $OSDV^1$, $OSDV^0$ replacing $OSDV$. When considering balanced Boolean functions, it should be noted that different OSV need to be constructed according to the Theorem 4. Most NPN classification methods define a canonical form utilizing cofactor signatures and hierarchical symmetry properties first and then propose a complex algorithm to compute it. Boolean functions with different symmetric properties further increase the complexity. Our classification method only needs bitwise operations and hash to finish the classification.

V. Experimental Evaluations

A. Setup

We implement our classifier algorithm in C++ and compare our work with some state-of-the-art NPN classification works. All procedure runs on an Intel Xeon 2-CPU 20-core computer with 60GB RAM. The input of a benchmark consists of a list of Boolean functions (in the truth-table form) to be classified under NPN equivalence. And the output consists of the number of equivalence classes, as well as the classified truth tables.

We use EPFL benchmarks [18] to test the effectiveness of our algorithm on real synthesis applications. The truth tables are extracted from these benchmarks using cut enumeration. We deleted the Boolean functions of the same truth table.

B. Evaluation of the Signature Vectors

We evaluate the effectiveness of each signature vector part and different combinations. Table II shows the results. The number of exact classes in the third column is run using Kitty when $n \leq 6$ and the exact version in [19] when $n > 6$. In general, cofactor signatures are more effective in classification than influence but worse than sensitivity. The combination of influence and sensitivity signature vectors is better than cofactor signature vectors. It performs an exact classification when $n \leq 7$. Therefore, point characteristics and face characteristics have different properties, and their combination can effectively complete NPN classification.

C. Evaluation of NPN Classifier

We evaluate our classification algorithm and compare the results with Kitty in EPFL logic synthesis libraries [21] and several state-of-the-art works [13], [14], [20], which are implemented in ABC [19] as command `testnpn` with different arguments. Due to some methods in [14] using an exhaustive enumeration for exact classification at the end, we modified ABC and removed this part for a fair comparison. Table III shows the results. Kitty can gain exact NPN classification, but it runs slowly and will not work when $n > 6$. The command `testnpn -7` fails when $n \leq 5$. So we omit these parts of the results. The number of exact classes runs using Kitty when $n \leq 6$ and the exact version in [19] when $n > 6$. Our classification gains up to 325x speedup over Kitty (6-bit) with the same classification accuracy. Although the algorithm in [13] is ultra-fast, it fails inaccurate classification. Algorithms in [20] and [14] show speed and classification accuracy improvements.

Previous classification methods need a canonical form and computation algorithm to get NPN classes. The runtime of such methods is related to the properties of Boolean functions, such as symmetries. If the Boolean functions have bad properties, then the computation of the canonical form will become complicated, leading to unstable runtime. The classifier proposed in this paper needs only bitwise computation and hash operations, so the runtime is only related to the bitwidth and the total number of Boolean

TABLE II: The results of classification using different signature vectors.

n	#Exact Classes	#Classes by OIV	#Classes by OCV_1	#Classes by OSV	#Classes by OIV+OSV	#Classes by OCV_1+OSV	#Classes by OCV_1+OCV_2 +OSV	#Classes by OIV+OSV +$OSDV$	#Classes by All
4	49	28	41	48	48	49	49	49	**49**
5	312	173	251	305	310	311	311	312	**312**
6	1673	1175	1380	1619	1654	1668	1671	1673	**1673**
7	6071	5224	5498	5985	6052	6057	6057	6052	**6071**
8	48895	44497	44183	48584	48876	48876	48876	48877	48887
9	92741	87485	87080	92381	92721	92723	92723	92721	92725
10	184832	178155	177799	184428	184794	184794	184795	184796	184796

TABLE III: Runtime and accuracy comparison of different NPN classifiers.

n	#Func	#Exact Class	Kitty		testnpn -6 [13]		testnpn -7 [20]		testnpn -11 [14]		**Ours**	
			#Class	Time	#Class	Time	#Class	Time	#Class	Time	#Class	Time
4	1146	49	49	0.031	251	0.0003	-	-	52	0.0024	**49**	**0.0013**
5	6824	312	312	0.858	1586	0.0026	-	-	322	0.015	**312**	**0.0049**
6	28672	1673	1673	39.453	7375	0.006	1752	0.021	1690	0.046	**1673**	0.121
7	80123	6071	-	-	23318	0.216	6249	0.067	6115	0.194	**6071**	0.773
8	480516	48895	-	-	190708	0.13	50066	0.554	49577	4.701	48887	12.350
9	691474	92741	-	-	278090	0.296	94283	1.438	93575	128.926	92725	**51.86**
10	1153464	184832	-	-	500911	0.881	187117	4.193	186098	1329.995	184796	**318.56**

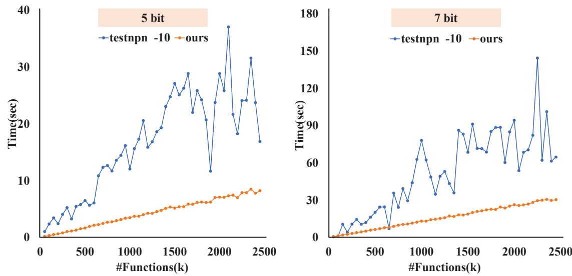


Fig. 5: Our classifier has stable runtime.

functions. Fig. 5 shows the runtime of our classifier and `testnpn -11` testing on randomly generated 5-bit and 7-bit Boolean functions. The horizontal axis is the number of generated Boolean functions. We randomly generate a fixed number of Boolean functions with truth tables in consecutive binary encoding for each bit. This figure shows that the runtime of our classifier is almost linear with the total number of Boolean functions, while `testnpn -11` fluctuates widely with different sets of Boolean functions. Actually, our classifier cannot return exact matching solutions. Influence and sensitivity still have great potential to be extended to the traditional method to achieve exact NPN classification, and we will explore them in the future.

VI. Conclusion

This paper rethought the NPN classification problem in terms of face and point characteristics of Boolean functions. We introduced Boolean sensitivity and influence, two concepts considering point and face-point characteristics in NPN classification. We designed some signature vectors based on these two characteristics. Combined with cofactor signatures, we develop a new classifier that only relies on signature vector computation. The experiments showed that the proposed NPN classifier gains better NPN classification accuracy with comparable and stable speed.

Acknowledgment

This work is partly supported by the National Natural Science Foundation of China (Grant No. 62090021) and the National Key R&D Program of China (Grant No. 2022YFB4500500). Shenggen Zheng acknowledges support in part from the Major Key Project of PCL.

References

[1] M. Soeken *et al.*, "Heuristic NPN classification for large functions using AIGs and LEXSAT," in *SAT*. Springer, 2016, pp. 212–227.
[2] D. Chai *et al.*, "Building a better Boolean matcher and symmetry detector," in *DATE*, 2006, pp. 1–6.
[3] A. Abdollahi *et al.*, "Symmetry detection and Boolean matching utilizing a signature-based canonical form of Boolean functions," *IEEE TCAD*, vol. 27, no. 6, pp. 1128–1137, 2008.
[4] G. Agosta *et al.*, "A transform-parametric approach to Boolean matching," *IEEE TCAD*, vol. 28, no. 6, pp. 805–817, 2009.
[5] X. Zhou *et al.*, "Fast adjustable NPN classification using generalized symmetries," *ACM TRETS*, vol. 12, no. 2, pp. 1–16, 2019.
[6] J. Zhang *et al.*, "Enhanced fast Boolean matching based on sensitivity signatures pruning," in *ICCAD*, 2021, pp. 1–9.
[7] E. M. Clarke *et al.*, "Spectral transforms for large Boolean functions with applications to technology mapping," in *DAC*, 1993, pp. 54–60.
[8] M. A. Thornton *et al.*, "Logic circuit equivalence checking using Haar spectral coefficients and partial BDDs," *VLSI Design*, vol. 14, no. 1, pp. 53–64, 2002.
[9] C.-C. Tsai *et al.*, "Boolean functions classification via fixed polarity reed-muller forms," *IEEE TC*, vol. 46, no. 2, pp. 173–186, 1997.
[10] A. Abdollahi *et al.*, "A new canonical form for fast Boolean matching in logic synthesis and verification," in *DAC*, 2005, pp. 379–384.
[11] J. Zhang *et al.*, "An efficient NPN Boolean matching algorithm based on structural signature and Shannon expansion," *Cluster*, vol. 22, no. 3, pp. 7491–7506, 2019.
[12] V. N. Kravets *et al.*, "Generalized symmetries in Boolean functions," in *ICCAD*, 2000, pp. 526–532.
[13] Z. Huang *et al.*, "Fast Boolean matching based on NPN classification," in *FPT*, 2013, pp. 310–313.
[14] X. Zhou *et al.*, "Fast exact NPN classification by co-designing canonical form and its computation algorithm," *IEEE TC*, vol. 69, no. 9, pp. 1293–1307, 2020.
[15] S. Cook *et al.*, "Bounds on the time for parallel RAM's to compute simple functions," in *STOC*, 1982, pp. 231–233.
[16] J. Kahn *et al.*, "The influence of variables on Boolean functions," in *SFCS*, 1988, pp. 68–80.
[17] H. S. Warren, *Hacker's delight*. Pearson Education, 2013.
[18] L. Amarú *et al.*, "The EPFL combinational benchmark suite," in *IWLS*, 2015.
[19] Berkeley Logic Synthesis and Verification Group. "ABC: A system for sequential synthesis and verification". 2022. [Online]. Available: https://people.eecs.berkeley.edu/~alanmi/abc/
[20] A. Petkovska *et al.*, "Fast hierarchical NPN classification," in *FPL*, 2016, pp. 1–4.
[21] M. Soeken *et al.*, "The EPFL logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.