symmetry

MDPI

Article

A New Pairwise NPN Boolean Matching Algorithm Based on Structural Difference Signature

Juling Zhang [1,2], Guowu Yang [2,3,*], William N. N. Hung [4], Jinzhao Wu [3,5,*] and Yixin Zhu [1,6]

1 The School of Computer Science and Engineering, University of Xinjiang Finance and Economics, Urumqi 830012, China; zjlgj@163.com (J.Z.); xjzhuyixin@163.com (Y.Z.)
2 The Big Data Research Center, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
3 Guangxi Key Laboratory of Hybrid Computation and IC Design Analysis, Guangxi University for Nationalities, Nanning 530006, China
4 Synopsys Inc., Mountain View, CA 94043, USA; William.Hung@synopsys.com
5 The School of Computer and Electronic Information, Guangxi University, Nanning 530004, China
6 Network and Data Security Key Laboratory of Sichuan Province, University of Electronic Science and Technology of China, Chengdu 611731, China
* Correspondence: ygwuestc@163.com (G.Y.); gxmdwjzh@aliyun.com (J.W.); Tel.: +86-180-3047-0403 (G.Y.); +86-151-9917-6691 (J.W.)

Received: 21 September 2018; Accepted: 21 December 2018; Published: 29 December 2018

Abstract: In this paper, we address an NPN Boolean matching algorithm. The proposed structural difference signature (SDS) of a Boolean function significantly reduces the search space in the Boolean matching process. The paper analyses the size of the search space from three perspectives: the total number of possible transformations, the number of candidate transformations and the number of decompositions. We test the search space and run time on a large number of randomly generated circuits and Microelectronics Center of North Carolina (MCNC) benchmark circuits with 7–22 inputs. The experimental results show that the search space of Boolean matching is greatly reduced and the matching speed is obviously accelerated.

Keywords: NPN Boolean matching; structural difference signature vector; independent variable; variable symmetry

1. Introduction

Boolean equivalence classification and matching constitute a long-standing and open problem. The authors of [1,2] applied a group algebraic approach to NP and NPN Boolean equivalence classification. Reference [2] computed the classification results for 10 inputs. Affine equivalence classification is also an important field of study with applications in logic synthesis and cryptography [3]. All Boolean functions in an equivalence class are equivalent to each other. NPN Boolean matching determines whether two Boolean functions are equivalent under input negation and/or permutation and/or output negation. This paper studies NPN Boolean matching for single-output completely specified Boolean functions.

NPN Boolean matching is an important research topic that can be applied to a number of applications in integrated circuit design, such as technology mapping, cell library binding and logic verification [4]. When a Boolean circuit is functionally NPN-equivalent to another Boolean circuit, one of these circuits can be realized by means of the other. There are $n!2^{n+1}$ NPN transformations for a n-variable Boolean function. If Boolean function f is NPN-equivalent to Boolean function g, there must be a NPN transformation that can transform f to g. On the contrary, no NPN transformation can transform f to g. The purpose of our proposed algorithm is to find the NPN transformation that can

transform Boolean function f to g as early as possible. Based on the structural signature (SS) vector in our previous study [5], we proposes a new combined signature vector, i.e., SDS vector. In this paper, Boolean difference sigture is introduced into SS vector to form SDS vector. The new signature vector SDS is better able to distinguish variables and reduce the research space for NPN Boolean matching. Experimental results show that the search space is reduced by more than 48% compared with [5] and that the run time of our algorithm is reduced by 42% and 80% compared with [5,6], respectively.

In the following, Section 2 introduces relevant works on NPN-equivalence matching. Section 3 introduces some terminology and notation. Section 4 describes the proposed algorithm in detail. In Section 5, we present experimental results to demonstrate the effectiveness of our algorithm. Section 6 concludes the paper.

2. Related Works

Many methods can be exploited to solve the problem of NPN Boolean matching. The main results of such research are focused on four methods: (1) algorithms on canonical forms; (2) pairwise matching algorithms using signatures; (3) algorithms based on satisfiability (SAT) and (4) algorithms based on spectral analysis.

Each method has its own advantages. In canonical-form-based matching algorithms, the canonical form of each Boolean circuit of cell library is stored in advance. When cell-library binding is implemented, the canonical form of each Boolean circuit to be matched is computed and compared with the canonical forms of each Boolean circuits in the cell library via a hash table. All Boolean functions in an equivalence class have the same canonical form. The canonical form of each equivalence class has a special value. References [6–12] studied Boolean matching based on canonical forms and attained significant achievements.

Reference [12] reported P-equivalence matching for 20-input Boolean functions. The canonical forms considered in reference [12] was the binary strings with the maximal scores in lexicographic comparison. Reference [7] devised a procedure to canonicalize a threshold logic function and judged equivalence of two threshold logic functions by their canonicalized linear inequalities. Based on the canonical form of Boolean function, the reference [8] reduced the number of configuration bits in an FPGA architecture. The authors of [10,11] proposed fast Boolean matching based on NPN Boolean classification; their canonical form has the maximal truth table. The authors of [6] proposed new canonical forms based on signatures.

A pairwise matching algorithm searches the NPN transformations between two Boolean functions using signatures, which is a semi-exhaustive search algorithm. The merit of this method is that once it finds a transformation that can prove the equivalence of two Boolean functions, other transformations will not be checked. The authors of [4,5,13,14] proposed Boolean matching algorithms based on pairwise matching and used binary decision diagrams (BDDs) to represent Boolean functions. The authors of [5] proposed a structural signature vector to search the transformations between two Boolean functions and implemented NPN Boolean matching for 22 inputs. In pairwise matching algorithms, signatures are usually used as a necessary condition for judging whether two Boolean functions are equivalent, and variable symmetry is commonly utilized to reduce the search space. Symmetric attributes are used in many fields. Reference [15] studied the symmetries of the unitary Lie group. The variable symmetric attributes of Boolean function are widely used in NPN Boolean equivalence matching. In reference [5], the search space was reduced and the matching speed was improved by means of structural signatures, variable symmetry, phase collision check and variable grouping.

Since a SAT solver can help solve the problem of NPN Boolean matching and because many quick SAT solvers can be utilized, many Boolean matching algorithms based on SAT have emerged in recent years. The authors of [16–20] studied SAT-based Boolean matching. Based on graphs, simulation and SAT, Matsunaga [16] achieved PP-equivalence Boolean matching with larger inputs and outputs. The authors of [17,18] studied Boolean matching for FPGAs utilizing SAT technology.

Cong et al. [19] used the implicant table to derive the SAT formulation and achieved significant improvements. The authors of [20] combined simulation and SAT to perform P-equivalent Boolean matching for large Boolean functions. Compared with studies based on the previous three methods, studies on Boolean matching that use spectral techniques are fewer in number. Moore et al. [21] presented an NPN Boolean matching algorithm using Walsh spectra. The authors of [22] utilized Haar spectra to check the equivalence of two logic circuits.

Regardless of which method is used, the key to Boolean matching is to reduce the search space. It is universally known that the search space for exhaustive NPN Boolean matching is $O(n!2^{n+1})$. In the methods discussed above, many strategies are used to reduce the search space. The authors of [6] used general signatures and symmetry to reduce the search space.

Based on our previous study [5], we propose a new combined signature, i.e., the structural difference signature. We present a new pairwise algorithm based on the following conditions: (1) two NP-equivalent Boolean functions have the same SDS vectors; (2) two variables of a variable mapping have the same SDS values; and (3) two groups of Boolean functions Shannon decomposed with splitting variables are NP-equivalent.

3. Terminology and Notation

Let $f(x_0, x_1, \cdots, x_{n-1})$ and $g(x_0, x_1, \cdots, x_{n-1})$ be two single-output completely specified Boolean functions. The problem to be solved in this paper is to determine whether f is NPN-equivalent to g. Some related terminology has been introduced in [5,6].

An NP transformation T is composed of input negations and/or permutations. It can also be expressed as a group of variable mappings. In reference [5], the mapping from the variable x_i of f to the variable x_j of g, φ_i, can be classified into two cases: (1) x_i maps to x_j with the same phase, in which case the mapping is $x_i \to x_j$ or $\overline{x}_i \to \overline{x}_j$; or (2) x_i maps to x_j with the opposite phase, in which case the mapping is $x_i \to \overline{x}_j$ or $\overline{x}_i \to x_j$. A same-phase relation indicates no input negation, whereas an opposite-phase relation indicates input negation. A same-phase variable mapping between the variables x_i and x_j is abbreviated as $i \to j-0$, and an opposite-phase variable mapping between the variables x_i and x_j is abbreviated as $i \to j-1$. For two NPN-equivalent Boolean functions f and g, there may be an output negation when $|f| = |\overline{g}|$.

Definition 1. *(NPN equivalence) Two Boolean functions f and g are NPN equivalent, $f \cong g$, if and only if there exists an NP transformation T that satisfies $f(TX) = g(X)$ or $f(TX) = \overline{g(X)}$.*

As a general signature, the cofactor signature is widely applied in NPN Boolean matching. The cofactor signature of $f(X)$ with respect to x_i ($\overline{x}_i$) is $|f_{x_i}|$ ($|f_{\overline{x}_i}|$) [5].

Reference [5] proposed SS vector. The SS value of f with respect to x_i, V_i, is $(|f_{x_i}|, |f_{\overline{x}_i}|, |C_i|, C_i, G_i)$. $(|f_{x_i}|, |f_{\overline{x}_i}|)$ is the 1st signature of the variable x_i, and C_i and $|C_i|$ are the symmetry mark, and G_i is group mark. According to their symmetry properties, the variables of a Boolean function are classified as either asymmetric and symmetric. An asymmetric variable may have a single-mapping set or a multiple-mapping set. The variable mapping set of the asymmetric variable x_i is denoted by χ_i. Similarly, a symmetric variable may have a single symmetry-mapping set or a multiple symmetry-mapping set. The symmetry-mapping set of the symmetry class C_i is denoted by S_i, and the symmetry mapping between C_i and C_j is denoted by $C_i \to C_j$. The literal ψ_i represents a group of two or more variable mappings generated by $C_i \to C_j$.

A P transformation does not change the cofactor signature of a variable. However, an N transformation changes the order of the positive and negative cofactor signatures without changing their numerical values. Therefore, we do not consider the order of the positive and negative cofactor signatures when comparing the 1st signature values of two variables. A variable x_i may be transformed into an arbitrary variable x_j, $0 \le j \le n-1$; therefore, we also do not consider the order of the variables when we compare two SS vectors.

Given two NP-equivalent Boolean functions f and g with a variable mapping $x_i \rightarrow x_j$ between them, we have the following four facts: (1) $V_f = V_g$ and $V_i = V_j$; (2) The Boolean functions decomposed with x_i and x_j using the Shannon expansion must be NP equivalent. Specifically, $x_i f_{x_i}$ is NP equivalent to $x_j g_{x_j}$, and $\overline{x}_i f_{\overline{x}_i}$ is NP equivalent to $\overline{x}_j g_{\overline{x}_j}$; (3) x_i and x_j are either both asymmetric variables or both symmetric variables; (4) If there is a variable mapping between x_l of f and x_h of g, then the SS values of x_l must be the same as those of x_h no matter how many times the Boolean functions f and g are decomposed [5].

Two Boolean functions f and g may undergo one or more transformations in the process of matching. A transformation consists of n variable mappings. The algorithm of [5] and the algorithm presented in this paper detect all possible transformations between f and g according to their SS and SDS vectors, respectively.

4. The Proposed Algorithm

The goal of the proposed algorithm is to reduce the size of the search space as much as possible, thereby improving the speed of NPN Boolean matching.

4.1. Boolean Difference

For n inputs, there are 2^{2^n} different Boolean functions. Many Boolean functions have one or more independent variables. Whether a variable x_i of f is independent can be determined using cofactors.

The Boolean difference of a Boolean function f with respect to x_i, f'_{x_i}, is the Boolean function $f_{x_i} \oplus f_{\overline{x}_i}$, where $f_{x_i} = f[x_i \leftarrow 1]$ and $f_{\overline{x}_i} = f[x_i \leftarrow 0]$ [23].

Definition 2. *(Boolean difference signature) The Boolean difference signature of a Boolean function f with respect to x_i, $|f'_{x_i}|$, is the number of minterms of f'_{x_i}.*

When a variable x_i of f is NP transformed into x_j $(\overline{x}_j)$, its Boolean difference signature does not change. Thus, Boolean difference signature, like cofactor signature, can be used to distinguish variables.

Example 1. *Consider an 5-input Boolean function* $f(X) = x_0\overline{x}_2\overline{x}_3 + \overline{x}_0x_1x_2\overline{x}_3 + x_0x_1x_3x_4 + \overline{x}_0\overline{x}_1x_3\overline{x}_4 + x_0\overline{x}_1\overline{x}_3\overline{x}_4 + x_0x_2x_3x_4 + \overline{x}_0\overline{x}_1\overline{x}_2x_3 + \overline{x}_0x_2\overline{x}_3x_4 + x_0x_1x_2x_3 + \overline{x}_0\overline{x}_2x_3\overline{x}_4$. *Let us compute the 1st signature and the Boolean difference signature of each variable.*

The 1st signatures of x_0, x_1, x_2, x_3 and x_4 are $(9,7)$, $(8,8)$, $(8,8)$, $(8,8)$ and $(8,8)$, respectively. The variable x_1 is symmetric to x_4. The Boolean difference signatures of the variables are 32, 12, 20, 28 and 12, respectively. From the 1st signatures and a symmetry check, we can distinguish variables x_0, x_1 and x_4. Variables x_2 and x_3 are both asymmetric variables and have the same 1st signature values. If we only utilize on their 1st signatures, the variables x_2 and x_3 cannot be distinguished. However, these two variables have different Boolean difference signatures. Thus, the variables x_2 and x_3 are actually different and can be distinguished.

Definition 3. *(Independent variable) A variable x_i of a Boolean function f is an independent variable if it satisfies $|f'_{x_i}| = 0$.*

NP transformations do not change the independence of a variable. Thus, an independent variable is still an independent variable after NP transformation.

Definition 4. *(Independent-variable set) The independent-variable set of a Boolean function f, D_f, is a set that consists of all independent variables of f.*

Lemma 1. *Two NPN-equivalent Boolean functions f and g have the same number of independent variables.*

Proof. If the Boolean function f is NPN-equivalent to g, then f and g are in the same NPN equivalence class. There must exist an NP transformation T that can transform f into g or $\overline{g}$. After NP transformation, an independent variable is still an independent variable. Therefore, it can be deduced that f and g have the same number of independent variables. □

Property 1. *The cofactor signature of a Boolean function f with respect to its variable x_i is $|f_{x_i}| = |f_{\overline{x}_i}| = \frac{1}{2}|f|$ when the variable x_i is an independent variable.*

Proof. Since $f'_{x_i} = f_{x_i} \oplus f_{\overline{x}_i}$ and $|f'_{x_i}| = 0$ and $|f_{x_i}| + |f_{\overline{x}_i}| = |f|$, it holds that $|f_{x_i}| = |f_{\overline{x}_i}| = \frac{1}{2}|f|$. □

Because the positive cofactor signature is the same as the negative cofactor signature for an independent variable, the phases of independent variables cannot determined by using the phase assignment method presented in [5,6]. However, independent variables have no influence on a Boolean function. Thus, the proposed algorithm assigns a positive phase to all independent variables.

Example 2. *Consider two 6-input Boolean functions $f(X) = \overline{x}_0x_1\overline{x}_2 + \overline{x}_0x_1x_2x_3 + x_0\overline{x}_1\overline{x}_2 + x_0\overline{x}_1x_2x_3$ and $g(X) = \overline{x}_0\overline{x}_1\overline{x}_3x_4 + \overline{x}_0x_1x_3x_4 + x_0\overline{x}_1\overline{x}_3 + x_0x_1x_3$. Let us compute the SS vectors, Boolean difference signatures and independent-variable sets.*

The SS vectors of f and g are as follows:
$V_f = \{(12, 12, 2, 0, 1), (12, 12, 2, 0, 1), (8, 16, 2, 2, 0), (16, 8, 2, 2, 0), (12, 12, 2, 4, 1), (12, 12, 2, 4, 1)\}$,
$V_g = \{(16, 8, 2, 0, 0), (12, 12, 2, 1, 1), (12, 12, 2, 2, 1), (12, 12, 2, 1, 1), (16, 8, 2, 0, 0), (12, 12, 2, 2, 1)\}$.
The Boolean difference signatures of the variables of f are 48, 48, 16, 16, 0 and 0. The Boolean difference signatures of the variables of g are 16, 48, 0, 48, 16 and 0. The independent-variable sets of f and g are $D_f = \{x_4, x_5\}$ and $D_g = \{x_2, x_5\}$.

Definition 5. *(Independent mapping set) The independent mapping set between Boolean functions f and g is $D = \{\varphi_i : x_i \to x_j | x_i \in D_f, x_j \in D_g\}$.*

Consider two NP-equivalent Boolean functions f and g with independent-variable sets of $D_f = \{x_{i_1}, x_{i_2}, \cdots x_{i_k}\}$ and $D_g = \{x_{j_1}, x_{j_2}, \cdots x_{j_k}\}$, respectively. If we do not consider the symmetry and independence of variables, then there are $2^k k!$ groups of different variable mappings between D_f and D_g according to their 1st signatures. However, according to the properties of independent variables, we need to consider only the positive phase and create one independent-mapping set $\{x_{i_1} \to x_{j_1}, x_{i_2} \to x_{j_2}, \cdots x_{i_k} \to x_{j_k}\}$. Therefore, the search space is reduced significantly if there are independent variables in the Boolean functions.

In Example 2, the Boolean function f has the three symmetry classes $C_0 = \{x_0, x_1\}$, $C_2 = \{x_2, x_3\}$ and $C_4 = \{x_4, x_5\}$, and the Boolean function g has the three symmetry classes $C_0 = \{x_0, x_4\}$, $C_1 = \{x_1, x_3\}$ and $C_2 = \{x_2, x_5\}$ if we do not consider the Boolean difference signatures. The symmetry class C_0 of f can be mapped to the symmetry classes C_1 and C_2 of g using the method of reference [5]. In other words, the symmetry classes C_0 and C_4 of Boolean function f cannot be distinguished. However, the variables in C_0 and C_4 of Boolean function f have different Boolean difference signatures. Thus, if we consider the Boolean difference signatures when searching the variable mappings, the symmetry class C_0 of f can be mapped only to the symmetry class C_1 of g, and the symmetry class C_4 of f can be mapped only to the symmetry class C_2 of g. There exists an independent-mapping set $\{x_4 \to x_2, x_5 \to x_5\}$.

Definition 6. *(Structural difference signature vector) An n-input Boolean function f has a structural difference signature (SDS) vector $V_f = \{V_0, V_1, \cdots, V_{n-1}\}$, where $V_i = (|f_{x_i}|, |f_{\overline{x}_i}|, |C_i|, C_i, G_i, |f'_{x_i}|)$.*

The algorithm presented in [5] groups variables by their 1st signature values. The algorithm proposed in this paper groups variables by their 1st signature values and Boolean difference signatures. We define the '$<$' relation between x_i and x_j as follows.

Definition 7. *($<$) The variables x_i and x_j have the relation $x_i < x_j$ if one of the following two cases is satisfied: (1) $(|f_{x_i}|, |f_{\overline{x}_i}|) < (|f_{x_j}|, |f_{\overline{x}_j}|)$ or (2) $(|f_{x_i}|, |f_{\overline{x}_i}|) = (|f_{x_j}|, |f_{\overline{x}_j}|) \wedge |f'_{x_i}| < |f'_{\overline{x}_i}|$.*

The group numbers of the variables are generated with the above '$<$' relation. The SDS vectors of f and g in example 2 are as follows: $V_f = \{(12, 12, 2, 0, 1, 48), (12, 12, 2, 0, 1, 48), (8, 16, 2, 2, 0, 16), (16, 8, 2, 2, 0, 16), (12, 12, 2, 4, 2, 0), (12, 12, 2, 4, 2, 0)\}$ and $V_g = \{(16, 8, 2, 0, 0, 16), (12, 12, 2, 1, 1, 48), (12, 12, 2, 2, 2, 0), (12, 12, 2, 1, 1, 48), (16, 8, 2, 0, 0, 16), (12, 12, 2, 2, 2, 0)\}$. In the process of grouping variables of Boolean function f, we first compare their 1st signature and do not consider the order of positive and negative cofactor signature. Therefore, variables x_2 and x_3 are grouped into group 0. The varibales in symmetry class C_0 and C_4 have the same 1st signature, so we compare their Boolean difference signature. Because the Boolean difference signature of variables in C_0 is greater than that of variables in symmetry class C_4, the group number of variables x_0 and x_1 are 1 and the group number of variables x_4 and x_5 are 2.

The SDS vector is a new signature vector that consists of the SS vector plus a Boolean difference mark. The variable mapping search and the transformation detection of our algorithm are based on two facts: (1) two NP-equivalent Boolean functions have the same SDS vectors; and (2) there exists a possible variable mapping between the variables x_i and x_j if they have the same SDS values.

4.2. SDS-Based Boolean Matching Algorithm

NPN Boolean matching is defined as follows:

Given two Boolean functions f and g, if there exists an NP transformation T that satisfies $f(TX) = g(X)$ or $f(TX) = \overline{g(X)}$, then f is NPN-equivalent to g.

Before searching the variable mappings, the proposed algorithm first determines whether there is an output negation for Boolean function f. If there is, then our algorithm will match f and $\overline{g}$. The method of identifying the presence of an output negation is the same as that in reference [5]. If $|f| = |g| \wedge |f| \neq |\overline{g}|$, then there is no output negation. There is an output negation if $|f| \neq |g| \wedge |f| = |\overline{g}|$. There is a tie if $|f| = |g| \wedge |f| = |\overline{g}|$. Our algorithm handles first the condition without output negation and then the condition with output negation if f is not NP equivalent to g.

The algorithm will terminate when it finds a transformation T that satisfies $f(TX) = g(X)$ $(\overline{g(X)})$ or when all candidate transformations have been checked and found not to satisfy $f(TX) = g(X)$ $(\overline{g(X)})$. The algorithm will attempt all possible variable mappings, and thus, it will certainly find an NP transformation T between two NP-equivalent Boolean functions f and g $(\overline{g})$.

The pseudo-code for NPN Boolean matching is given in Procedure 1.

In Procedure 1, *trans_list* is a tree that stores the NP transformations generated in the process of transformation detection. A candidate transformation is an unabridged branch in *trans_list*. *sp_f* and *sp_g* are the decomposition expressions for f and g, respectively. After the existence of an output negation has been determined, Procedure 1 calls Handle_SDS() to detect the NP transformations between f and g $(\overline{g})$ and judge the NP equivalence of f and g $(\overline{g})$.

Any one NP transformation between the Boolean functions f and g $(\overline{g})$ is composed of n variable mappings. Thus, the proposed algorithm searches variable mappings and generates NP transformations. In this paper, the necessary condition for two Boolean functions to be judged NP equivalent is that they must have the same SDS vector. For a variable mapping to be established between x_i and x_j, these two variables must satisfy the following conditions:

(1) x_i and x_j have the same 1st signature values, i.e., $(|f_{x_i}|, |f_{\overline{x}_i}|) = (|f_{x_j}|, |f_{\overline{x}_j}|) \vee (|f_{x_i}|, |f_{\overline{x}_i}|) = (|f_{\overline{x}_j}|, |f_{x_j}|)$.

(2) x_i and x_j have the same Boolean difference signature, i.e., $|f'_{x_i}| = |f'_{x_j}|$.

(3) x_i and x_j have the same symmetry class cardinality, i.e., $|C_i| = |C_j|$.
(4) x_i and x_j have the same group number, i.e., $G_i = G_j$.

Procedure 1 NPN Boolean Matching.

```
Input: f and g
Output: 0 or 1
  function MATCHING(f, g)
      Create BDD of f and g
      sp_f = bddtrue, sp_g = bddtrue, trans_list = NULL
      Compute |f| and |g|
      if |f| = |g| then
          if |f| ≠ |ḡ| then
              Return Handle_SDS(f, g)
          else
              if Handle_SDS(f, g)=1 then
                  Return 1
              else
                  Return Handle_SDS(f, ḡ)
              end if
          end if
      else
          if |f| = |ḡ| then
              Return Handle_SDS(f, ḡ)
          else
              Return 0
          end if
      end if
  end function
```

In the process of the variable mapping search, Handle_SDS() searches the variable mappings for each variable that has not been identified. A variable is identified when its phase and variable mappings are determined in a transformation. After searching all variable mapping sets, Handle_SDS() selects the minimal variable mapping set to handle. The minimal variable mapping set is the one with the lowest cardinality. There are eight possible cases for the variable mapping set of the variable x_i of f, as follows.

(1) The variable x_i is an asymmetric variable. The phase of x_i is determined, and there is only one variable x_j of g that has the same SDS values as those of x_i. The variable mapping set of x_i is a single-mapping set. $\chi_i = \{i \to j - k\}, k \in \{0, 1\}$, and $|\chi_i| = 1$.

(2) The variable x_i is an asymmetric variable. There exist multiple variables $x_{j_1}, x_{j_2}, \cdots, x_{j_m}$ of g, where $m \geq 2$, that have the same SDS values as those of x_i, and their phases are determined. The variable mapping set of x_i is a multiple-mapping set. $\chi_i = \{i \to j_1 - k_1, i \to j_2 - k_2, \cdots, i \to j_m - k_m\}$, where $k_1, k_2, \cdots, k_m \in \{0, 1\}$, and $|\chi_i| = m$.

(3) The variable x_i is an asymmetric variable. There exist one or more variables $x_{j_1}, x_{j_2}, \cdots, x_{j_m}$ of g, where $m \geq 1$, that have the same SDS values as those of x_i, and their phases are not determined. The variable mapping set of x_i is a multiple-mapping set. $\chi_i = \{i \to j_1 - 0, i \to j_1 - 1, i \to j_2 - 0, i \to j_2 - 1, \cdots, i \to j_m - 0, i \to j_m - 1\}$, and $|\chi_i| = 2m$.

(4) The variable x_i is a symmetric variable, and its symmetry class is $C_i = \{x_i, x_{i_1}, x_{i_2}, \cdots, x_{i_{m-1}}\}$. There exists only one symmetry class $C_j = \{x_j, x_{j_1}, x_{j_2}, \cdots, x_{j_{m-1}}\}$ of g whose variables have the same SDS values as those of the variables in C_i, where $|C_i| = |C_j|$, and the phase of x_i is determined. The variable mapping set of x_i, S_i, is a single symmetry-mapping set, i.e., $|S_i| = 1$. There exists one group of variable mappings $\{i \to j - k, i_1 \to j_1 - k_1, i_2 \to j_2 - k_2, \cdots, i_{m-1} \to j_{m-1} - k_{m-1}\}$, where $k, k_1, k_2, \cdots, k_{m-1} \in \{0, 1\}$, between C_i and C_j.

(5) The variable x_i is a symmetric variable, and its symmetry class is $C_i = \{x_i, x_{i_1}, x_{i_2}, \cdots, x_{i_{m-1}}\}$. There is only one symmetry class $C_j = \{x_j, x_{j_1}, x_{j_2}, \cdots, x_{j_{m-1}}\}$ of g whose variables have the same SDS values as those of the variables in C_i, where $|C_i| = |C_j|$, and the phase of x_i is not determined. The variable mapping set of x_i, S_i, is a multiple symmetry-mapping set: $|S_i| = 2$. There are two groups of variable mappings, $\{i \to j - 0, i_1 \to j_1 - k_1, i_2 \to j_2 - k_2, \cdots, i_{m-1} \to j_{m-1} - k_{m-1}\}$, where

$k_1, k_2, \cdots, k_{m-1} \in \{0,1\}$, and $\{i \to j-1, i_1 \to j_1 - p_1, i_2 \to j_2 - p_2, \cdots, i_{m-1} \to j_{m-1} - p_{m-1}\}$, where $p_1, p_2, \cdots, p_{m-1} \in \{0,1\}$, between C_i and C_j.

When the variable symmetry is checked, the phase relation between two symmetric variables is known. The variable mapping relations between C_i and C_j can be generated in the following way.

We first consider the case in which x_i and x_j have the same phase, i.e., there exists a variable mapping $i \to j-0$. A variable mapping $i_1 \to j_1 - 0$ exists in two cases: (1) x_i is symmetric to x_{i_1} and x_j is symmetric to x_{j_1} or (2) x_i is symmetric to $\overline{x}_{i_1}$ and x_j is symmetric to $\overline{x}_{j_1}$. A variable mapping $i_1 \to j_1 - 1$ exists in two cases: (1) x_i is symmetric to x_{i_1} and x_j is symmetric to $\overline{x}_{j_1}$ or (2) x_i is symmetric to $\overline{x}_{i_1}$ and x_j is symmetric to x_{j_1}. Then, we consider the case in which x_i and x_j have the opposite phase, i.e., there exists a variable mapping $i \to j-1$. A variable mapping $i_1 \to j_1 - 1$ exists in two cases: (1) x_i is symmetric to x_{i_1} and x_j is symmetric to x_{j_1} or (2) x_i is symmetric to $\overline{x}_{i_1}$ and x_j is symmetric to $\overline{x}_{j_1}$. A variable mapping exists $i_1 \to j_1 - 0$ in two cases: (1) x_i is symmetric to x_{i_1} and x_j is symmetric to $\overline{x}_{j_1}$ or (2) x_i is symmetric to $\overline{x}_{i_1}$ and x_j is symmetric to x_{j_1}. Thus, two groups of variable mappings between C_i and C_j will be generated via this method.

(6) The variable x_i is a symmetric variable, and its symmetry class is C_i. There exist multiple symmetry classes $C_{j_1}, C_{j_2}, \cdots, C_{j_m}$, where $2 \le m \le \left\lfloor \frac{n}{2} \right\rfloor$, whose variables have the same SDS values as the variables in C_i, where $|C_i| = |C_{j_1}| = |C_{j_2}| = \cdots = |C_{j_m}|$, and the phase of x_i is determined. The variable mapping set of x_i, S_i, is a multiple symmetry-mapping set: $|S_i| = m$. There exists one group of variable mappings between C_i and each C_{j_p}, where $p \in \{1, 2, \cdots, m\}$.

(7) The variable x_i is a symmetric variable, and its symmetry class is C_i. There exist one or more symmetry classes $C_{j_1}, C_{j_2}, \cdots, C_{j_m}$, where $1 \le m \le \left\lfloor \frac{n}{2} \right\rfloor$, whose variables have the same SDS values as those of the variables of C_i, where $|C_i| = |C_{j_1}| = |C_{j_2}| = \cdots = |C_{j_k}|$, and the phase of x_i is not determined. The variable mapping set of x_i, S_i, is a multiple symmetry-mapping set: $|S_i| = 2m$. There exist two groups of variable mappings between C_i and each C_{j_p}, where $p \in \{1, 2, \cdots, m\}$.

(8) The variable x_i is an independent variable. The variable mapping set of x_i is an independent mapping set.

All possible variable mapping sets are listed above. To generate an NP transformation, n variable mappings are needed for $x_1, x_2, \cdots, x_n$. Each node in the NP transformation tree, *trans_list*, represents a variable mapping, and all nodes in a given layer belong to the same variable mapping set. The methods for handling the variable mapping sets are as follows.

(1) If it is the first computation of SDS vectors, a check for independent variables is performed. If there are one or more independent variables, an independent-mapping set is created and added to *trans_list*, and the minimal variable mapping set is then sought among the remaining variables. If there are no independent variables, Handle_SDS() searches the variable mapping sets for all variables.

(2) If the current variable mapping set of x_i is a single-mapping set, our algorithm adds the variable mapping in χ_i to *trans_list*. The variable x_i is identified.

(3) If the current variable mapping set of x_i is a single symmetry-mapping set and x_i belongs to C_i, where $|C_i| = m$, then the group ψ_i of variable mappings of S_i is added to *trans_list*. To the NP transformation tree, m layers are added, where each layer contains a variable mapping node. The variables in the symmetry class C_i are all identified.

(4) If the current variable mapping set of x_i is a multiple-mapping set or a multiple symmetry-mapping set, then the cardinalities of the variable mapping sets are computed, and the minimal variable mapping set is recorded.

After searching all variable mapping sets, as in reference [5], our algorithm updates the two decomposition expressions sp_f and sp_g in the case of a single-mapping set or a single symmetry-mapping set. Otherwise, our algorithm handles the minimal variable mapping set. If the cardinality m of the minimal variable mapping set satisfies $m \ge 2$, then m branches will be generated in *trans_list*. Each branch is handled in order.

The purpose of Procedure 2 is to search the variable mappings for all possible NP transformations. In the process of recursive_search, Procedure 2 uses the same methods applied in [5] to find and prune error NP transformation branches. That is, the current branch will be pruned if the two SDS vectors are not the same or if the current variable mapping has a phase collision.

The pseudo-code for Procedure 2 is as follows.

Procedure 2 recursive_search.

Input: f, g, sp_f, sp_g, and $trans\_list$
Output: 0 or 1
function HANDLE_SDS($f, g, sp\_f, sp\_g, trans\_list$)
if D_1 **then**
return VERIFY(f, g, T)
end if
UPDATE(f, g, sp_f, sp_g)
if $V_f \neq V_g$ **then**
return 0
end if
if D_2 **then**
Compute D_f and D_g
if $NotEmpty(D_f)$ **then**
Add independent mappings to $trans\_list$
end if
end if
min_number=32768
for all $x_i \in f(x)$ **do**
if D_3 **then**
Continue
end if
for all $x_j \in g(x)$ **do**
Search variable mappings
end for
Compute $\chi_i(S_i)$
if D_4 **then**
for all $\varphi_j(\psi_j) \in \chi_i(S_i)$ **do**
if D_5 **then**
return 0
else
Add $\varphi_j(\psi_j)$ to $trans\_list$
end if
end for
$min\_number = 1$
else
if $|\chi_i|(|S_i|) < min\_number$ **then**
$min = i$
end if
end if
end for
if D_6 **then**
Update sp_f and sp_g
Return Handle_SDS($f, g, sp\_f, sp\_g, trans\_list$)
else
for all $\varphi_j(\psi_j) \in \chi_{min}(S_{min})$ **do**
if D_5 **then**
continue
else
Add $\varphi_j(\psi_j)$ to $trans\_list$
Update sp_f and sp_g
Return Handle_SDS($f, g, sp\_f, sp\_g, trans\_list$)
end if
end for
Return 0
end if
end function

The meanings of conditions D_1, D_2, D_3, D_4, D_5 and D_6 and the operations that need to be performed when these conditions are satisfied are defined as follows:

D_1: When D_1 is true, a candidate transformation is generated. Procedure 2 checks whether the current NP transformation T can transform f into g ($\overline{g}$).

D_2: When D_2 is true, the transformation tree is NULL, and this is the first time that the SDS vectors have been computed. Procedure 2 checks and handles the independent-mapping set between f and g ($\overline{g}$).

D_3: When D_3 is true, the current variable x_i has already been identified, and Procedure 2 fetches the next x_i to handle.

D_4: When D_4 is true, the variable-mapping set of x_i is a single-mapping set or a single symmetry-mapping set.

D_5: When D_5 is true, there is a phase collision.

D_6: When D_6 is true, the cardinality of the minimal variable mapping set is 1.

In the process of transformation detection, Procedure 2 attempts each variable mapping in each multiple-mapping set or each group of variable mappings in each multiple symmetry-mapping set. For two NP-equivalent Boolean functions f and g ($\overline{g}$), Procedure 2 must find a candidate transformation that satisfies $f(TX) = g(X)$ ($\overline{g(X)}$). The purpose of VERIFY() is to check whether $f(TX) = g(X)$ ($\overline{g(X)}$).

UPDATE() serves the following functions:

(1) Updates the SDS vector V_f of f and the SDS vector V_g of g by means of Shannon decomposition and the decomposition expressions sp_f and sp_g.

(2) Updates the phases of the variables in f and g.

(3) Checks the variable symmetry when the SDS vectors are computed for the first time.

(4) Groups the variables of f and g by their 1st signature values and Boolean difference signatures.

In Example 2, if we use the SS values to search the variable mappings, then there are one single symmetry-mapping set $S_2 = \{C_2 \rightarrow C_0\}$ and two multiple symmetry-mapping sets $S_0 = \{C_0 \rightarrow C_1, C_0 \rightarrow C_2\}$ and $S_4 = \{C_4 \rightarrow C_1, C_4 \rightarrow C_2\}$, where $|S_0| = |S_4| = 4$. Procedure 2 handles the single symmetry-mapping set S_2, and the variable mappings $\overline{x}_2 \rightarrow x_0$ and $x_3 \rightarrow x_4$ are added to *trans_list*. sp_f and sp_g are updated to $\overline{x}_2$ and x_0. After the SS vectors are updated, the 1st signatures of the remaining variables of f and g are all $(8, 8)$. Therefore, the remaining four variables still cannot be distinguished, and there are two multiple symmetry-mapping sets, S_0 and S_4. The first symmetry-mapping set S_0 is selected to be handled. The transformation tree for Example 2 using SS vectors is shown in Figure 1.

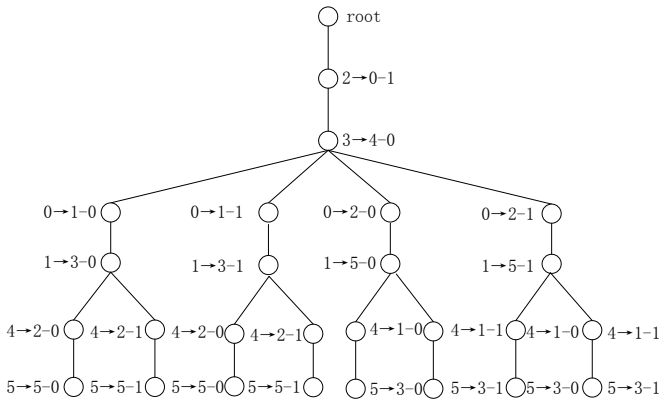

Figure 1. The transformation search tree for example 2 using SS vectors.

If we use the SDS values to search the variable mappings, then there are one independent-mapping set, one single symmetry-mapping set and one multiple symmetry-mapping set according to the first computed SDS vectors. Procedure 2 first adds the variable mappings $x_4 \rightarrow x_2$ and $x_5 \rightarrow x_5$ to the transformation tree. Then, the two variable mappings $\overline{x}_2 \rightarrow x_0$ and $x_3 \rightarrow x_4$ of the symmetry mapping $C_2 \rightarrow C_0$ are added to the transformation tree. The catch is that the independent variable is not a splitting variable because the decomposition results obtained via Shannon expansion with the independent variable are unchanged. Thus, the decomposition expressions sp_f and sp_g are updated

to $\overline{x}_2$ and x_0. UPDATE() is called to compute new SDS vectors, and the SDS vectors are updated in accordance with sp_f and sp_g.

In this way, Procedure 2 determines 4 variable mappings, namely, $x_4 \to x_2$, $x_5 \to x_5$, $\overline{x}_2 \to x_0$ and $x_3 \to x_4$, after the first variable mapping search for example 2. In the next variable mapping search, there is one multiple symmetry-mapping set, $S_0 = \{\{x_0 \to x_1, x_1 \to x_3\}, \{x_0 \to \overline{x}_1, x_1 \to \overline{x}_3\}\}$. The transformation tree for Example 2 using SDS vectors is shown in Figure 2.

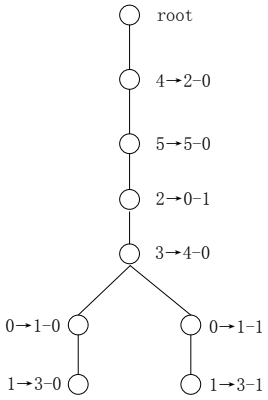

Figure 2. The transformation search tree for example 2 using SDS vectors.

From Example 2, we can see that the number of candidate transformations decreases from 8 to 2. The use of Boolean difference signatures helps to distinguish symmetry classes C_1 and C_5, and we need to consider only the positive phase for independent variables. Thus, Boolean difference signatures are very beneficial for distinguishing variables.

In cell library binding, a benchmark Boolean circuit is found to realize another NPN equivalent Boolean function. Example 3 demonstrates the process of NPN equivalent matching by SS and SDS vectors respectively, and illustrates the validity of the SDS vectors proposed in this paper.

Example 3. *Consider two 6-input Boolean functions $f(X)$ and $g(X)$:*

$f(X) = x_0x_1(x_3x_5 + x_4x_5) + x_0\overline{x}_1(x_4\overline{x}_5 + x_2x_3\overline{x}_4\overline{x}_5) + x_1\overline{x}_3\overline{x}_4\overline{x}_5 + x_1\overline{x}_2\overline{x}_4\overline{x}_5 + \overline{x}_0\overline{x}_1(\overline{x}_4x_5 + x_2\overline{x}_3x_4x_5) + \overline{x}_1\overline{x}_3\overline{x}_4x_5 + \overline{x}_0x_1(\overline{x}_2x_3x_4 + x_2\overline{x}_3x_4 + x_3x_4x_5 + x_2\overline{x}_4\overline{x}_5 + \overline{x}_2x_4\overline{x}_5) + \overline{x}_1x_2x_3x_4\overline{x}_5,$

$g(X) = \overline{x}_0x_1x_2\overline{x}_3 + x_0x_1\overline{x}_2\overline{x}_3 + \overline{x}_0\overline{x}_1\overline{x}_2x_3 + \overline{x}_0x_1x_2x_5 + \overline{x}_0x_1x_3x_4\overline{x}_5 + x_0\overline{x}_1x_2\overline{x}_5 + x_0x_1\overline{x}_2x_5 + \overline{x}_0\overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_0\overline{x}_1x_2\overline{x}_3x_5 + \overline{x}_0\overline{x}_1x_2x_4x_5 + \overline{x}_0\overline{x}_1\overline{x}_2x_4\overline{x}_5 + \overline{x}_0x_1\overline{x}_2x_3\overline{x}_4\overline{x}_5 + x_0x_1x_2x_3\overline{x}_4\overline{x}_5 + x_0\overline{x}_1\overline{x}_2\overline{x}_3x_4x_5.$

The transformation detection process using SS vectors is as follows:

(1) Compute the SS vectors of f and g. The results are:

$V_f = \{(16,16,-1,-1,1),(19,13,-1,-1,0),(16,16,-1,-1,1),(16,16,-1,-1,1),(16,16,-1,-1,1),(16,16,-1,-1,1)\},$

$V_g = \{(13,19,-1,-1,1),(16,16,-1,-1,0),(16,16,-1,-1,1),(16,16,-1,-1,1),(16,16,-1,-1,1),(16,16,-1,-1,1)\}.$

From the above results, we can draw three conclusions: (1) these two SS vectors are the same; (2) the phases of the variable x_1 of f and the variable x_0 of g are determined; and (3) there is only one variable x_0 of g with the same SS values as those of the variable x_1 of f. Therefore, there is a single-mapping set $\{x_1 \to \overline{x}_0\}$. Concerning the splitting variables, the new splitting expressions are $sp\_f = x_1$ and $sp\_g = \overline{x}_0$.

(2) The algorithm enters the next iteration and computes the new SS vectors. The new SS vectors are:

$V_f = \{(9,10,-1,-1,1),(0,0,-1,-1,0),(9,10,-1,-1,1),(10,9,-1,-1,1),(10,9,-1,-1,1),(9,10,-1,-1,1)\},$

$V_g = \{(0,0,-1,-1,1),(9,10,-1,-1,0),(10,9,-1,-1,1),(10,9,-1,-1,1),(10,9,-1,-1,1),(10,9,-1,-1,1)\}.$

The results show the following: (1) the two new SS vectors are the same; (2) the phases of all variables are determined; and (3) the next variable-mapping set to be handled is $\chi_0 = \{\overline{x}_0 \to \overline{x}_1, \overline{x}_0 \to x_2, \overline{x}_0 \to x_3, \overline{x}_0 \to x_4, \overline{x}_0 \to x_5\}$. Procedure 2 adds 5 nodes to the second layer of the transformation

tree. Procedure 2 handles the first variable mapping in the order of the variable mappings in the set and updates $sp\_f = x_1\overline{x}_0$ and $sp\_g = x_0\overline{x}_1$.

In the subsequent variable mapping search, the $\overline{x}_0 \to \overline{x}_1$ branch is pruned by a phase collision. The $\overline{x}_0 \to x_2$, $\overline{x}_0 \to x_3$ and $\overline{x}_0 \to x_4$ branches are pruned by having different SS vectors.

(3) Then, Procedure 2 handles the variable mapping $\overline{x}_0 \to x_5$ and detects a candidate transformation $T = \{x_1 \to \overline{x}_0, \overline{x}_0 \to x_5, x_4 \to \overline{x}_1, \overline{x}_5 \to x_2, \overline{x}_2 \to x_4, \overline{x}_3 \to \overline{x}_3\}$. After verification, this transformation is found to satisfy $f(TX) = g(X)$. Therefore, f is NPN-equivalent to g.

The transformation tree for Example 3 using SS vectors is shown in Figure 3.

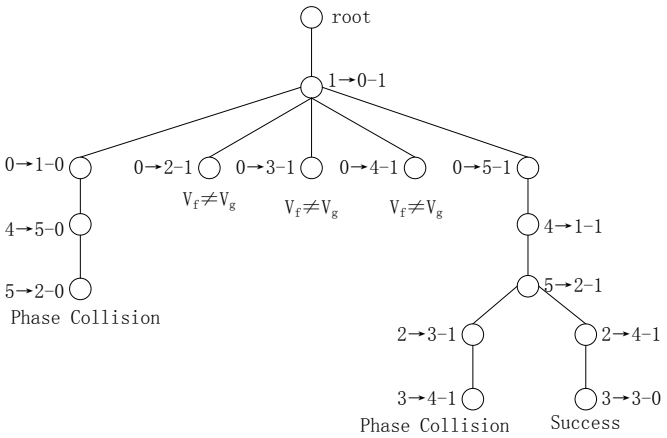

Figure 3. The transformation search tree for example 3 using SS vectorss.

Figure 3 shows that this transformation tree for Example 3 has 6 branches and that the two Boolean functions are decomposed 4 times. Let us examine the detection process using SDS vectors.

(1) The SDS vectors of f and g are as follows:

$V_f = \{(16, 16, -1, -1, 3, 28), (19, 13, -1, -1, 0, 64), (16, 16, -1, -1, 5, 12), (16, 16, -1, -1, 4, 20), (16, 16, -1, -1, 2, 36), (16, 16, -1, -1, 1, 52)\}$,

$V_g = \{(13, 19, -1, -1, 0, 64), (16, 16, -1, -1, 2, 36), (16, 16, -1, -1, 1, 52), (16, 16, -1, -1, 4, 20), (16, 16, -1, -1, 5, 12), (16, 16, -1, -1, 3, 28)\}$.

From these results, we can draw the following conclusions: (1) these two SDS vectors are the same; (2) the phases of the variable x_1 of f and the variable x_0 of g are determined,; and (3) there is one single-mapping set $\{x_1 \to \overline{x}_0\}$ to be used in the search. In Procedure 2, the splitting variables x_1 and $\overline{x}_0$ are used to decompose f and g, respectively.

(2) The new SDS vectors are as follows:

$V_f = \{(9, 10, -1, -1, 3, 14), (0, 0, -1, -1, 0, 64), (9, 10, -1, -1, 5, 6), (10, 9, -1, -1, 4, 10), (10, 9, -1, -1, 2, 18), (9, 10, -1, -1, 1, 26)\}$,

$V_g = \{(0, 0, -1, -1, 0, 64), (9, 10, -1, -1, 2, 18), (10, 9, -1, -1, 1, 26), (10, 9, -1, -1, 4, 10), (10, 9, -1, -1, 5, 6), (10, 9, -1, -1, 3, 14)\}$.

From these two new SDS vectors, the following can be seen: (1) the phases of all variables are determined; and (2) all unidentified variables can be identified from their Boolean differences. A candidate transformation $T = \{x_1 \to \overline{x}_0, \overline{x}_0 \to x_5, \overline{x}_2 \to x_4, x_3 \to x_3, x_4 \to \overline{x}_1, \overline{x}_5 \to x_2\}$ is generated, and this T is verified to be correct.

When SDS vectors are used to perform Boolean matching, the transformation tree for Example 3 contains only one candidate transformation. In the transformation detection process, the search space comprises all branches of the transformation tree, including unabridged and abridged branches. The unabridged branches are the candidate transformations, and the abridged branches are the pruned transformations. When the transformation tree possesses fewer branches, the algorithm considers a smaller search space. The purpose of decomposing the Boolean functions is to update the SDS vectors to search the new variable mappings. When the algorithm requires fewer decompositions,

more variables are identified in each iteration. These three indicators can be used to measure how much of the search space our algorithm searches.

In the best case, the variable mapping set of every variable is a single-mapping set, and there is only one candidate transformation. In this case, the spatial complexity is $O(1)$, and the time complexity is $O(n^2)$. In the worst case, there are no symmetric variables, every variable has the same SDS value, and the phases of all variables cannot be determined in each SDS update. There are $2^{n+1}n!$ candidate transformations that need to be verified. The spatial complexity is $O(2^n n!)$, and the time complexity is $O(n^3)$.

5. Experimental Results

To demonstrate the effectiveness of the proposed method, we re-implemented the algorithm of [6] and tested the algorithm presented in this paper, the algorithm of [5] and the algorithm of [6] on both a randomly generated circuit set and an MCNC benchmark circuit set. In the random circuit set, there were 1200 circuits in each input circuit set. Every circuit in the random circuit set contained at least two candidate transformations. In the test, we recorded the three indicators concerning the search space and the run time. The proposed algorithm was implemented in C with buddy package. The following experimental results were obtained in a hardware environment with a 3.3-GHz Intel Xeon processor and 4 GB of memory.

In the following tables, the first column shows the number of input variables (#I), and the following four columns show the experimental results for our algorithm. The next four columns show the corresponding experimental results of [5], and the last column shows the average run time of the algorithm of [6].

Tables 1 and 2 show the average number of branches (#B.N.), the average number of candidate transformations (#C.N.), the average number of decompositions (#D.N.) and the average run time (#R.T.) of our algorithm and of the algorithm of [5] on the random circuit set and on the MCNC benchmark circuit set, respectively.

Table 1. Boolean matching results on random circuits.

#I	#B.N.	#C.N.	#D.N.	#R.T.	#B.N. of [5]	#C.N. of [5]	#D.N. of [5]	#R.T. of [5]	#R.T. of [6]
7	2.2	2.1	3.2	0.00024	4.6	2.8	4.1	0.00024	0.00067
8	2.1	1.9	3.4	0.00022	9.2	3.7	4.9	0.00030	0.00091
9	3.7	2.2	3.6	0.00034	11.5	7.1	5.0	0.00047	0.00099
10	1.9	1.8	3.5	0.00027	15.0	14.1	5.3	0.00074	0.00123
11	7.8	7.7	3.3	0.00059	19.4	19.2	4.2	0.00053	0.00228
12	2.1	2.1	3.2	0.00032	6.7	6.6	3.8	0.00146	0.00585
13	3.3	3.2	3.2	0.00061	16.9	16.4	4.3	0.00127	0.00638
14	2.1	2.0	3.3	0.00063	8.7	7.3	4.6	0.00217	0.02743
15	1.8	1.7	3.2	0.00079	8.2	6.9	4.6	0.00262	0.02998
16	2.5	2.4	3.5	0.00155	10.1	8.6	4.5	0.00426	0.04310
17	2.9	2.8	3.7	0.00308	9.3	8.4	4.8	0.00784	0.05044
18	2.3	2.2	3.2	0.00445	7.0	6.1	4.6	0.02274	0.07177
19	2.4	2.3	3.1	0.00870	7.7	6.6	4.2	0.03285	0.08870
20	3.0	2.9	3.6	0.02069	7.5	6.4	4.9	0.04337	0.13250
21	2.1	2.1	3.2	0.02879	9.6	8.6	4.3	0.11471	0.17362
22	3.8	3.8	3.6	0.10301	8.7	7.7	5.1	0.20554	0.30469

From Table 1, we can see that the run time of our algorithm is improved by 54% relative to that of [5] and by 84% relative to that of [6]. From the comparison of the three indicators for the search space, we can see that the number of branches in the transformation tree is reduced by 70%, the number of candidate transformations is reduced by 65%, and the number of decompositions is reduced by 27%. Because the Boolean difference facilitates the identification of the variables, the proposed algorithm reduces the search space and speeds up the matching process. Figure 4 presents the diagram of the search space comparison results for our algorithm and that of reference [5] tested on the random circuit set.

Table 2. Boolean matching results on MCNC benchmark circuitsd.

#I	#B.N.	#C.N.	#D.N.	#R.T.	#B.N. of [5]	#C.N. of [5]	#D.N. of [5]	#R.T. of [5]	#R.T. of [6]
7	1.0	1.0	1.5	0.00014	1.3	1.0	1.8	0.00012	0.00121
8	1.1	1.1	3.2	0.00042	1.2	1.2	3.5	0.00035	0.00146
9	1.2	1.2	1.8	0.00034	1.6	1.5	2.4	0.00051	0.00186
10	1.1	1.1	1.3	0.00060	3.4	1.5	1.5	0.00063	0.00193
11	1.0	1.0	1.3	0.00074	1.4	1.4	2.6	0.00078	0.00243
12	1.0	1.0	1.3	0.00080	1.3	1.2	1.7	0.00091	0.00255
13	1.1	1.0	2.5	0.00467	1.3	1.1	3.8	0.00447	0.00535
14	1.1	1.1	1.4	0.00401	1.2	1.1	2.0	0.00368	0.01245
15	1.4	1.3	2.0	0.00569	1.7	1.5	3.1	0.00475	0.04077
16	1.2	1.2	1.9	0.01346	2.0	1.6	3.2	0.00151	0.04849
17	1.1	1.1	1.5	0.10502	1.2	1.2	2.2	0.11518	0.31644
18	1.1	1.0	1.5	0.10944	1.5	1.5	2.5	0.23394	0.64273
19	1.8	1.8	2.6	0.80747	5.5	5.3	4.4	0.89123	1.62971
20	1.3	1.3	2.1	1.00309	2.9	2.8	2.7	1.28156	2.13035
21	1.3	1.3	1.5	3.73072	1.7	1.5	2.7	3.71382	10.2122
22	1.6	1.6	1.7	6.50130	2.8	2.7	3.1	6.24368	11.2760

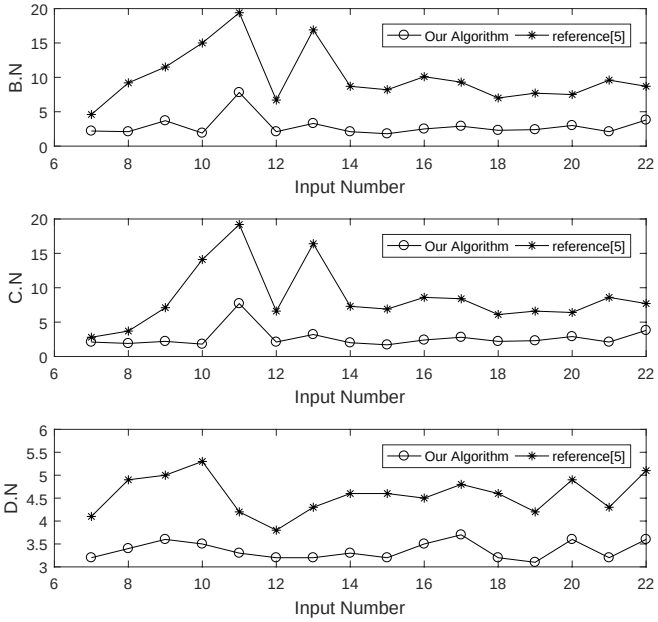

Figure 4. The search space comparison results for testing on random circuits.

Figure 5 presents the diagram of the speed comparison results for our algorithm, that of reference [6] and that of reference [5] tested on the random circuit set.

Table 2 shows the experimental results obtained during testing on the MCNC benchmark circuit set.

Table 2 shows that with the proposed algorithm, the values of the three indicators for the search space are decreased, and the run time is also slightly reduced. When there are 22 inputs, however, the average run time of our algorithm is higher than that of [5]. This is because the variables of this group circuit are easy to identify and because the search space of our algorithm is almost the same as that of [5]. In this case, our algorithm spends additional time in computing the Boolean differences compared with the algorithm of [5].

From Tables 1 and 2, we can see that the matching speed on the MCNC benchmark circuits is slower than that on random circuits, although the search space for the MCNC benchmark circuits is less than that for the random circuits. In this paper, we use BDDs to represent Boolean functions. The BDD structure of a Boolean function is closely related to the speed of operations on the BDD. Because the BDD operation speed on the MCNC benchmark circuits is slower than that on the random circuits, the matching speed on the MCNC benchmark circuits is also slower than that on the random circuits .

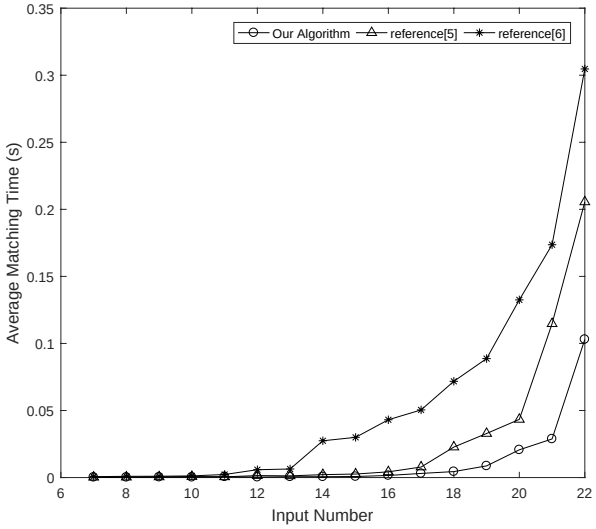

Figure 5. The matching speed comparison results for testing on random circuits.

6. Conclusions

The major contribution of this paper is the raise of SDS vector. The paper demonstrates how SDS vectors can be used to effectively search variable mappings and reduce the search space. The algorithm of this paper take advantage of cofactor, symmetry and Boolean different when search the variable mappings between two Boolean functions. Therefore, the search space and match speed of ours algorithm is better than the competitors. Compared with the algorithm of [5], the search space is cut in 48%, and the run time is reduced by 42% and 80% compared with [5,6], respectively. The experimental results prove that the algorithm proposed in this paper is more effective than competing algorithms on general circuits. In future work, we will extend our algorithm to multiple-output Boolean matching and Boolean matching with don't care sets.

Author Contributions: J.Z. proposed and implemented the algorithm. G.Y. was the research advisor and provided suggestions. W.N.N.H. provided guidance for this paper and contributed to the revisions and gave advice on optimization issues. J.W. and Y.Z. completed the generation of data and test.

Funding: This research was funded by the National Natural Science Foundation of China Grant (Nos. 61572109, 11371003 and 61751110), the Special Fund for Bagui Scholars of Guangxi (Grant No. 113000200230010), Network and Data Security Key Laboratory of Sichuan Province Open Project (NDSMS201603).

Acknowledgments: We would like to thank the above funds for their technical and financial support.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Slepian, D. On the number of symmetry types of Boolean functions of n variables. *Can. J. Math.* **1955**, *2*, 185–193. [CrossRef]
2. Zhang, J.; Yang, G.; Hung, W.N.N.; Liu, T.; Song, X.; Perkowski, M.A. A group algebraic approach to NPN classification of Boolean functions. *Theory Comput. Syst.* **2018**. [CrossRef]
3. Zhang, Y.; Yang, G.; Hung, W.N.N.; Zhang, J. Computing affine equivalence classes of Boolean functions by group isomorphism. *IEEE Trans. Comput.* **2016**, *12*, 3606–3616. [CrossRef]
4. Lai, Y.-T.; Sastry, S.; Pedram, M. Boolean matching using binary decision diagrams with applications to logic synthesis and verification. In Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors, Cambridge, MA, USA, 11–14 October 1992. [CrossRef]
5. Zhang, J.; Yang, G.; Hung, W.N.N.; Zhang, Y.; Wu, J. An efficient NPN Boolean matching algorithm based on structural signature and Shannon expansion. *Cluster Comput.* **2018**, *6*, 1–16. [CrossRef]
6. Adbollahi, A.; Pedram, M. Symmetry detection and Boolean matching utilizing a signature-based canonical form of Boolean functions. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2008**, *6*, 1128–1137. [CrossRef]

7. Lee, S.Y.; Lee, N.Z.; Jiang, J.H.R. Canonicalization of threshold logic representation and its applications. In Proceedings of the International Conference on Computer-Aided Design, San Diego, CA, USA, 5–8 November 2018; p. 85.
8. Asghar, A.; Iqbal, M.M.; Ahmed, W.; Ali, M.; Parvez, H.; Rashid, M. Logic algebra for exploiting shared SRAM-table based FPGAs for large LUT inputs. In Proceedings of the Electrical Engineering and Computing Technologies, Karachi, Pakistan, 15–16 November 2017; pp. 1–4. [CrossRef]
9. Soeken, M.; Mishchenko, A.; Petkovska, A.; Sterin, B.; Ienne, P.; Brayton, R.K.; De Micheli, G. Heuristic NPN classification for large functions using AIGs and LEXSAT. In Proceedings of the International Conference on Theory and Applications of Satisfiability Testing, Bordeaux, France, 5–8 July 2016; pp. 212–227.
10. Huang, Z.; Wang, L.; Nasikovskiy, Y.; Mishchenko, A. Fast Boolean matching based on NPN classification. In Proceedings of the International Conference on Field-Programmable Technology, Kyoto, Japan, 9–11 December 2013; pp. 310–313. [CrossRef]
11. Petkovska, A.; Soeken, M.; Micheli, G.D.; Ienne, P.; Mishchenko, A. Fast hierarchical NPN classification. In Proceedings of the International Conference on Field Programmable Logic and Applications, Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–4. [CrossRef]
12. Agosta, G.; Bruschi, F.; Pelosi, G.; Sciuto, D. A transform-parametric approach to Boolean matching. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2009**, *6*, 805–817. [CrossRef]
13. Chen, K.C.; Yang, C.Y. Boolean matching algorithms. In Proceedings of the International Symposium on VLSI Technology, Systems, and Applications, Taipei, Taiwan, 12–14 May 1993; pp. 44–48.
14. Kapoor, B. Improved technology mapping using a new approach to Boolean matching. In Proceedings of the European Design and Test Conference, Paris, France, 6–9 March 1995; pp. 86–90. [CrossRef]
15. Vos, A.D.; Baerdemacker, S.D. Symmetry groups for the decomposition of reversible computers, quantum computers, and computers in between. *Symmetry* **2011**, *2*, 305–324. [CrossRef]
16. Katebi, H.; Igor, I.L. Large-scale Boolean matching. In Proceedings of the Conference on Design, Automation and Test in Europe, Dresden, Germany, 8–12 March 2010; pp. 771–776. [CrossRef]
17. Matsunaga, Y. Accelerating SAT-based Boolean matching for heterogeneous FPGAs using one-hot encoding and CEGAR technique. In Proceedings of the Design Automation Conference of 20th Asia and South Pacifics, Chiba, Japan, 19–22 January 2015; pp. 255–260. [CrossRef]
18. Ghaderi, Z.; Bagherzadeh, N.; Albaqsami, A. STABLE: Stress-Aware Boolean Matching to Mitigate BTI-Induced SNM Reduction in SRAM-Based FPGAs. *IEEE Trans. Comput.* **2018**, *99*, 1. [CrossRef]
19. Cong, J.; Minkovich, K. Improved SAT-based Boolean matching using implicants for LUT-based FPGAs. In Proceedings of the ACM/sigda International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 18–20 February 2007; pp. 139–147.
20. Wang, K.H.; Chan, C.M.; Liu, J.C. Simulation and SAT-based Boolean matching for large Boolean networks. In Proceedings of the Design Automation Conference, San Francisco, CA, USA, 26–31 July 2009; pp. 396–401.
21. Moore, J.; Fazel, K.; Thornton, M.A.; Miller, D.M. Boolean function matching using Walsh Spectral decision diagrams. In Proceedings of the Design, Applications, Integration and Software, Richardson, TX, USA, 29–30 October 2006; pp. 127–130. [CrossRef]
22. Thornton, M.A.; Drechsler, R.; Gunther, W. Logic circuit equivalence checking using Haar Spectral coefficients and partial BDDs. *VLSI Des.* **2014**, *1*, 53–64. [CrossRef]
23. Zhang, J.S.; Chrzanowska-Jeske, M.; Mishchenko, A.; Burch, J.R. Linear cofactor relationships in Boolean functions. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2006**, *6*, 1011–1023. [CrossRef]