IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 October 2022, accepted 2 November 2022, date of publication 14 November 2022,
date of current version 18 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3221764

METHODS

A Heuristic Boolean NPN Equivalent Matching Verification Method Based on Shannon Decomposition

JULING ZHANG[1], WENQIANG GUO[1], GUOWU YANG[2], (Member, IEEE), YIXIN ZHU[1], AND XIAOYI LV[3]

[1]School of Cyberspace Security, Xinjiang University of Finance and Economics, Urumqi 830012, China
[2]Big Data Research Center, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[3]School of Software, Xinjiang University, Urumqi 830000, China

Corresponding author: Juling Zhang (zjlgj@163.com)

This work was supported in part by the Natural Science Foundation Project of the Xinjiang Autonomous Region under Grant 2019D01A27; in part by the School-Level Scientific Research Fund Project of Xinjiang University of Finance and Economics under Grant 2019XYB005; and in part by Sichuan Regional Innovation Cooperation Project 2020YFQ0018.

ABSTRACT In this paper, we describe a new verification method to accelerate the input negation and/or input permutation and/or output negation (NPN) Boolean matching for a single-output completely specified Boolean function. Through research on the Boolean Shannon decomposition binary tree, we prove that the signature vectors of the left child node and right child node are complementary relative to the signature vector of the parent node. We introduce an independent variable check to speed up the detection of candidate transformation. The proposed approach utilises this complementarity, a symmetry check, an independent variable check and a phase collision check, which can quickly verify whether the candidate transformation obtained in the detection of the candidate transformation of the Boolean matching process is accurate. We perform experiments on two types of Boolean function sets. One type consists of Boolean functions from randomly generated circuits. The other is exported from the Microelectronics Center of North Carolina (MCNC) benchmark. The experimental results show that the average runtime of our algorithm is 68.8% faster than those in Zhang et al. (2019) on two randomly generated circuits and 51% faster than those in Zhang et al. (2019) when tested on the MCNC benchmark circuit set. Therefore, the experimental results demonstrate the effectiveness of the proposed method.

INDEX TERMS Boolean matching, NPN equivalence, Shannon expansion, Shannon decomposition binary tree, signature vector.

I. INTRODUCTION

Since the 1930s, scholars have realised that Boolean matching and Boolean classification play very important roles in the analysis of switching circuits and started to study how to construct a ''good'' Boolean network. At present, Boolean matching and Boolean classification are usually applied to industrial applications such as cell-library binding, technology mapping, engineering, synthesis, and circuit optimisation [2], [3], [4].

The associate editor coordinating the review of this manuscript and approving it for publication was Abdallah Kassem.

Boolean equivalence classification divides Boolean functions into several equivalence classes. Any two Boolean functions in an equivalent class can realise each other. In circuit optimisation, the circuit with the lowest cost can be selected to replace other Boolean functions [5], [6], [7], [8]. Reference [5] used the group algebra method to complete the input negation and/or input permutation (NP)-equivalent classification and input negation and/or input permutation and/or output negation (NPN)-equivalent classification of Boolean functions with 10 variables. Based on the canonical form and by exploiting symmetries under different phase assignments and higher-order symmetries of Boolean

functions, Reference [6] proposed a Boolean NPN-equivalent classification algorithm. A hierarchical method introduced in Reference [7] enables a rapid exact NPN classification for functions up to 10 inputs, and its speed is 3.7 times faster than a state-of-the-art nonhierarchical method. Reference [8] studied a new method for the affine equivalence classification for Boolean functions with no more than R(7,2).

We study the Boolean NPN-equivalent matching problem. When there is an NPN transformation τ that can transform Boolean function f into Boolean function $g/\overline{g}$, we say that the functions are NPN equivalent. For an n-variable Boolean function, there are $n!2^{n+1}$ NPN transformations [9]. With the increase in the number of variables, the number of NPN transformations exponentially increases. Therefore, Boolean NPN equivalence classification and matching are nondeterministic polynomial time (NP)-hard problems.

Currently, NPN-equivalent matching methods based on the canonical form, pairwise comparison and Boolean satisfiability (SAT) are commonly used. Different methods adopt various strategies to resolve the NPN equivalence matching problem. In the canonical-based Boolean matching algorithm, two equivalent Boolean functions have identical canonical forms. Therefore, the key issue of the canonical-based Boolean matching algorithm is finding the canonical form [9], [10], [11], [12], [13], [14], [15], [16]. Pairwise comparison algorithms utilise signature differentiated variables and search the variable correspondence relation between two Boolean functions [1], [17], [18], [19], [20]. Therefore, the most important consideration is the choice and use of signatures. The SAT-based Boolean matching method is often applied to equivalent matching for technology mapping of field-programmable gate arrays (FPGAs). For a programmable circuit g and a Boolean function f, f can be realised by circuit g if and only if there is a set of configurations in the programmable bits of the circuit g that satisfy $f \equiv g$ under all input and output conditions [21], [22], [23], [24], [25]. The difficulty of the SAT-based method is reducing the search space.

Based on the study of Shannon expansion, we construct a Shannon decomposition binary tree for the Boolean NPN equivalent matching process. Each node of the Shannon decomposition binary tree is a Shannon cofactor. We research the relationship between the cofactor signature of parents' nodes and that of left and right children's nodes, prove that the signature vectors of the left child node and right child node are complementary relative to the signature vector of the parent node, and find the relationship between the cofactor signatures of the left and right child nodes. We also introduce independent variable detection, which can speed up candidate transformation detection. According to these two relationships, variable symmetry and independent variables, this heuristic verification method is proposed. The pairwise comparison matching algorithm first finds the variable correspondence relationship between two Boolean functions, i.e., obtains a candidate transformation τ. Then, it computes $f(\tau X)$ and determines $f(\tau X) = g(X)/\overline{g}(X)$ to verify whether τ is correct or not. Our verification algorithm is faster than the traditional approach, which speeds up Boolean matching.

The remainder of this paper is organized as follows. In Section II, the related research on Boolean matching is reviewed. In Section III, we introduce the terminologies and key definitions and retrospectively review the techniques of the algorithm in Reference [1]. Through proofs and examples, Section IV presents our new verification method based on the Boolean Shannon decomposition binary tree. Section V elaborates on the Boolean NPN equivalent matching verification method based on Shannon decomposition. Section VI demonstrates the effectiveness of our algorithm by presenting experimental results. We summarise our work and outline future work in Section VII.

II. RELATED WORKS

The ultimate goal of almost all equivalent matching is to find the transformation between two equivalent Boolean functions. Canonical-based NPN Boolean matching uses the property that two NPN equivalent Boolean functions have identical canonical forms. It is most commonly used to solve the problem of Boolean matching in cell-library binding. Reference [9] proposed a semicanonical form for the NPN Boolean matching algorithm and applied it to Boolean functions of 6-16 inputs, but its runtime degrades when the number of input variables is 18-22. The author of Reference [10] proposed a compact signature vector that is composed of general signatures of 0^{th}, $1^{st}\ldots$, n^{th}. After researching the Shannon cofactor operation, Reference [11] introduced a Boolean difference signature into the general signature vector of Reference [10], which formed a difference and cofactor (DC) signature vector. The use of a DC signature vector reduced the search space and substantially accelerated the calculation of the canonical form. Reference [12] presented an NP Boolean matching method that used a table look-up and a tree-based breadth-first search strategy to compute the NP representative for a given Boolean function. In Reference [13], the authors proposed a new canonical form that used the spectral method. They identified a linear transformation that could speed up Boolean matching. The authors of Reference [14] and Reference [15] studied constructing minimal representations of multiple-output Boolean functions and their complexity. A co-designing canonical form is presented in Reference [16], the NPN classification algorithm of which can exactly classify any practical Boolean function up to 16 inputs in reasonable time. References [10], [13] and [16] are only experiments on Boolean functions with no more than 20 inputs. The analysis and utilisation of the Boolean difference signature are not sufficient in Reference [11].

Reference [1] researched pairwise NPN Boolean matching and presented a structural signature vector that could efficiently eliminate incorrect variable mappings when a candidate NP transformation is detected. Symmetry detection, variable grouping and phase collision detection were proposed, which greatly reduced the search space and

accelerated the matching speed. The authors of Reference [17] researched NPN Boolean matching in the presence of incompletely specified functions. They addressed a pairwise matching algorithm based on the 1^{st} and 2^{nd} general signatures. A fast pairwise Boolean matching algorithm was proposed in Reference [18], which made full use of the cofactor signature, Boolean difference signature and symmetric variables. References [19] and [20] both used signatures to reduce the complexity of pairwise comparisons. There are no more than 16 variables of Boolean matching in References [17], [19] and [20], and the comparisons with previous algorithms are not comprehensive.

Reference [21] proposed an SAT-based Boolean matching algorithm that integrates simulation and SAT techniques. Reference [22] utilised symmetry to overcome the bottleneck of the SAT-based Boolean matching method. Their approaches greatly reduced the scale of the problem and improved the performance of SAT-based Boolean matching for FPGAs. Reference [23] improved the SAT-based Boolean matching method for look-up table (LUT)-based circuits by using one-hot encoding and counterexample-guided abstraction refinement (CEGAR).

Other methods have been utilised to resolve Boolean matching. Reference [24] introduced the Bloom filter to the SAT-based Boolean matching method to solve insufficient storage space. The authors of Reference [25] and Reference [26] used incremental and dynamic learning approaches to solve the Boolean matching problem, respectively. Through conflict-driven dynamic learning and abstraction, Reference [27] effectively pruned infeasible matching solutions. Reference [28] created a treelike logic network that represented the decomposition properties of Boolean functions and implemented a tree-based Boolean matching algorithm. Reference [29] researched a dynamic data structure that efficiently expresses Boolean functions, and Reference [30] proposed a negation and permutation of outputs and negation and permutation of inputs (NPNP)-equivalent matching algorithm for large-scale Boolean functions.

III. PRELIMINARIES

A. BOOLEAN FUNCTION AND NPN EQUIVALENT

Let X be a Boolean vector $(x_0, x_1, \cdots, x_{n-1})$; the Boolean function $f(X)$ can be expressed by $f(X) : B^n \rightarrow B$. Here, $f(X)$ is a single-output and completely specified Boolean function.

The commonly used Boolean function representation methods are the truth table and minterm representations. The truth table method makes each combination of Boolean function inputs and its output into a table. Each set of inputs in the truth table that makes output 1 is called a minterm. In the minterm representation, a Boolean function is the result of the ''or'' operation on all minterms. A simplified truth table, i.e., a truth table that contains only minterms, is used to represent Boolean functions in many research work [12]. Its purpose is to facilitate the calculation of the signature. For a Boolean function $f = \bar{x}_0\bar{x}_2 + x_0x_2 + \bar{x}_0x_1x_2 + x_0\bar{x}_1\bar{x}_2$, the left side of Fig. 1 is its truth table, and the right side is its simplified truth table. Binary decision diagrams (BDDs) are compressed representations of Boolean functions. Since BDD is the most efficient representation format for Boolean functions and is widely adopted both in industry and in research works [31], our algorithm uses BDD to represent Boolean functions.

Definition 1 (P-Transformation): Perform a permutation transformation π on vector X: $X_\pi(x_0, x_1, \cdots, x_n) = (x_{\pi(0)}, x_{\pi(1)}, \cdots, x_{\pi(n-1)})$,$\pi(i) = j$, where $i, j \in \{0, 1, \cdots, n-1\}$.

Definition 2 (N-Transformation): Perform a negation transformation φ on a vector X. We have the following: $X^\varphi(x_0, x_1, \cdots, x_n) = (x_0^{\varphi(0)}, x_1^{\varphi(1)}, \cdots, x_{n-1}^{\varphi(n-1)})$. Here, $x_i^{\varphi(i)} = x_i$ when $\varphi_{(i)} = 1$, and $x_i^{\varphi(i)} = \bar{x}_i$ when $\varphi_{(i)} = 0$.

Definition 3 (NP-Transformation): When a negation transformation φ followed by a permutation transformation π acts on X, we have an NP transformation $X_\pi^\varphi = (x_{\pi(0)}^{\varphi(0)}, x_{\pi(1)}^{\varphi(1)}, \cdots, x_{\pi(n-1)}^{\varphi(n-1)})$. In this paper, the *NP*-transformation is denoted by T.

Definition 4 (NPN Equivalent): Boolean function $f(X)$ is NPN equivalent to Boolean function $g(X)$ if and only if there exists an *NP*-transformation T that makes the equation $f(TX) = g(X)$ or $f(TX) = \overline{g(x)}$ true.

NPN equivalence of Boolean functions $f(X)$ and $g(X)$ is denoted by $f(X) \equiv g(X)$.

Variable mapping involves the correspondence relation between two variables of two Boolean functions [1]. An *NP* transformation can be denoted by a variable mapping set as follows: $T = \{x_{i_0}^{\alpha_{i_0}} \rightarrow x_{j_0}^{p_{j_0}}, x_{i_1}^{\alpha_{i_1}} \rightarrow x_{j_1}^{p_{j_1}}, \cdots, x_{i_{(n-1)}}^{\alpha_{i_{(n-1)}}} \rightarrow x_{j_{(n-1)}}^{p_{j_{(n-1)}}}\}$.

Example 1: Let $f(X) = x_0x_1x_2 + x_0\bar{x}_1\bar{x}_2$, $g(X) = \bar{x}_0\bar{x}_1x_2 + x_0\bar{x}_1\bar{x}_2$. There is an NP transformation $T = \{x_0 \rightarrow \bar{x}_1, x_1 \rightarrow x_2, x_2 \rightarrow \bar{x}_0\}$, which satisfies $f(TX) = g(X)$. Therefore, the Boolean function $f(X)$ is NPN equivalent to $g(X)$.

Lemma 1: Shannon expansion constitutes the identity $f = x_if_{x_i} + \bar{x}_if_{\bar{x}_i}$ [1].

Boolean functions f_{x_i} and $f_{\bar{x}_i}$ of the Shannon expansion are called cofactors, and $f_{x_i} = f[x_i \leftarrow 1], f_{\bar{x}_i} = f[x_i \leftarrow 0]$. Therefore, f_{x_i} and $f_{\bar{x}_i}$ are also marked as f_1 and f_0, respectively [32]. The Shannon expansion is also called Shannon decomposition, and variable x_i is the decomposition variable. Iterative Shannon decomposition continues the Shannon decomposition of cofactors. Iterative Shannon decomposition of an n-variable Booelan function will produce multiple cofactors with variables less than n. For example, the n-variable Boolean function f is decomposed by x_i and then by x_j to obtain $f = x_ix_jf_{x_ix_j} + x_i\bar{x}_jf_{x_i\bar{x}_j} + \bar{x}_ix_jf_{\bar{x}_ix_j} + \bar{x}_i\bar{x}_jf_{\bar{x}_i\bar{x}_j} = x_ix_jf_{11} + x_i\bar{x}_jf_{10} + \bar{x}_ix_jf_{01} + \bar{x}_i\bar{x}_jf_{00}$. Here, $f_{11} = f[x_i \leftarrow 1, x_j \leftarrow 1], f_{10} = f[x_i \leftarrow 1, x_j \leftarrow 0], f_{01} = f[x_i \leftarrow 0, x_j \leftarrow 1], f_{00} = f[x_i \leftarrow 0, x_j \leftarrow 0]$. This decomposition generates four cofactors, i.e., four Boolean functions with $n-2$ variables. An n-variable Boolean function is decomposed by k variables, which forms 2^k Boolean functions with $n-k$ variables.

x_0	x_1	x_2	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

x_0	x_1	x_2	f
0	0	0	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	1	1

FIGURE 1. Truth table and simplified truth table of Boolean function f.

Normally, a signature is a quantitative property that is independent of the NP transformation of a Boolean function. The cofactor signature is the satisfy count of the cofactors with respect to each variable, which is widely utilised to address the problem of Boolean matching; it is also called the general signature [10]. The zeroth-order signature of Boolean function $f(X)$ is its satisfy count, which is denoted by $|f|$. A cube b is the conjunction of $k-1$ variables, i.e., $|b| = k-1$. f_b is an *(n-k-1)*-variable cofactor of the Shannon expansion for Boolean function f. $|f_b|$ is a cofactor signature that is the number of minterms of f_b.

*Definition 5 (*k*-th Signature Value):* For a variable x_i of cofactor f_b generated by the Shannon decomposition of n-variable function f and $|b| = k-1$, $(|f_{bx_i}|, |f_{b\bar{x}_i}|)$ is a k-th signature value of x_i with respect to f. $|f_{bx_i}|$ is the satisfy count of f_{bx_i}. $|f_{b\bar{x}_i}|$ is the satisfy count of $f_{b\bar{x}_i}$.

For example, $(|f_{x_i}|, |f_{\bar{x}_i}|)$ is the 1^{st} signature value of x_i with respect to Boolean function f. $(|f_{x_jx_i}|, |f_{x_j\bar{x}_i}|)$ is the 2^{nd} signature value of x_i with respect to Boolean function f when $b = x_j$. In this paper, the 1^{st} signature of x_i with respect to Boolean function f is denoted by (a_{0i}, b_{0i}), and the k-th signature value of x_i with respect to Boolean function f is denoted by $(a_{(k-1)i}, b_{(k-1)i})$. If variable x_i has been decomposed, its k-th signature value is denoted by $(-, -)$.

In Fig. 1, the 1^{st} signature value of x_0 is $(|f_{x_0}|, |f_{\bar{x}_0}|)$, i.e., $(3, 3)$. $(|f_{x_0x_1}|, |f_{x_0\bar{x}_1}|) = (1, 2)$ is the 2^{nd} signature value of x_1 with respect to f when $b = x_0$.

*Definition 6 (*k*-th Signature Vector):* For a cube $|b| = k-1$, $\{(|f_{bx_0}|, |f_{b\bar{x}_0}|), (|f_{bx_1}|, |f_{b\bar{x}_1}|), \cdots, (|f_{bx_{n-1}}|, |f_{b\bar{x}_{n-1}}|)\}$ is a k-th signature vector with respect to Boolean function f.

For the Boolean function f in Fig. 1, the 2-nd signature vector of f is $\{(-, -), (1, 2), (2, 1)\}$ when $b = x_0$.

Definition 7 (Independent Variable): Variable x_i of Boolean function f is an independent variable when $|f'_{x_i}| = 0$, where $f'_{x_i} = f_{x_i} \oplus f_{\bar{x}_i}$ [18].

B. OVERVIEW OF OUR PREVIOUS ALGORITHM

We proposed an NPN Boolean matching algorithm based on the structural signature and Shannon expansion in Reference [1]. The structural signature vector of Reference [1] includes the cofactor signature, symmetry information and grouping information. The algorithm of Reference [1] obtained candidate transforms T between n-variable Boolean functions f and g by iterative Shannon decomposition. Each candidate transformation T has n variable mappings. To find the correct transformation as early as possible and reduce the search space, Reference [1] employed the following strategies.

1) ITERATIVE SHANNON DECOMPOSITION AND SIGNATURE VECTOR UPDATING

Given two NP equivalent Boolean functions f and g and an NP transformation T between them, the cofactors generated by the Shannon decomposition of these two Boolean functions must be NP equivalent according to T. Therefore, Reference [1] used the iterative Shannon decomposition update signature vector and obtained variable mappings between f and g.

2) PHASE CONFLICT CHECK

The phase relationship between two variables of a correct variable mapping will not change in the iterative Shannon decomposition. Using phase conflict can discover incorrect variable mappings.

3) VARIABLE GROUPING

For two variables of a correct variable mapping, the changes in their signature value are synchronous in the iterative Shannon decomposition process. Variable grouping can identify the wrong variable mapping.

4) SYMMETRY CHECK

The symmetry variable and nonsymmetry variable may have identical signature values. However, the variable mapping between the symmetry variable and the nonsymmetry variable must be faulty. Symmetry detection can avoid this error.

IV. BOOLEAN SHANNON DECOMPOSITION BINARY TREE

In this section, we research the Boolean Shannon decomposition lemma and propose a new verification method. The goal of our method is to accelerate the verification speed of Boolean matching.

A. BOOLEAN SHANNON DECOMPOSITION

For an n-variable Boolean function f, the Shannon decomposition with respect to x_i is $f = x_i f_{x_i} + \overline{x}_i f_{\overline{x}_i} = x_i f_1 + \overline{x}_i f_0$. Here, the variable x_i is the decomposition variable [1]. The Shannon decomposition binary tree is shown in Fig. 2.

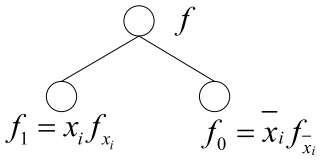

FIGURE 2. Shannon decomposition binary tree of Boolean function f.

Suppose that the 1^{st} signature vector of f is expressed as follows: $V_f = \{(a_{00}, b_{00}), (a_{01}, b_{01}), \cdots, (a_{0i}, b_{0i}), \cdots, (a_{0(n-1)}, b_{0(n-1)})\}$. After Shannon decomposition by variable x_i, let the signature value of the decomposition variable x_i be denoted by $(-, -)$. The 1^{st} signature vector of decomposed Boolean functions f_1 and f_0 can be expressed as follows:

$$\begin{aligned} V_{f_1} &= \{(a_{10}, b_{10}), (a_{11}, b_{11}), \cdots, (a_{1(i-1)}, b_{1(i-1)}), (-, -), \\ &\quad (a_{1(i+1)}, b_{1(i+1)}), \cdots, (a_{1(n-1)}, b_{1(n-1)})\} \\ V_{f_0} &= \{(a'_{10}, b'_{10}), (a'_{11}, b'_{11}), \cdots, (a'_{1(i-1)}, b'_{1(i-1)}), (-, -), \\ &\quad (a'_{1(i+1)}, b'_{1(i+1)}), \cdots, (a'_{1(n-1)}, b'_{1(n-1)})\} \end{aligned}$$

For f_1 and f_0, the above signature vectors are their 1^{st} signature vectors. However, these signature vectors are part of the 2-nd signature vector for f.

When the algorithm in Reference [1] is used to decompose Boolean functions, the number of decompositions is less than or equal to $n - 1$. This decomposition process produces a decomposition binary tree, which we call the Shannon decomposition binary tree. Here, we call the subtree on the left the transformation search subtree and the subtree on the right the transformation verification subtree, as shown in Fig. 3.

The signature values of the variables of Boolean function f' on each node of the Boolean Shannon decomposition binary tree satisfy Property 1 below.

Property 1: Suppose that the signature vector of Boolean function f' is $\{(a_{\alpha 0}, b_{\alpha 0}), (a_{\alpha 1}, b_{\alpha 1}), \cdots, (a_{\alpha(n-1)}, b_{\alpha(n-1)})\}$, the signature values of the variables other than those that have been decomposed from f' satisfy equation (1) below.

$$\begin{aligned} a_{\alpha 0} + b_{\alpha 0} &= a_{\alpha 1} + b_{\alpha 1} = \cdots = a_{\alpha i} + b_{\alpha i} \\ &= \cdots = a_{\alpha(n-1)} + b_{\alpha(n-1)} = |f'| \end{aligned} \quad (1)$$

Proof: This follows immediately from the definition of the signature value. ■

Suppose that the Boolean functions of a parent node, its left and right child nodes are f_p, f_l and f_r in Fig. 3. The signature vectors of these three Boolean functions are as follows:

$$\begin{aligned} f_p &= \{(a_{k0}, b_{k0}), (a_{k1}, b_{k1}), \cdots, (a_{ki}, b_{ki}), \cdots, \\ &\quad (a_{k(n-1)}, b_{k(n-1)})\} \\ f_l &= \{(a_{(k+1)0}, b_{(k+1)0}), (a_{(k+1)1}, b_{(k+1)1}), \cdots, \\ &\quad (a_{(k+1)i}, b_{(k+1)i}), \cdots, (a_{(k+1)(n-1)}, b_{(k+1)(n-1)})\} \\ f_r &= \{(a'_{(k+1)0}, b'_{(k+1)0}), (a'_{(k+1)1}, b'_{(k+1)1}), \cdots, \\ &\quad (a'_{(k+1)i}, b'_{(k+1)i}), \cdots, (a'_{(k+1)(n-1)}, b'_{(k+1)(n-1)})\} \end{aligned}$$

All signature values of the decomposed variables of Boolean functions f_p, f_l and f_r are $(-, -)$.

Property 2: In the Shannon decomposition process, Boolean function f_p is decomposed by x_i to yield Boolean functions f_l and f_r. The signature values of the variables except those that have been decomposed from these three Boolean functions satisfy equations (2)-(5) as follows:

$$\begin{aligned} &a_{(k+1)0} + b_{(k+1)0} \\ &\quad = a_{(k+1)1} + b_{(k+1)1} = \cdots = a_{(k+1)(i-1)} \\ &\quad\quad + b_{(k+1)(i-1)} = a_{(k+1)(i+1)} + b_{(k+1)(i+1)} = \cdots \\ &\quad = a_{(k+1)(n-1)} + b_{(k+1)(n-1)} = a_{ki} \end{aligned} \quad (2)$$

$$\begin{aligned} &a'_{(k+1)0} + b'_{(k+1)0} \\ &\quad = a'_{(k+1)1} + b'_{(k+1)1} = \cdots = a'_{(k+1)(i-1)} \\ &\quad\quad + b'_{(k+1)(i-1)} = a'_{(k+1)(i+1)} + b'_{(k+1)(i+1)}) = \cdots \\ &\quad = a'_{(k+1)(n-1)} + b'_{(k+1)(n-1)} = b_{ki} \end{aligned} \quad (3)$$

$$\begin{aligned} &a_{(k+1)0} + a'_{(k+1)0} \\ &\quad = a_{k0}, a_{(k+1)1} + a'_{(k+1)1} = a_{k1}, \cdots, \\ &\quad\quad a_{(k+1)(i-1)} + a'_{(k+1)(i-1)} \\ &\quad = a_{k(i-1)}, a_{(k+1)(i+1)} + a'_{(k+1)(i+1)} \\ &\quad = a_{k(i+1)}, \cdots, a_{(k+1)(n-1)} + a'_{(k+1)(n-1)} = a_{k(n-1)} \end{aligned} \quad (4)$$

$$\begin{aligned} &b_{(k+1)0} + b'_{(k+1)0} \\ &\quad = b_{k0}, b_{(k+1)1} + b'_{(k+1)1} = b_{k1}, \cdots, \\ &\quad\quad b_{(k+1)(i-1)} + b'_{(k+1)(i-1)} \\ &\quad = b_{k(i-1)}, b_{(k+1)(i+1)} + b'_{(k+1)(i+1)} \\ &\quad = b_{k(i+1)}, \cdots, b_{(k+1)(n-1)} + b'_{(k+1)(n-1)} = b_{k(n-1)} \end{aligned} \quad (5)$$

Proof: This follows immediately from the definition of the signature value and Shannon expansion lemma. ■

Corollary 1: In the Shannon decomposition binary tree, signature vector V_{f_l} of the left child node and signature vector V_{f_r} of the right child node are complementary relative to the signature vector V_{f_p} of the parent node.

Proof: According to Equations (4) and (5), the signature vector of f_r can be expressed as follows:

$$\begin{aligned} V_{f_r} &= \{(a_{k0} - a_{(k+1)0}, b_{k0} - b_{(k+1)0}), (a_{k1} - a_{(k+1)1}, b_{k1} \\ &\quad - b_{(k+1)1}), \cdots, (a_{k(i-1)} - a_{(k+1)(i-1)}, b_{k(i-1)} \\ &\quad - b_{(k+1)(i-1)}), (-, -), (a_{k(i+1)} - a_{(k+1)(i+1)}, b_{k(i+1)} \\ &\quad - b_{(k+1)(i+1)}), \cdots, (a_{k(n-1)} - a_{(k+1)(n-1)}, b_{k(n-1)} \\ &\quad - b_{(k+1)(n-1)})\} \end{aligned}$$

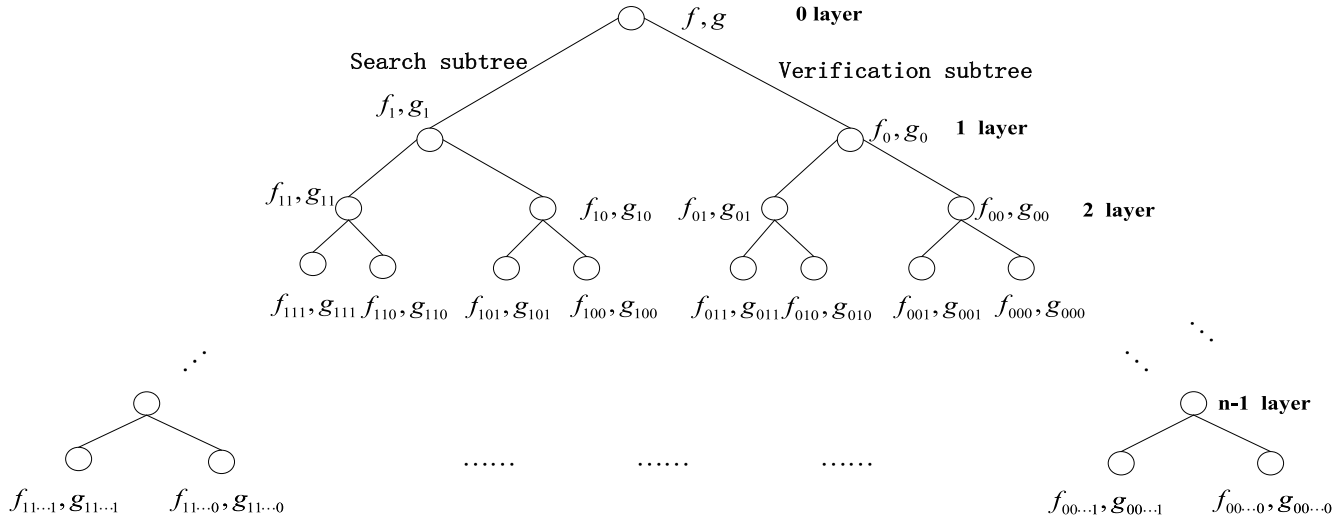

FIGURE 3. n-variable Shannon decomposition binary tree.

Therefore, the signature vector of f_l is complementary to that of f_r relative to that of f_p, and $V_{fl} = V_{fp} - V_{fr}$. ■

Example 2: There is a Boolean function $f(x) = x_0\bar{x}_1\bar{x}_2 + x_0x_2x_3 + x_0x_1\bar{x}_2\bar{x}_3 + \bar{x}_0x_2\bar{x}_3 + \bar{x}_0x_1\bar{x}_2\bar{x}_3$. We decompose f by x_0 and compute the signature vectors of f, f_1 and f_0.

$$V_f = (5, 3), (4, 4), (4, 4), (4, 4)$$
$$V_{f_1} = V_{f_{x_0}} = (-, -), (2, 3), (2, 3), (3, 2)$$
$$V_{f_0} = V_{f_{\bar{x}_0}} = (-, -), (2, 1), (2, 1), (1, 2)$$

The signature vectors of f_1 and f_0 satisfy Corollary 1. Then, we decompose f_1 by the variable $\bar{x}_1$, and the signature vectors of $V_{f_{11}}$ and $V_{f_{10}}$ are as follows:

$$V_{f_{11}} = (-, -), (-, -), (1, 2), (2, 1)$$
$$V_{f_{10}} = (-, -), (-, -), (1, 1), (1, 1)$$

Here, Property 1, Property 2 and Corollary 1 are satisfied.

B. BOOLEAN DECOMPOSITION VERIFICATION

According to the algorithm of Reference [1], the signature vectors of two Boolean functions at any node on the first branch of the transformation search tree are identical and satisfy all variable mappings of transformation T found in the decomposition process. If the signature vectors of the two Boolean functions on all nodes in the Shannon decomposition binary tree are identical and satisfy the variable mappings of transformation T, then transformation T must transform Boolean function f into Boolean function g; that is, f and g are NP equivalents.

However, we know that n-variable Boolean function decomposition forms 2^n branches. If we verify that all nodes of each branch satisfy the candidate transformation, the verification speed is very slow. Therefore, we aim to ensure that all nodes of each branch are satisfied if we verify that all nodes of the first and the $(2^{n-1} + 1)$th branch are satisfied.

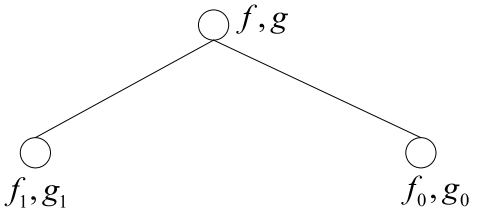

FIGURE 4. Shannon decomposition binary tree of the 2-variable Boolean function.

If we can prove or verify that it is feasible, the Boolean matching speed will be greatly increased.

Here, we state a proposition: *Given two n-variable Boolean functions f and g and an NP transformation T, T follows from the process of NP transformation detection in Reference [1]. If the two signature vectors of each node in the $(2^{n-1} + 1)$th branch of the Shannon decomposition binary tree are identical and satisfy the variable mappings of T, then the two signature vectors of each node of the Shannon decomposition binary tree are identical and satisfy the variable mappings of T.*

If the above proposition is true, then rapid verification can be achieved. We prove the proposition by induction for two n-variable Boolean functions, f and g.

1. If $n = 2$, there are two 2-variable Boolean functions, f and g, and there is a transformation $T = \{x_{i_0}^{\alpha_{i_0}} \rightarrow x_{j_0}^{p_{j_0}}, x_{i_1}^{\alpha_{i_1}} \rightarrow x_{j_1}^{p_{j_1}}\}$ between them detected by the algorithm of Reference [1]. In the transformation T, x_{i_0} and y_{j_0} have a variable mapping, and x_{i_1} and y_{j_1} have a variable mapping. The order of variables in the following signature vectors is consistent with the order in which they appear in transformation T. It is not the sequence $0, 1, \cdots, n-1$. The following descriptions are all similar to this.

Fig. 4 shows a Shannon decomposition binary tree of the 2-variable Boolean function.

1) SETTING THE SIGNATURE VECTORS

Here, we make the following assumptions about the signature vectors of Boolean functions in each node of the Shannon decomposition binary tree in Fig. 3.

$$\begin{aligned} V_f &= \{(a_{0i_0}, b_{0i_0}), (a_{0i_1}, b_{0i_1})\} \\ V_g &= \{(c_{0j_0}, d_{0j_0}), (c_{0j_1}, d_{0j_1})\} \\ V_{f_1} &= \{(-,-), (a_{1i_1}, b_{1i_1})\} \\ V_{g_1} &= \{(-,-), (c_{1j_1}, d_{1j_1})\} \\ V_{f_0} &= \{(-,-), (a'_{1i_1}, b'_{1i_1})\} \\ V_{g_0} &= \{(-,-), (c'_{1j_1}, d'_{1j_1})\} \end{aligned}$$

Because the signature vectors of two Boolean functions at any node on the first branch of this Shannon decomposition binary tree are identical and satisfy all variable mappings in transformation T, there are two equations as follows:

$$\begin{aligned} &① \ V_f = V_g \\ &② \ V_{f_1} = V_{g_1} \end{aligned}$$

According to Corollary 1, the signature vectors of f_1 and f_0 are complementary to f, and the signature vectors of g_1 and g_0 are complementary to g. Therefore, there are four equations as follows:

$$\begin{aligned} &③ \ a'_{1i_1} = a_{0i_1} - a_{1i_1} \\ &④ \ b'_{1i_1} = b_{0i_1} - b_{1i_1} \\ &⑤ \ c'_{1j_1} = c_{0j_1} - c_{1j_1} \\ &⑥ \ d'_{1j_1} = d_{0j_1} - d_{1j_1} \end{aligned}$$

2) PHASE PROBLEM

Next, we discuss whether signature vectors f_0 and g_0 are identical and satisfy the transformation T. The variable mapping between x_{i_0} and y_{j_0} may involve the same phase or opposite phase. The same consideration applies to the variable mapping between x_{i_1} and y_{j_1}. Therefore, there are four different mapping relationships.

1) x_{i_0} and y_{j_0} involve same-phase mapping, and x_{i_1} and y_{j_1} involve same-phase mapping.

According to Equations ① and ②, the following results are obtained.

$$\begin{aligned} a_{0i_0} &= c_{0j_0}, \quad b_{0i_0} = d_{0j_0} \\ a_{0i_1} &= c_{0j_1}, \quad b_{0i_1} = d_{0j_1} \\ a_{1i_1} &= c_{1j_1}, \quad b_{1i_1} = d_{1j_1} \end{aligned}$$

According to Equations ③-⑥, we can obtain $a'_{1i_1} = c'_{1j_1}$ and $b'_{1i_1} = d'_{1j_1}$. Therefore, $V_{f_0} = V_{g_0}$, which satisfies transformation T.

2) x_{i_0} and y_{j_0} involve same-phase mapping, and x_{i_1} and y_{j_1} involve opposite-phase mapping.

According to Equations ① and ②, the following results are obtained.

$$\begin{aligned} a_{0i_0} &= c_{0j_0}, \quad b_{0i_0} = d_{0j_0} \\ a_{0i_1} &= d_{0j_1}, \quad b_{0i_1} = c_{0j_1} \\ a_{1i_1} &= d_{1j_1}, \quad b_{1i_1} = c_{1j_1} \end{aligned}$$

According to Equations ③-⑥, we can obtain $a'_{1i_1} = d'_{1j_1}$ and $b'_{1i_1} = c'_{1j_1}$. Therefore, $V_{f_0} = V_{g_0}$, satisfying transformation T.

3) x_{i_0} and y_{j_0} involve opposite-phase mapping, and x_{i_1} and y_{j_1} involve same-phase mapping.

According to Equations ① and ②, the following results are obtained.

$$\begin{aligned} a_{0i_0} &= d_{0j_0}, \quad b_{0i_0} = c_{0j_0} \\ a_{0i_1} &= c_{0j_1}, \quad b_{0i_1} = d_{0j_1} \\ a_{1i_1} &= c_{1j_1}, \quad b_{1i_1} = d_{1j_1} \end{aligned}$$

According to Equations ③-⑥, we can obtain $a'_{1i_1} = c'_{1j_1}$ and $b'_{1i_1} = d'_{1j_1}$. Therefore, $V_{f_0} = V_{g_0}$, and they satisfy transformation T.

4) x_{i_0} and y_{j_0} have opposite-phase mappings, and x_{i_1} and y_{j_1} have opposite-phase mappings.

According to Equations ① and ②, the following results are obtained.

$$\begin{aligned} a_{0i_0} &= d_{0j_0}, \quad b_{0i_0} = c_{0j_0} \\ a_{0i_1} &= d_{0j_1}, \quad b_{0i_1} = c_{0j_1} \\ a_{1i_1} &= d_{1j_1}, \quad b_{1i_1} = c_{1j_1} \end{aligned}$$

According to Equations ③-⑥, we can obtain $a'_{1i_1} = d'_{1j_1}$ and $b'_{1i_1} = c'_{1j_1}$.

Therefore, $V_{f_0} = V_{g_0}$, and they satisfy the transformation T. Therefore, the proposition is true when $n = 2$.

2. Next, we assume that the hypothesis is true when $n = k$. We must prove that the hypothesis is also true when $n = k+1$.

When we match two *(k+1)*-variable Boolean functions, the Shannon decomposition binary tree is as shown in Fig. 5.

Here, we assume that when $n = k$, our proposition is true, and the NP transformation of two k-variable Boolean functions with NP equivalence is $T = \{x_{i_0}^{\alpha_{i_0}} \to x_{j_0}^{p_{j_0}}, x_{i_1}^{\alpha_{i_1}} \to x_{j_1}^{p_{j_1}}, \cdots, x_{i_{(k-1)}}^{\alpha_{i_{(k-1)}}} \to x_{j_{(k-1)}}^{p_{j_{(k-1)}}}\}$. We have the following assumption: (1) Boolean function f consists of two k-variable Boolean functions f' and f'', and Boolean function g consists of two k-variable Boolean functions g' and g''; (2) f' is NP equivalent to g', f'' is NP equivalent to g'', and both satisfy our hypothesis. Therefore, *(k+1)*-variable Boolean functions f and g can be expressed as follows:

$$\begin{aligned} f &= x_{i_k}^{\alpha_{i_k}} f' + x_{i_k}^{1-\alpha_{i_k}} f'' = x_{i_k}^{\alpha_{i_k}} f_{x_{i_k}^{\alpha_{i_k}}} + x_{i_k}^{1-\alpha_{i_k}} f_{x_{i_k}^{1-\alpha_{i_k}}} \\ g &= x_{j_k}^{p_{j_k}} g' + x_{j_k}^{1-p_{j_k}} g'' = x_{j_k}^{p_{j_k}} g_{x_{j_k}^{p_{j_k}}} + x_{j_k}^{1-p_{j_k}} g_{x_{j_k}^{1-p_{j_k}}} \end{aligned}$$

According to the matching and decomposition process, combined with Figure 5, the layer where node H is located is the last decomposition of the k-variable Boolean function, and the decomposition result is the nodes of the layer where node A is located. For *(k+1)*-Boolean functions, the layer where node H is located is the *(k-1)*-th decomposition; that is, the Boolean functions on node H are $f_H = f_{x_{i_0}^{\alpha_{i_0}} x_{i_1}^{\alpha_{i_1}} \cdots x_{i_{(k-2)}}^{\alpha_{i_{(k-2)}}}}$

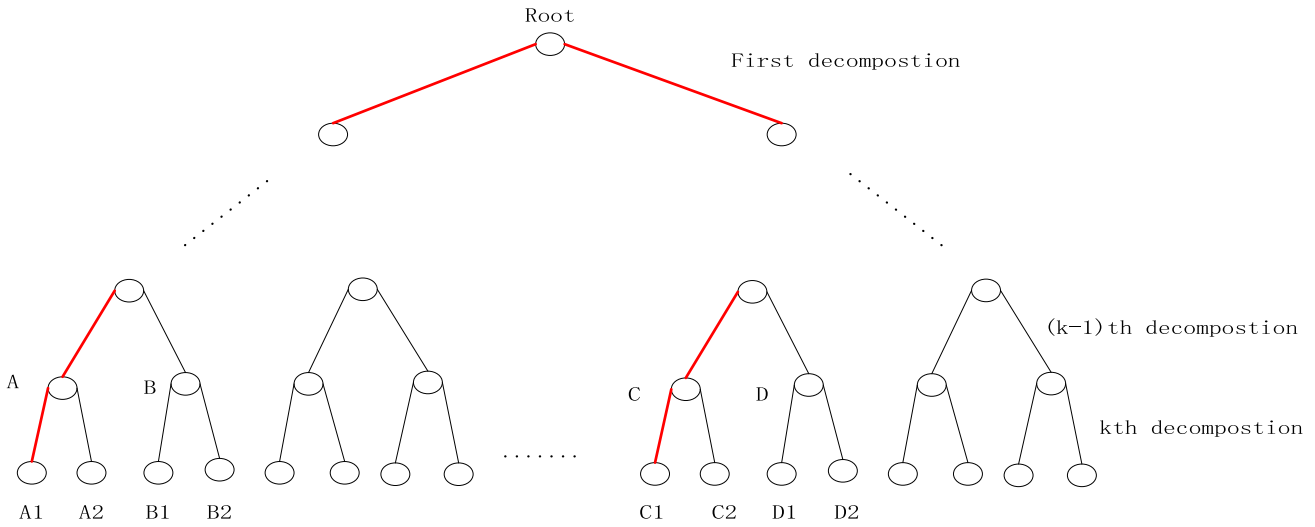

FIGURE 5. Shannon decomposition binary tree of the (k+1)-variable Boolean function.

and $g_H = g^{p_{j0}\, p_{j1} \cdots p_{j(k-2)}}_{x_{j0}\, x_{j1} \cdots x_{j(k-2)}}$. The functions on node H are decomposed by variable mapping $x^{\alpha_{i(k-1)}}_{i(k-1)} \rightarrow x^{\alpha_{j(k-1)}}_{j(k-1)}$ into node A and node B. The Boolean functions on these two nodes are as follows.

$$f_A = f^{\alpha_{i0}\, \alpha_{i1} \cdots \alpha_{i(k-1)}}_{x_{i0}\, x_{i1} \cdots x_{i(k-1)}}, \quad g_A = g^{p_{j0}\, p_{j1} \cdots p_{j(k-1)}}_{x_{j0}\, x_{j1} \cdots x_{j(k-1)}}$$
$$f_B = f^{\alpha_{i0}\, \alpha_{i1} \cdots 1-\alpha_{i(k-1)}}_{x_{i0}\, x_{i1} \cdots x_{i(k-1)}}, \quad g_B = g^{p_{j0}\, p_{j1} \cdots 1-p_{j(k-1)}}_{x_{j0}\, x_{j1} \cdots x_{j(k-1)}}$$

Since we suppose that the hypothesis is true when $n = k$, the two Boolean functions on all nodes from the first level to the *(k-1)*th level have identical signature vectors and satisfy our proposition. Next, we must prove that the signature vectors of the two Boolean functions on all nodes in the k-th layer are equal.

For the convenience of subsequent proof, the two Boolean functions of node A1 are f_{A1} and g_{A1}, the two Boolean functions of node A2 are f_{A2} and g_{A2}, the two Boolean functions of node B1 are f_{B1} and g_{B1}, and the two Boolean functions of node B2 are f_{B2} and g_{B2}, as shown in Fig. 5.

If all signature vectors of two Boolean functions on nodes A1, A2, B1 and B2 are equal, the signature vectors of two Boolean functions on each node of the left subtree are equal and satisfy transformation T. The same applies to the right subtree. If all signature vectors of two Boolean functions on nodes C1, C2, D1 and D2 are equal, the signature vectors of two Boolean functions on each node of the right subtree are equal and satisfy transformation T. Then, it is proven that the hypothesis is satisfied when $n = k + 1$.

This is because the hypothesis is satisfied when $n = k$. We have equations $V_{f_A} = V_{g_A}$ and $V_{f_B} = V_{g_B}$. In the algorithm of Reference [1], any two Boolean functions in the first branch are equal and computed. Therefore, we have the equation $f_{A1} = g_{A1}$.

(1) Proof that $V_{f_{A2}} = V_{g_{A2}}$.

According to the above complementary Corollary 1, $f_{A2} = g_{A2}$ can be derived from the conditions $V_{f_A} = V_{g_A}$ and $f_{A1} = g_{A1}$.

(2) Proof that $V_{f_{B1}} = V_{g_{B1}}$ and $V_{f_{B2}} = V_{g_{B2}}$.

Let $V_{f_B} = \{(-,-),(-,-),\cdots,(a_{i_{k-1}}, b_{i_{k-1}}),(a_{i_k}, b_{i_k})\}$ and $V_{g_B} = \{(-,-),(-,-),\cdots,(c_{j_{k-1}}, d_{j_{k-1}}),(c_{j_k}, b_{j_k})\}$. Because $V_{f_B} = V_{g_B}$, $(a_{i_{k-1}}, b_{i_{k-1}}) = (c_{j_{k-1}}, d_{j_{k-1}})$, and $(a_{i_k}, b_{i_k}) = (c_{j_k}, b_{j_k})$. Here, as in the case of proving n = 2, there are four cases, and we prove only the cases of $x_{i_{k-1}}$ and $y_{j_{k-1}}$ involving same-phase mapping and x_{i_k} and y_{j_k} involving same-phase mapping; the other cases are similar and not reviewed.

According to $V_{f_B} = V_{g_B}$, the following conditions must be satisfied.

$$\begin{aligned} a_{i_{k-1}} &= c_{j_{k-1}} \\ b_{i_{k-1}} &= d_{j_{k-1}} \\ a_{i_k} &= c_{j_k} \\ b_{i_k} &= d_{j_k} \end{aligned}$$

Let the signature vectors of Boolean functions V_{fB1}, V_{gB1}, V_{fB2} and V_{gB2} be as follows:

$$\begin{aligned} V_{f_{B1}} &= \{(-,-),\cdots,(-,-)(a,b)\} \\ V_{g_{B1}} &= \{(-,-),\cdots,(-,-)(c,d)\} \\ V_{f_{B2}} &= \{(-,-),\cdots,(-,-)(\widetilde{a},\widetilde{b})\} \\ V_{g_{B2}} &= \{(-,-),\cdots,(-,-)(\widetilde{c},\widetilde{d})\} \end{aligned}$$

The Boolean functions in nodes B1 and B2 are generated by the same-phase variable mapping between $x_{i_{k-1}}$ of f_B and $x_{j_{k-1}}$ of g_B. Therefore, the following conditions must be satisfied.

$$\begin{aligned} a + b &= a_{i_{k-1}} \\ c + d &= c_{j_{k-1}} \\ \widetilde{a} + \widetilde{b} &= b_{i_{k-1}} \end{aligned}$$

$$\widetilde{c} + \widetilde{d} = d_{j_{k-1}}$$
$$a + \widetilde{a} = a_{i_k}$$
$$b + \widetilde{b} = b_{i_k}$$
$$c + \widetilde{c} = c_{j_k}$$
$$d + \widetilde{d} = d_{j_k}$$

In the last layer of the decomposition binary tree, each Boolean function has only one variable that is not decomposed. The signature vector of each node in the last layer has only one variable signature value. This variable signature value can take only one of (0, 0), (0, 1), (1, 0) and (1, 1); that is, (a, b),$(\widetilde{a}, \widetilde{b})$,$(c, d)$ and $(\widetilde{c}, \widetilde{d})$ can take only one of the above four values

- $(a, b) = (0, 0)$
  According to Conditions 5 and 6, $a + b = c + d$. Therefore, (c, d) must be (0, 0), and $V_{fB1} = V_{gB1}$ is satisfied. We have $(\widetilde{a}, \widetilde{b}) = (a_{i_k}, b_{i_k})$ and $(\widetilde{c}, \widetilde{d}) = (c_{j_k}, b_{j_k})$. According to Conditions 3 and 4, $V_{fB2} = V_{gB2}$ can be deduced.
- $(a, b) = (1, 1)$
  According to Conditions 5 and 6, $a + b = c + d$. Therefore, (c, d) must be (1, 1), and $V_{fB1} = V_{gB1}$ is satisfied. We have $(\widetilde{a}, \widetilde{b}) = (a_{i_k} - 1, b_{i_k} - 1)$ and $(\widetilde{c}, \widetilde{d}) = (c_{j_k} - 1, b_{j_k} - 1)$. According to Conditions 3 and 4, $V_{fB2} = V_{gB2}$ can be deduced.
- $(a, b) = (0, 1)$
  According to Conditions 5 and 6, $a + b = c + d$. Therefore, (c, d) is (0, 1) or (1, 0).
  If $(c, d) = (0, 1)$, $(\widetilde{a}, \widetilde{b}) = (a_{i_k}, b_{i_k} - 1)$, and $(\widetilde{c}, \widetilde{d}) = (c_{j_k}, d_{j_k} - 1)$. According to Conditions 3 and 4, $V_{fB2} = V_{gB2}$ can be deduced.
  If $(c, d) = (1, 0)$, $(\widetilde{a}, \widetilde{b}) = (a_{i_k}, b_{i_k} - 1)$, and $(\widetilde{c}, \widetilde{d}) = (c_{j_k} - 1, d_{j_k})$. According to Conditions 3 and 4, $V_{fB2} = V_{gB2}$ can be deduced.
  The difference between $(c, d) = (0, 1)$ and $(c, d) = (1, 0)$ is the phase. Because our algorithm performs phase collision detection, the phase problem has been resolved.
- $(a, b) = (1, 0)$
  The case of $(a, b) = (1, 0)$ is identical to that of $(a, b) = (0, 1)$.

The proof of the right subtree is identical to that of the left subtree. Now, we can prove that the signature vectors of two Boolean functions in each node on the last layer are equal and satisfy transformation T. Therefore, the hypothesis is true when $n = k + 1$.

In this hypothesis, we assume that the signature vectors of the two Boolean functions on each node of the front k layers are identical. However, are there special circumstances in the matching that do not meet our assumption? Through experiments, it was found that in the worst case, that is, when the positive and negative signatures of many or even all dependent variables are always identical in the decomposition process, errors may occur. Nevertheless, this kind of situation is very rare. Therefore, when we encounter these special circumstances, we resort to the verification method in Reference [1].

Because our algorithm makes full use of symmetry detection, phase conflict detection, and decomposition to obtain new signatures and grouping methods, the new verification method is effective in most cases. Next, we demonstrate this method with two examples.

Example 3: Consider the Boolean functions $f(x) = \overline{x}_0\overline{x}_1x_3 + x_0x_1x_3 + \overline{x}_0x_1\overline{x}_2\overline{x}_3 + x_0\overline{x}_1\overline{x}_2x_3 + \overline{x}_0x_1x_2\overline{x}_3 + x_0\overline{x}_1x_2\overline{x}_3$ and $g(x) = x_0x_1\overline{x}_3 + x_0\overline{x}_1x_3 + \overline{x}_0x_1x_3 + x_0\overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_0\overline{x}_1x_2\overline{x}_3$. We use the signature vector matching Boolean functions f and g without symmetry detection, variable grouping and phase conflict detection and assess the signature vector of the function at each node of the first and $(2^{n-1} + 1)$th branches of the decomposition tree in this process. Here, the order of the variables is x_0,x_1,x_2 and x_3.

1. We compute the signature vectors of Boolean functions f and g.

$$V_f = \{(4, 4), (4, 4), (4, 4), (5, 3)\}$$
$$V_g = \{(5, 3), (4, 4), (4, 4), (4, 4)\}$$

2. According to the above signature vectors, we have variable mapping $x_3 \to x_0$. We decompose Boolean function f by x_3 and Boolean function g by x_0. $f_1 = f_{x_3}$, $f_0 = f_{\overline{x}_3}$, $g_1 = f_{x_0}$, and $g_0 = f_{\overline{x}_0}$. We compute the signature vectors of these four Boolean functions.

$$V_{f_1} = \{(3, 2), (2, 3), (2, 3), (-, -)\}$$
$$V_{g_1} = \{(-, -), (2, 3), (2, 3), (2, 3)\}$$
$$V_{f_0} = \{(1, 2), (2, 1), (2, 1), (-, -)\}$$
$$V_{g_0} = \{(-, -), (2, 1), (2, 1), (2, 1)\}$$

3. From the above results, we obtain $V_{f_1} = V_{g_1}$ and $V_{f_0} = V_{g_0}$. There are three candidate variable mappings: $x_0 \to \overline{x}_1, x_0 \to \overline{x}_2$ and $x_0 \to \overline{x}_3$. Here, we try $x_0 \to \overline{x}_1$ and obtain the following Boolean functions.

$$f_{11} = x_0 f_{1x_0}, \quad g_{11} = \overline{x}_1 g_{1\overline{x}_1},$$
$$f_{10} = \overline{x}_0 f_{1\overline{x}_0}, \quad g_{10} = x_1 g_{1x_1},$$
$$f_{01} = x_0 f_{0x_0}, \quad g_{01} = \overline{x}_1 g_{0\overline{x}_1},$$
$$f_{00} = \overline{x}_0 f_{0\overline{x}_0}, \quad g_{00} = x_1 g_{0x_1}.$$

We compute the signature vectors of these eight Boolean functions.

$$V_{f_{11}} = \{(-, -), (2, 1), (1, 2), (-, -)\}$$
$$V_{g_{11}} = \{(-, -), (-, -), (1, 2), (2, 1)\}$$
$$V_{f_{10}} = \{(-, -), (0, 2), (1, 1), (-, -)\}$$
$$V_{g_{10}} = \{(-, -), (-, -), (1, 1), (0, 2)\}$$
$$V_{f_{01}} = \{(-, -), (0, 1), (1, 0), (-, -)\}$$
$$V_{g_{01}} = \{-, -), (-, -), (1, 0), (0, 1)\}$$
$$V_{f_{00}} = \{(-, -), (2, 0), (1, 1), (-, -)\}$$
$$V_{g_{00}} = \{(-, -), (-, -), (1, 1), (2, 0)\}$$

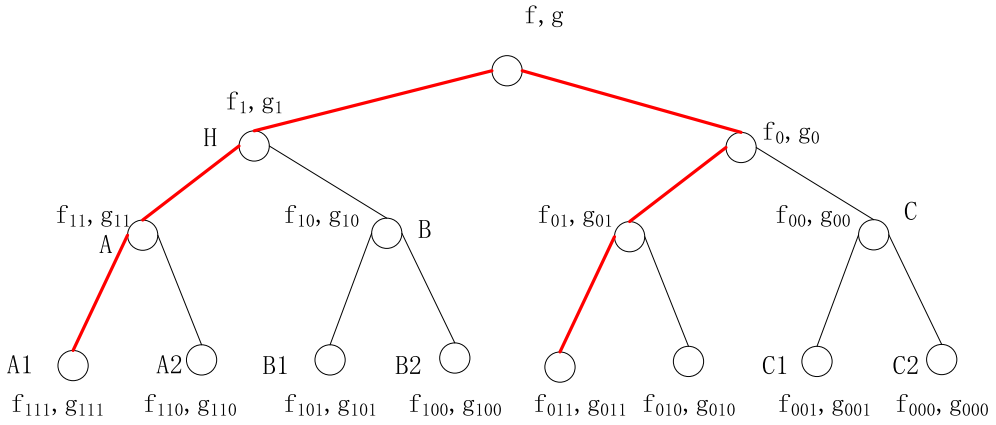

FIGURE 6. Shannon decomposition binary tree of Example 3.

4. From the above results, $V_{f_{11}} = V_{g_{11}}$,$V_{f_{10}} = V_{g_{10}}$, $V_{f_{01}} = V_{g_{01}}$, and $V_{f_{00}} = V_{g_{00}}$. According to $V_{f_{11}} = V_{g_{11}}$, we have two variable mappings: $x_1 \to \overline{x}_2$ and $x_1 \to x_3$. We try $x_1 \to \overline{x}_2$.

$$
\begin{aligned}
&f_{111} = x_1 f_{11x_1}, \quad g_{111} = \overline{x}_2 g_{11\overline{x}_2},\ f_{110} = \overline{x}_1 f_{11\overline{x}_1},\\
&g_{110} = x_2 g_{11x_2}, \quad f_{101} = x_1 f_{10x_1},\ g_{101} = \overline{x}_2 g_{10\overline{x}_2},\\
&f_{100} = \overline{x}_1 f_{10\overline{x}_1}, \quad g_{100} = x_2 g_{10x_2}\ f_{011} = x_1 f_{01x_1},\\
&g_{011} = \overline{x}_2 g_{01\overline{x}_2}, \quad f_{010} = \overline{x}_1 f_{01\overline{x}_1},\ g_{010} = x_2 g_{01x_2},\\
&f_{001} = x_1 f_{00x_1}, \quad g_{001} = \overline{x}_2 g_{00\overline{x}_2},\ f_{000} = \overline{x}_1 f_{00\overline{x}_1},\\
&g_{000} = x_2 g_{00x_2}.
\end{aligned}
$$

We compute the signature vectors of these eight Boolean functions.

$$
\begin{aligned}
V_{f_{111}} &= \{(-,-),(-,-),(1,1),(-,-)\}\\
V_{g_{111}} &= \{(-,-),(-,-),(-,-),(1,1)\}\\
V_{f_{110}} &= \{(-,-),(-,-),(0,1),(-,-)\}\\
V_{g_{110}} &= \{(-,-),(-,-),(-,-),(1,0)\}\\
V_{f_{101}} &= \{(-,-),(-,-),(-,-),(-,-)\}\\
V_{g_{101}} &= \{(-,-),(-,-),(-,-),(0,1)\}\\
V_{f_{100}} &= \{(-,-),(-,-),(1,1),(-,-)\}\\
V_{g_{100}} &= \{(-,-),(-,-),(-,-),(0,1)\}\\
V_{f_{011}} &= \{(-,-),(-,-),(-,-),(-,-)\}\\
V_{g_{011}} &= \{(-,-),(-,-),(-,-),(-,-)\}\\
V_{f_{010}} &= \{(-,-),(-,-),(1,0),(-,-)\}\\
V_{g_{010}} &= \{(-,-),(-,-),(-,-),(0,1)\}\\
V_{f_{001}} &= \{(-,-),(-,-),(1,1),(-,-)\}\\
V_{g_{001}} &= \{(-,-),(-,-),(-,-),(1,0)\}\\
V_{f_{000}} &= \{(-,-),(-,-),(-,-),(-,-)\}\\
V_{g_{000}} &= \{(-,-),(-,-),(-,-),(1,0)\}
\end{aligned}
$$

From the above results, $V_{f_{111}} = V_{g_{111}}$, $V_{f_{110}} = V_{g_{110}}$, $V_{f_{101}} \neq V_{g_{101}}$, $V_{f_{100}} \neq V_{g_{100}}$, $V_{f_{011}} = V_{g_{011}}$, $V_{f_{010}} = V_{g_{010}}$, $V_{f_{001}} \neq V_{g_{001}}$, and $V_{f_{000}} \neq V_{g_{000}}$. The binary tree generated by the decomposition is shown in Fig. 6.

From the above results, the following anomalies are observed.

(1) In the second branch, $V_{f_{10}} = V_{g_{10}}$, but $V_{f_{101}} \neq V_{g_{101}}$, and $V_{f_{100}} \neq V_{g_{100}}$.

(2) In the fourth branch, $V_{f_{00}} = V_{g_{00}}$, but $V_{f_{001}} \neq V_{g_{001}}$, and $V_{f_{000}} \neq V_{g_{000}}$.

By observing the decomposed variables, we find the causes of these two problems as follows.

(1) Decomposition variable x_1 of Boolean function f is a symmetry variable. However, decomposition variable x_2 of Boolean function g is a nonsymmetry variable. The two variables are of different types. Variable mapping $x_1 \to \overline{x}_2$ is incorrect. In our algorithm, we perform symmetry detection. Therefore, our algorithm must not attempt variable mapping $x_1 \to \overline{x}_2$ but rather variable mapping $x_1 \to x_3$. When we attempt $x_1 \to x_3$, the above situation does not occur, and our proposition is satisfied.

(2) In node H, variable x_1 of Boolean function f and variable x_2 of Boolean function g are of the same phase. However, they have the opposite phase in node A. In our algorithm, we check for a phase conflict. If a variable mapping is correct, the relation of their phase remains unchanged. Thus, our algorithm also excludes this mapping early.

Example 4: Consider Boolean functions $f(x) = \overline{x}_1 x_2 + x_1\overline{x}_4 + \overline{x}_2 x_3$ and $g(x) = x_1 x_2 + \overline{x}_3\overline{x}_4 + \overline{x}_2\overline{x}_3 + \overline{x}_2 x_3 x_4$. As shown in Example 3, we review the detection and verification process.

1. We compute the signature vectors of Boolean functions f and g.

$$
\begin{aligned}
V_f &= \{(5,6),(6,5),(7,4),(4,7)\}\\
V_g &= \{(7,4),(5,6),(4,7),(6,5)\}
\end{aligned}
$$

2. According to the above signature vectors, there are two candidate variable mappings: $\overline{x}_1 \to \overline{x}_2$ and $\overline{x}_1 \to x_4$. We attempt the first variable mapping. We decompose Boolean function f by $\overline{x}_1$ and Boolean function g by $\overline{x}_2$.

$f_1 = \overline{x}_1 f_{\overline{x}_1}$, $f_0 = x_1 f_{x_1}$, $g_1 = \overline{x}_2 f_{\overline{x}_2}$, and $g_0 = x_2 f_{x_2}$. We compute the signature vectors of these four

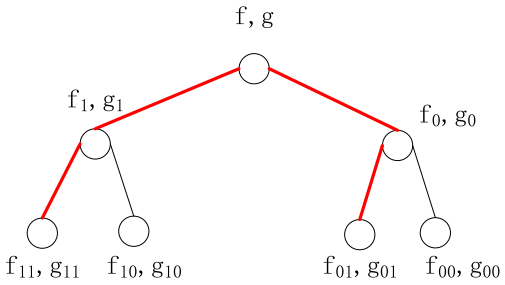

FIGURE 7. Shannon decomposition binary tree of Example 4.

Boolean functions.

$$\begin{aligned} V_{f_1} &= \{(-,-),(4,2),(4,2),(3,3)\} \\ V_{g_1} &= \{(3,3),(-,-),(2,4),(4,2)\} \\ V_{f_0} &= \{(-,-),(2,3),(3,2),(1,4)\} \\ V_{g_1} &= \{(4,1),(-,-),(2,3),(2,3)\} \end{aligned}$$

3. From the above results, we obtain $V_{f_1} = V_{g_1}$ and $V_{f_0} = V_{g_0}$. There are two candidate variable mappings: $x_2 \rightarrow \overline{x}_3$ and $x_2 \rightarrow x_4$. Here, we try $x_2 \rightarrow \overline{x}_3$.

$$\begin{aligned} f_{11} &= x_2 f_{1x_2}, \quad g_{11} = \overline{x}_3 g_{1\overline{x}_3}, \; f_{10} = \overline{x}_2 f_{1\overline{x}_2}, \; g_{10} = x_3 g_{1x_3}, \\ f_{01} &= x_2 f_{0x_2}, \quad g_{01} = \overline{x}_3 g_{0\overline{x}_3}, \; f_{00} = \overline{x}_2 f_{0\overline{x}_2}, \; g_{00} = x_3 g_{0x_3}. \end{aligned}$$

We compute the signature vectors of these eight Boolean functions.

$$\begin{aligned} V_{f_{11}} &= \{(-,-),(-,-),(2,2),(2,2)\} \\ V_{g_{11}} &= \{(2,2),(-,-),(-,-),(2,2)\} \\ V_{f_{10}} &= \{(-,-),(-,-),(2,0),(1,1)\} \\ V_{g_{10}} &= \{(1,1),(-,-),(-,-),(2,0)\} \\ V_{f_{01}} &= \{(-,-),(-,-),(1,1),(0,2)\} \\ V_{g_{01}} &= \{2,1),(-,-),(-,-),(1,2)\} \\ V_{f_{00}} &= \{(-,-),(-,-),(2,1),(1,2)\} \\ V_{g_{01}} &= \{(2,0),(-,-),(-,-),(1,1)\} \end{aligned}$$

From the above results, $V_{f_{11}} = V_{g_{11}}$,$V_{f_{10}} = V_{g_{10}}$, $V_{f_{01}} \neq V_{g_{01}}$, and $V_{f_{00}} \neq V_{g_{00}}$. The binary tree generated by decomposition is shown in Fig. 7.

The above results do not satisfy our proposition. According to Reference [1], the signature values of two variables of a correct variable mapping are synchronously changed. In this example, variable x_2 of Boolean function f and variable x_4 of Boolean function g violate this principle. Therefore, our algorithm must not generate variable mapping $x_2 \rightarrow \overline{x}_3$.

V. MATCHING ALGORITHM

Our algorithm is mainly composed of three parts. Parts 1 and 2 are the methods of transformation detection, which are similar to the algorithm in Reference [1] with minor differences. Part 3 is our proposed verification algorithm.

A. BOOLEAN MATCHING INITIALISATION AND JUDGMENT OF THE NOT OPERATION ON OUTPUT

(1) For two n-variable Boolean functions, f and g, we use a zeroth-order signature to determine whether the NOT operation is on the output. If $|f| = 2^n - |g| \wedge |f| \neq |g|$, there is a NOT operation on the output. If $|f| = |g| = 2^{n-1}$, there may or may not be a NOT operation on the output. Handling this special case is identical to the algorithm in Reference [1]. If f is not NP equivalent to g, the algorithm will match f and $\overline{g}$. (2) We calculate the 1^{st} structural signature vectors of f and g, symmetry variable check and grouping. (3) We compare the two 1^{st} structural signature vectors. If they are different, they are not NPN equivalent. Otherwise, we execute the algorithm of part 2 to search the candidate transformation [1].

The differences from Reference [1] in part 1 are as follows: we add (1) an independent variable check; (2) a comparison of the symmetric variable structure of two Boolean functions; and (3) a comparison of the number of independent variables of two Boolean functions. These differences can help determine the inequivalence of two Boolean functions as early as possible.

B. CANDIDATE TRANSFORMATION DETECTION

We extend the method of candidate transformation detection in Reference [1]. In our algorithm, independent variables are detected in the initialization and preliminary evaluation procedures. During the candidate transformation detection, we create variable mappings between independent variables, and the independent variable mapping set is the minimum variable mapping set. We first use the variables of an independent variable mapping to perform the Shannon decomposition.

C. VERIFY CANDIDATE TRANSFORMATIONS

Based on the hypothesis of this paper, we propose a new verification method as follows.

When our algorithm detects a candidate transformation, this transformation is stored in a linked list. Let the candidate transformation be $T = \{x_{i0}^{\alpha_{i0}} \rightarrow x_{j0}^{p_{j0}}, x_{i1}^{\alpha_{i1}} \rightarrow x_{j1}^{p_{j1}}, \cdots, x_{i(n-1)}^{\alpha_{i(n-1)}} \rightarrow x_{j(n-1)}^{p_{j(n-1)}}\}$. Our algorithm orders the variable mappings of this linked list by the value of $|f_{x_{i_k}^{1-\alpha_{i_k}}}|$, where $k \in \{0, 1, \cdots, n-1\}$. After sorting, the variables of the first variable mapping have the maximum signature value. The goal of this task is to ensure that Boolean functions, f and g, can be decomposed as many times as possible. Therefore, our algorithm reorders the remaining variable mappings after each variable mapping verification.

Condition D_0 checks whether all variable mappings of candidate transformation are verified. When D_0 is true, the current candidate transformation is verified as correct. Three cases indicate that the current candidate transformation does not satisfy the NP equivalent condition.

(1) If UPDATE(f, g, $cube\_f$, $cube\_g$) = 0, the two new SS vectors are not identical. The algorithm returns 0.

Algorithm 1 Transformation Verification

Input: f, g, $cube\_f$, $cube\_g$, map_list
Output: 0 or 1

```
Compute the number of verified mappings
if D0 then
  Return 1
else
  if UPDATE(f, g, cube_f, cube_g)=0 then
    Return 0
  else
    if V'_{p->x} ≠ V'_{p->y} then
      Return 0
    else
      if D1 then
        Return 0
      else
        Update cube_f and cube_g
        Return VERIFY(cube_f, cube_g, p- > next)
      end if
    end if
  end if
end if
```

(2) If Condition D_1 is true, the two variables of the current variable mapping have a phase collision. The algorithm returns 0.

(3) If $V'_{p->x} \neq V'_{p->y}$, the group numbers of the two variables in the current variable map are different. This shows that their signature values do not change synchronously and that the mapping is wrong. The algorithm returns 0.

When the current variable mapping is verified to be true, our algorithm verifies the next variable mapping. If the current variable mapping is verified as false, our algorithm performs candidate transformation detection to obtain the next candidate transformation.

However, this verification method cannot verify the worst-case Boolean matching. In the worst case, most variables satisfy $|f_{x_i}| = |f_{\bar{x}_i}|$; however, a case can occur where the phase of some variables can never be determined. Therefore, in this case, we use the verification method of Reference [1].

VI. EXPERIMENTAL RESULTS

The algorithm described in this paper is implemented in a hardware environment with a 3.3-GHz central processing unit (CPU) and 4 GB of RAM. The runtime is the time spent by the CPU when matching.The runtimes are reported in seconds. The algorithm in this paper is implemented in C.

To demonstrate the effectiveness of our proposed algorithm, we conduct an experiment using large circuit sets. We perform experiments on three data sets. The data in the first data set come from the MCNC benchmark. The second and third data sets are randomly generated. However, the Boolean functions in the third data set satisfy the condition $|f| = |g| = 2^{n-1}$. All of these Boolean circuits have 7-22 input variables. For every Boolean function, whether from the MCNC benchmark or randomly generated, we use the program to automatically generate random NPN transformations and use these transformations to generate equivalent Boolean functions, which form our test data set. Then, we compare the results of our algorithm with those of Reference [1].

TABLE 1. Boolean matching runtimes on the first data set.

#I	#MIN	#MAX	#AVG	#MIN of Ref. [1]	#MAX of Ref. [1]	#AVG of Ref. [1]	#PR
7	0.00001	0.00019	0.00007	0.00002	0.00035	0.00011	40.0%
8	0.00002	0.00021	0.00013	0.00013	0.00059	0.00030	57.6%
9	0.00003	0.00062	0.00021	0.00005	0.00111	0.00044	53.3%
10	0.00007	0.00148	0.00033	0.00011	0.00188	0.00063	48.3%
11	0.00008	0.00199	0.00036	0.00009	0.00217	0.00079	54.6%
12	0.00011	0.00484	0.00051	0.00018	0.00625	0.00090	43.0%
13	0.00064	0.01743	0.00075	0.00089	0.02500	0.00140	46.7%
14	0.00038	0.01837	0.00167	0.00073	0.01809	0.00360	53.8%
15	0.00025	0.03509	0.00284	0.00039	0.05908	0.00478	40.6%
16	0.00042	0.25204	0.05505	0.00064	0.31375	0.11360	51.5%
17	0.00071	0.41088	0.06753	0.00102	0.65791	0.11360	40.6%
18	0.00095	0.95697	0.12132	0.00192	1.63740	0.23490	48.4%
19	0.00291	1.01509	0.46216	0.00391	1.94918	0.88743	47.9%
20	0.00402	2.17304	0.53560	0.00904	4.75490	1.28150	58.2%
21	0.06674	3.26995	1.44901	0.16699	5.74320	3.73066	61.2%
22	0.21096	5.31913	2.50095	1.29078	16.44130	6.24368	59.9%

The following three tables are the matching runtime comparison results between our algorithm and the algorithm in Reference [1] on the above three different sets of Boolean functions. Table 1 shows the experimental results on the first data set, i.e., the equivalent MCNC benchmark circuit. Table 2 shows the results tested on the second data set, which is a randomly generated circuit set. Table 3 shows the results tested on the third data set, which is a randomly generated circuit set that satisfies $|f| = |g| = 2^{n-1}$.

The first column (#I) is the number of input variables. The #MIN, #MAX, and #AVG represent the minimum, maximum and average matching runtimes, respectively where the average matching time is the arithmetic mean of the runtimes. The second, third and fourth columns are the matching runtime results of our algorithm, and the fifth, sixth and seventh columns are the matching runtime results of Reference [1]. #PR is the percentage of the improvement in average runtime between our algorithm and the algorithm in Reference [1].

Table 1 shows that the average matching speed of our algorithm is 51% higher than that of Reference [1] on the first Boolean function set. The results of Table 2 show that the average matching speed of our algorithm is increased by 65.5% compared with that of Reference [1] on the second Boolean function set. The average matching speed of our algorithm on the third Boolean function set is 72.1% faster than that of Reference [1]. Overall, the average matching speed of our algorithm is increased by 62.9% compared with that of Reference [1]. From the above experimental results, it can be seen that the speed increases with the increase in the number of variables. In Table 2, when the number of variables is 7 and 8, the matching speed is slightly lower than

TABLE 2. Boolean matching runtimes on the second data set.

#I	#MIN	#MAX	#AVG	#MIN of Ref. [1]	#MAX of Ref. [1]	#AVG of Ref. [1]	#PR
7	0.000064	0.000086	0.000032	0.000033	0.000122	0.000030	-7.7%
8	0.000077	0.000113	0.000063	0.000035	0.000176	0.000060	-5.0%
9	0.000047	0.000062	0.000053	0.000042	0.000245	0.000078	31.5%
10	0.000073	0.000099	0.000084	0.000061	0.000491	0.000140	40.2%
11	0.000095	0.000148	0.000118	0.000093	0.001300	0.000268	55.9%
12	0.000101	0.000135	0.000117	0.000110	0.001643	0.000320	63.3%
13	0.000103	0.000168	0.000134	0.000120	0.000713	0.000338	60.4%
14	0.000118	0.000172	0.000141	0.000230	0.003597	0.000609	76.9%
15	0.000150	0.000255	0.000189	0.000355	0.003071	0.000731	74.1%
16	0.000135	0.000214	0.000174	0.000562	0.003916	0.000956	81.8%
17	0.000158	0.000250	0.000202	0.005144	0.007390	0.001745	88.4%
18	0.000220	0.000315	0.000265	0.002187	0.022871	0.005380	95.1%
19	0.000220	0.000320	0.000269	0.004002	0.053468	0.007601	96.5%
20	0.000265	0.000413	0.000329	0.007600	0.745964	0.017245	98.1%
21	0.000286	0.000378	0.000325	0.015034	0.033574	0.047286	99.3%
22	0.000319	0.000406	0.000362	0.029658	0.066031	0.050146	99.3%

TABLE 3. Boolean matching runtimes on the third data set.

#I	#MIN	#MAX	#AVG	#MIN of Ref. [1]	#MAX of Ref. [1]	#AVG of Ref. [1]	#PR
7	0.00012	0.00020	0.00015	0.00008	0.00079	0.00018	15.5%
8	0.00014	0.00025	0.00019	0.00014	0.00109	0.00024	20.0%
9	0.00015	0.00028	0.00022	0.00013	0.00256	0.00070	68.3%
10	0.00035	0.00073	0.00054	0.00071	0.00504	0.00118	54.0%
11	0.00109	0.00193	0.00147	0.00159	0.00957	0.00243	39.5%
12	0.00109	0.00193	0.00147	0.00059	0.01470	0.00715	79.4%
13	0.00181	0.00281	0.00228	0.01326	0.02894	0.01312	82.6%
14	0.00297	0.00559	0.00401	0.03208	0.06498	0.02976	86.5%
15	0.00680	0.01122	0.00867	0.07196	0.13799	0.06567	86.8%
16	0.01358	0.02175	0.01720	0.15473	0.26420	0.15203	88.7%
17	0.02896	0.04198	0.03529	0.38728	0.73046	0.34597	89.8%
18	0.07945	0.11354	0.09701	0.70077	0.82638	0.79530	87.8%
19	0.18504	0.21969	0.20244	1.70311	1.97701	1.85297	89.1%
20	0.46446	0.51885	0.49157	3.92404	4.73963	4.23075	88.4%
21	1.07895	1.19584	1.13003	8.86246	10.53660	9.60024	88.2%
22	3.25928	3.74444	3.46888	26.40660	38.05870	31.74640	89.1%

that of Reference [1]. These results show that our verification method is more effective for Boolean function matching with larger variables.

VII. CONCLUSION

This paper proposes a Boolean matching verification algorithm. The experimental results verify that our algorithm can effectively improve the Boolean matching speed. Compared with the algorithm of Reference [1], our algorithm performs 62.9% faster on average. Our algorithm can be applied well to technology mapping and cell-library binding. In future work, we plan to research how to resolve NPN Boolean matching for the worst Boolean function.

REFERENCES

[1] J. Zhang, G. Yang, W. N. N. Hung, Y. Zhang, and J. Wu, "An efficient NPN Boolean matching algorithm based on structural signature and Shannon expansion," *Cluster Comput.*, vol. 22, no. S3, pp. 7491–7506, May 2019, doi: 10.1007/s10586-018-1787-x.

[2] C. Wu, C. Hsu, and K. Y. Khoo, "ICCAD-2016 CAD contest in non-exact projective NPNP Boolean matching and benchmark suite," in *Proc. 35th Int. Conf. Computer-Aided Design*, Austin, TX, USA, Nov. 2016, pp. 1–5.

[3] V. K. Mishra, M. Dixit, T. Choudhary, A. Goswami, M. Kaur, O. Cheikhrouhou, and H. Hamam, "A heuristic-driven and cost effective majority/minority logic synthesis for post-CMOS emerging technology," *IEEE Access*, vol. 9, pp. 168689–168702, 2021, doi: 10.1109/ACCESS.2021.3079310.

[4] Y. Guo, X. Wang, Q. Hong, and Y. Zhang, "A serial access scheme design on memristor-CMOS hybrid memory," *IEEE Access*, vol. 8, pp. 35031–35037, 2020, doi: 10.1109/ACCESS.2020.2974015.

[5] J. Zhang, G. Yang, W. N. N. Hung, T. Liu, X. Song, and M. A. Perkowski, "A group algebraic approach to NPN classification of Boolean functions," *Theory Comput. Syst.*, vol. 63, no. 6, pp. 1278–1297, Aug. 2019, doi: 10.1007/s00224-018-9903-0.

[6] X. Zhou, L. Wang, P. Zhao, and A. Mishchenko, "Fast adjustable NPN classification using generalized symmetries," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Dublin, Ireland, Aug. 2018, pp. 1–16.

[7] A. Petkovska, M. Soeken, G. De Micheli, P. Ienne, and A. Mishchenko, "Fast hierarchical NPN classification," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Lausanne, Germany, Aug. 2016, pp. 1–4.

[8] Z. Wang, X. Zeng, J. Wu, and G. Yang, "A method for determining the affine equivalence of Boolean functions," *IEEE Access*, vol. 7, pp. 156326–156337, 2019, doi: 10.1109/ACCESS.2019.2949310.

[9] Z. Huang, L. Wang, Y. Nasikovskiy, and A. Mishchenko, "Fast Boolean matching based on NPN classification," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Kyoto, Japan, Dec. 2013, pp. 310–313.

[10] A. Adbollahi and M. Pedram, "Symmetry detection and Boolean matching utilizing a signature-based canonical form of Boolean functions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 6, pp. 1128–1137, May 2008, doi: 10.1109/TCAD.2008.923256.

[11] J. Zhang, "A canonical-based NPN Boolean matching algorithm utilizing Boolean difference and cofactor signature," *IEEE Access*, vol. 5, pp. 27777–27785, 2017, doi: 10.1109/ACCESS.2017.2778338.

[12] D. Debnath and T. Sasao, "Efficient computation of canonical form under variable permutation and negation for Boolean matching in large libraries," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 89, no. 12, pp. 3443–3450, Dec. 2006, doi: 10.1093/ietfec/e89-a.12.3443.

[13] G. Agosta, F. Bruschi, G. Pelosi, and E. al, "A transform-parametric approach to Boolean matching' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 6, pp. 805–817, Jun. 2009, doi: 10.1109/TCAD.2009.2016547.

[14] S. F. Vinokurov, L. Ryabets, S. I. Todikov, and A. S. Frantseva, "Algorithm for constructing minimal representations of multiple-output Boolean functions in the reversible logic circuits," in *Proc. 20th IEEE Int. Conf. Soft Comput. Meas. (SCM)*, St.Peterburg, Russia, May 2017, pp. 541–543.

[15] S. F. Vinokurov and A. S. Frantseva, "Complexity of representations of multiple-output Boolean functions in the reversible logic circuits," in *Proc. 25th IEEE Int. Conf. Soft Comput. Meas. (SCM)*, St. Peterburg, Russia, May 2016, pp. 374–376.

[16] X. Zhou, L. Wang, and A. Mishchenko, "Fast exact NPN classification by co-designing canonical form and its computation algorithm," *IEEE Trans. Comput.*, vol. 69, no. 9, pp. 1293–1307, Sep. 2020, doi: 10.1109/TC.2020.2971466.

[17] A. Abdollahi, "Signature based Boolean matching in the presence of don't cares," in *Proc. 45th Annu. Conf. Design Autom. (DAC)*, Anaheim, CA, USA, Jun. 2008, pp. 642–647.

[18] J. Zhang, G. Yang, W. Hung, J. Wu, and Y. Zhu, "A new pairwise NPN Boolean matching algorithm based on structural difference signature," *Symmetry*, vol. 11, no. 1, p. 27, Dec. 2018, doi: 10.3390/sym11010027.

[19] J. Zhang, L. Ni, S. Zheng, H. Liu, X. Zou, F. Wang, and G. Luo, "Enhanced fast Boolean matching based on sensitivity signatures pruning," in *Proc. ICCAD*, Munich, Germany, Nov. 2021, pp. 1–9.

[20] F. Wang, J. Zhang, L. Wu, W. Zhang, and G. Luo, "Search space reduction for the non-exact projective NPNP Boolean matching problem," in *Proc. ISCAS*, Baltimore, MD, USA, May 2017, pp. 1–4.

[21] K. Wang, C. Chan, and J. Liu, "Simulation and SAT-based Boolean matching for large Boolean networks," *Proc. 46th Int. Conf. ACM/IEEE Design Automation Conf.*, San Francisco, CA, USA, Jul. 2009, pp. 396–401.

[22] W. Xiu-Qin and Y. Yang, "New approach of exploiting symmetry in SAT-based Boolean matching for FPGA technology mapping," in *Proc. IEEE Int. Conf. Veh. Electron. Saf.*, Dongguan, China, Jul. 2013, pp. 282–285.

[23] Y. Matsunaga, "Accelerating SAT-based Boolean matching for heterogeneous FPGAs using one-hot encoding and CEGAR technique," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 99, no. 7, pp. 1374–1380, Jul. 2016, doi: 10.1587/transfun.E99.A.1374.

[24] C. Zhang, H. Yu, L. Wang, 'Accelerating Boolean matching using Bloom filter," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 93, no. 10, pp. 1775–1781, Oct. 2010.
[25] K. Wang and C. Chan, "Incremental learning approach and SAT model for Boolean matching with don't cares," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA. Dec. 2007, pp. 234–239.
[26] C. F. Lai, J. Jiang, and K. H. Wang, "BooM: A decision procedure for Boolean matching with abstraction and dynamic learning," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, Jun. 2010, pp. 499–504.
[27] C. F. Lai, J. Jiang, and K. H. Wang, "Boolean matching of function vectors with strengthened learning," in *Proc. ICCAD*, San Jose, CA, USA, Nov. 2010, pp. 596–601.
[28] M. Damiani and A. Y. Selchenko, "Boolean technology mapping based on logic decomposition," in *Proc. 16th Symp. Integr. Circuits Syst. Design (SBCCI)*, Sao Paulo, Brazil, Sep. 2003, pp. 35–40.
[29] S. Ji and H.-A. Jacobsen, "A-tree: A dynamic data structure for efficiently indexing arbitrary Boolean expressions," in *Proc. Int. Conf. Manage. Data*, Jun. 2021, pp. 817–829.
[30] C.-W. Pui, P. Tu, H. Li, G. Chen, and E. F. Y. Young, "A two-step search engine for large scale Boolean matching under NP3 equivalence," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jeju, South Korea, Jan. 2018, pp. 592–598.
[31] G. Agosta, F. Bruschi, G. Pelosi, and D. Sciuto, "A unified approach to canonical form-based Boolean matching," in *Proc. 44th ACM/IEEE Design Autom. Conf.*, San Diego, CA, USA, Jun. 2007, pp. 841–846.
[32] J. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch, "Linear cofactor relationships in Boolean functions," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 1011–1023, Jun. 2006, doi: 10.1109/TCAD.2005.855951.

JULING ZHANG received the M.S. degree from the School of Information and Communication Engineering, Xinjiang University, China, in 2007, and the Ph.D. degree from the School of Computer Science and Engineering, University of Electronic Science and Technology of China, China, in 2018. She is currently an Associate Professor with the Xinjiang University of Finance and Economics. Her research interests include logic synthesis, information security risk assessments, and internet public sentiment management.

WENQIANG GUO received the B.S., M.S., and Ph.D. degrees from the School of Computer Science and Technology, Dalian University of Technology, China, in 1998, 2004, and 2007, respectively. He is currently a Full Professor at the Xinjiang University of Finance and Economics. He has published more than 100 journals and conference papers. His research interests include internal security, renewable energy systems, and smart grids.

GUOWU YANG (Member, IEEE) received the B.S. degree from the University of Science and Technology of China, in 1989, the M.S. degree from the Wuhan University of Technology, in 1994, and the Ph.D. degree in electrical and computer engineering from Portland State University, in 2005. He worked 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. He has published more than 100 journals and conference papers. His research interests include verification, logic synthesis, quantum computing, and machine learning.

YIXIN ZHU received the Ph.D. degree from the School of Information and Software Engineering, University of Electronic Science and Technology of China, in 2015. He is currently an Associate Professor at the Xinjiang University of Finance and Economics. His current research interests include information security and complex network propagation dynamics.

XIAOYI LV received the M.S. degree from the School of Information and Communication Engineering, Xinjiang University, China, in 2006, and the Ph.D. degree from the School of Electronic and Information Engineering, Xi'an Jiaotong University, China, in 2010. He is currently a Full Professor at the School of Software, Xinjiang University. His research interests include logical synthesis and artificial intelligence.