Exploiting K-Distance Signature for Boolean Matching and $\mathcal{G}$-Symmetry Detection *

Kuo-Hua Wang
Dept. of Computer Science and Information Engineering
Fu Jen Catholic University
Hsinchuang City, Taipei County 24205, Taiwan, R.O.C.
khwang@csie.fju.edu.tw

ABSTRACT

In this paper, we present the concept of *k-distance signature* which is a general case of many existing signatures. By exploiting k-distance signature, we propose an *Algorithm for Input Differentiation* (*AID*) which is very powerful to distinguish inputs of Boolean functions. Moreover, based on *AID*, we propose a heuristic method to detect $\mathcal{G}$-symmetry of Boolean functions and a Boolean matching algorithm. The experimental results show that our methods are not only effective but also very efficient for Boolean matching and $\mathcal{G}$-symmetry detection.

Categories and Subject Descriptors: B.6.3 [**Hardware**] Logic Design: Design Aids - Automatic Synthesis.

General Terms: Algorithms, Design, Verification.

1. INTRODUCTION

Boolean matching is a technique to check if two functions are equivalent under input permutation and input/output phase assignments (so called *NPN-Class*). It can be applied in technology mapping and logic verification. In the early 90's, the work in the article [1] opened the door of research on Boolean matching for technology mapping. Over the past decade, many studies had been devoted to Boolean matching and various Boolean matching methods were proposed [1]-[6]. Among these methods, exploiting *signatures* and computing *canonical forms* of Boolean functions are the most successful approaches to deal with Boolean matching.

The idea behind using signatures for Boolean matching is very simple. Various signatures can be defined to characterize the input variables of Boolean functions, where the variables with different signatures can be distinguished with each other and many infeasible permutations and phase assignments can be pruned quickly. The major issue of signature based approach is the *aliasing* problem, i.e. two or more input variables have the same signature and therefore can't be distinguished. In such a case, an exhaustive search is still required to find the input mapping. The paper [8] had shown signatures have the inherent limitations to distinguish all input variables for those Boolean functions with $\mathcal{G}$-symmetry. To the best of our knowledge, there were few research devoted to detect $\mathcal{G}$-symmetry of Boolean functions [9][10].

Yet, another group of research was based on computing canonical forms of Boolean functions [5][6]. Using this approach, each *NPN-class* of functions will be represented as a unique canonical function. If two functions can be transformed to the same canonical function, then they are matched. In a recent paper [6], a new approach combining signature and canonical form of Boolean functions was proposed. Their method was based on *generalized signatures* (signatures of one or more variables) and canonicity-producing (CP) transformation for Boolean matching under input permutation and input phase assignment.

In this paper, we consider permutation independent Boolean matching problem. A new k-distance signature will be presented and correlated with many previously proposed signatures. Based on k-distance signature, we will propose a very effective *Algorithm for Input Differentiation* (*AID*) to distinguish inputs of Boolean functions. With respect to (w.r.t.) the aliasing problem of signatures, we also present a heuristic method to detect $\mathcal{G}$-symmetry of Boolean functions. Combining *AID*, $\mathcal{G}$-symmetry detection, and transformation of Ordered Binary Decision Diagrams (OBDD's), we also propose a Boolean matching algorithm to match two Boolean functions.

The remainder of this paper is organized as follows. Section 2 briefly reviews Boolean matching and some previously proposed signatures. Then we introduce k-distance signature and propose a heuristic to detect $\mathcal{G}$-symmetry of Boolean functions in Section 3 and Section 4, respectively. Our *AID* and Boolean matching algorithms will be described in Section 5. Finally, the experimental results and conclusion will be given in Section 6 and Section 7, respectively.

2. PRELIMINARIES

2.1 Boolean Matching and Symmetry

Boolean matching is to check the equivalence of two functions under input permutation and input/output phase assignment. Let B^n denote the multi-dimensional space spanned by n binary-valued Boolean variables. A *minterm* is a point in the Boolean space B^n. A *literal* is a variable x_i (*positive literal*) or its complement form $\bar{x_i}$ (*negative literal*). The *weight* of a minterm m, denoted as $W(m)$,

*This work was supported by the National Science Council of Taiwan under Grants NSC 94-2215-E-030-006.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

is the number of positive literals x_i's involved in m. The *cofactor* of $f(x_1, \cdots, x_i, \cdots, x_n)$ w.r.t. the literal x_i ($\bar{x_i}$) is $f_{x_i} = f(x_1, \cdots, 1, \cdots, x_n)$ ($f_{\bar{x_i}} = f(x_1, \cdots, 0, \cdots, x_n)$). The *satisfy count* of f, denoted as $|f|$, is the number of minterms for which the function value is 1.

Given two inputs x_i and x_j of a function f, (x_i, x_j) is a *symmetric pair* of f if and only if f is invariant under the swapping of x_i and x_j, i.e. $f_{\bar{x_i}x_j} = f_{x_i\bar{x_j}}$. Let X_s be an input subset of a Boolean function $f(X)$. X_s is a *maximal symmetric set* of f if and only if f is not symmetric w.r.t. $X_s \cup \{x_i\}$ for any input $x_i \in X - X_s$ [8]. $f(X)$ is a *totally symmetric function* if and only if X is a maximal symmetric set of f. In other words, f evaluates to 1 w.r.t. all the minterms with weight $i \in A$, where $A \subseteq \{0, 1, \cdots, n\}$ and n is the number of input variables. It is denoted as S_A^n and can be shortened as S_k^n if $A = \{k\}$.

2.2 Review of Some Signatures

Consider a Boolean function $f(X)$ and an input variable $x_i \in X$. The *signature* of f w.r.t. x_i is a description of the input x_i that is independent of the permutation of the inputs of Boolean function f. Four types of previously proposed signatures are redefined as below.

DEFINITION 2.1. The *cofactor-satisfy-count signature* of f w.r.t. x_i is the integer $|f_{x_i}|$. ■

DEFINITION 2.2. The *partner signature* [3] of f w.r.t. x_i is an integer vector $[p_0, p_1, p_2, p_3]$, where p_0, p_1, p_2, and p_3 are $|\bar{f}_{\bar{x_i}} \cdot \bar{f}_{x_i}|$, $|\bar{f}_{\bar{x_i}} \cdot f_{x_i}|$, $|f_{\bar{x_i}} \cdot \bar{f}_{x_i}|$, and $|f_{\bar{x_i}} \cdot f_{x_i}|$, respectively. ■

DEFINITION 2.3. The *cofactor-breakup signature* [4] of f w.r.t. x_i is an integer vector $[b_1, \cdots, b_i, \cdots, b_n]$, where b_i is the number of minterms with weight i in the domain $x_i = 1$, i.e. $|f_{x_i}| = \sum_{i=1}^{n} b_i$. ■

DEFINITION 2.4. The *cross signature* [7] of f w.r.t. x_i is an integer vector $[|f_{x_i}|, |f_{x_i\bar{x_1}}|, \cdots, |f_{x_i\bar{x_n}}|]$. ■

DEFINITION 2.5. Let s_1 and s_2 be two different signature types. We say that s_1 *dominates* s_2 if two inputs $x_i, x_j \in X$ of any Boolean function $f(X)$ can be distinguished by s_2, then they can be distinguished by s_1 as well. ■

THEOREM 2.1. The *partner* and *cofactor-breakup signatures* dominate the *cofactor-satisfy-count signature*.
< **Proof** :> This theorem can be proved using the fact that the partner and cofactor-breakup signatures are integer decompositions of $|f_{x_i}|$. The details are however omitted because of space limitation. ■

THEOREM 2.2. The *cross signature* dominates the *cofactor-satisfy-count signature*.
< **Proof** :> It can be easily proved because the first element of cross signature is equal to the cofactor-satisfy-count signature. ■

2.3 $\mathcal{G}$-Symmetry

Let $\mathcal{G} \subseteq \mathcal{P}_n$, where $\mathcal{P}_n$ involves all input permutations w.r.t. an input variable set X. A Boolean function $f(X)$ is *$\mathcal{G}$-symmetric* if and only if f is invariant under any input permutation $\psi \in \mathcal{G}$. There are three types of $\mathcal{G}$-symmetry discussed in the paper [8]: *group symmetry* (*g-symmetry*), *hierarchical symmetry* (*h-symmetry*), and *rotational symmetry* (*r-symmetry*).

DEFINITION 2.6. Consider a Boolean function $f(X)$. Let $X_i = [x_{i_1}, x_{i_2}, \cdots, x_{i_k}]$ and $X_j = [x_{j_1}, x_{j_2}, \cdots, x_{j_k}]$ be two disjoint ordered subsets of X with $k \geq 1$. f is *group symmetric* (*g-symmetric*) w.r.t. to X_i and X_j if and only if f is invariant under the swapping of x_{i_m} and x_{j_m} for $m = 1, 2, \cdots, k$. We call (X_i, X_j) is a *g-symmetric pair* of f. ■

EXAMPLE 2.1. Let $f = x_1(x_2 + \bar{x_3}) + x_4(x_5 + \bar{x_6})$. It is obvious that f is g-symmetric w.r.t. ordered sets $[x_1, x_2, x_3]$ and $[x_4, x_5, x_6]$. ■

By Def. 2.6, the traditional symmetric pair can be viewed as a special case of g-symmetry pair with $k = 1$. It is easy to prove that g-symmetric pair is an equivalence relation. That is, a larger g-symmetric sets can be derived from g-symmetric pairs directly. In addition, h-symmetry is a special case of g-symmetry, i.e. all swapping groups are maximal symmetric sets.

DEFINITION 2.7. Let X_i and X_j be two maximal symmetric sets of $f(X)$. f is *h-symmetric* w.r.t. X_i and X_j if and only if f is g-symmetric w.r.t. X_i and X_j. ■

EXAMPLE 2.2. Consider the function $f = x_1x_2x_3 + x_4x_5x_6 + x_7x_8x_9$. There are three maximal symmetric sets $X_1 = \{x_1, x_2, x_3\}$, $X_2 = \{x_4, x_5, x_6\}$, and $X_3 = \{x_7, x_8, x_9\}$. It is clear that f is h-symmetric w.r.t. the permutation of X_1, X_2, and X_3. ■

DEFINITION 2.8. Consider a function $f(X)$ and an ordered subset X_i of X. Without loose of generality, let $X_i = [x_1, x_2, \cdots, x_k]$ which is not a symmetric set of f for $k \geq 3$. f is *rotational symmetric* (*r-symmetric*) w.r.t. X_i if and only if f is invariant under the permutations $\psi = (x_m, x_{m+1}, \cdots, x_k, x_1, \cdots, x_{m-1})$ for $m = 1, 2, \cdots, k$. ■

EXAMPLE 2.3. Let $f = x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_1$. f is invariant under rotating the variables of f w.r.t. the ordered set $[x_2, x_3, x_4, x_5, x_1]$. ■

3. K-DISTANCE SIGNATURES

3.1 Five types of K-Distance Signature

Let $p = (m_a, m_b)$ be a pair of minterms w.r.t. the same input set. The *mark* of p, denoted as $Mark(p)$, is the product by anding the literals involved in m_a which are different from the literals involved in m_b. For example, consider the minterm pair $p = (m_a, m_b)$ w.r.t. the input set $X = \{x_1, x_2, x_3, x_4, x_5\}$, where $m_a = 01100$ and $m_b = 10000$. Then $Mark(p) = \bar{x_1}x_2x_3$.

Given two functions f and g w.r.t. the same input set, consider any minterm pair $p = (m_a, m_b)$ with mark M_i, where m_a and m_b are from f and g, respectively. Assume the number of literals in the mark M_i is k, denoted as $|M_i|$. Let $S_{M_i}^k$ and $|S_{M_i}^k| = w_i$ denote the set involving all minterm pairs with mark M_i and the number of minterm pairs in $S_{M_i}^k$, respectively. Then the tuple (w_i, M_i) can characterize a relation between f and g. If we further classify (w_i, M_i)'s in terms of the number of literals involved in M_i, then each group can be viewed as a signature showing the *k-disjoint* situations between f and g. In the following, we define the k-distance signature and give a simple example.

DEFINITION 3.1. Consider two functions $f(X)$ and $g(X)$. Given an integer $k > 0$, the *k-distance signature* of f w.r.t. to g, denoted as $ksig(f, g, k)$, can be formally defined as

$$ksig(f, g, k) = \{(w_i, M_i) || S_{M_i}^k | = w_i, |M_i| = k\} \quad (1)$$

, where $S_{M_i}^k = \{p = (m_a, m_b) | Mark(p) = M_i, \forall m_a \in f, \forall m_b \in g\}$. ■

EXAMPLE 3.1. Consider two functions $f = x_1\bar{x_2}x_3 = \sum m(5)$ and $g = (\bar{x_1} + x_2)x_3 = \sum m(1,3,7)$. There are two minterm pairs (m_5, m_1) and (m_5, m_7) with mark x_1 and $\bar{x_2}$, respectively. So $ksig(f,g,1) = \{(1,x_1),(1,\bar{x_2})\}$. For $k=2$, $ksig(f,g,k=2) = \{(1,x_1\bar{x_2})\}$ because of one minterm pair (m_5, m_3) with mark $x_1\bar{x_2}$. ■

Instead of computing Equation (1) directly, given an input x_i , $ksig(f,g,k)$ can be computed in a recursive way as below.

$$ksig(f,g,k) = ksig(f_{\bar{x_i}}, g_{\bar{x_i}}, k) \bigcup ksig(f_{x_i}, g_{x_i}, k) \\ \bigcup \bar{x_i} \cdot ksig(f_{\bar{x_i}}, g_{x_i}, k-1) \bigcup x_i \cdot\ ksig(f_{x_i}, g_{\bar{x_i}}, k-1) \quad (2)$$

By Def. 3.1, $ksig(f,g,k)$ can be viewed as a $2^k \times C_k^n$ integer matrix M with rows and columns corresponding to different input phase assignments and input combinations of k inputs from X, respectively.

EXAMPLE 3.2. Consider $f(x_1,x_2,x_3,x_4) = \sum(0,1,3,6,8,13,14,15)$. $ksig(f,f,k=1)$ is a $2 \times C_1^4$ integer matrix shown below:

$$\begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \left[\begin{array}{cccc} 2 & 0 & 2 & 2 \\ 2 & 0 & 2 & 2 \end{array}\right] \end{array}$$

W.r.t. $k=2$, it is a $2^2 \times C_2^4$ integer matrix shown below:

$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1x_2 & x_1x_3 & x_1x_4 & x_2x_3 & x_2x_4 & x_3x_4 \end{array} \\ \left[\begin{array}{cccccc} 2 & 0 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 2 & 1 & 1 \end{array}\right] \end{array}$$ ■

The following definition shows how to extract the signature from $ksig(f,g,k)$ w.r.t. an input $x_i \in X$.

DEFINITION 3.2. Consider a *target function* $f(X)$ and an *anchor function* $g(X)$. Let x_i be an input of X. Given an integer $k > 0$, the k-distance signature of f w.r.t. x_i and g is an integer vector $[v_1, \cdots, v_j, \cdots, v_s]$ with $s = 2^{k-1} \times C_{k-1}^{n-1}$, where $v_j = |f_{x_i m_a}|$ and $m_a \in B^{k-1}$ which is expanded by any $(k-1)$-input subset of X without involving x_i. ■

EXAMPLE 3.3. Consider the same function $f(X)$ shown in Example 3.2 and an input $x_1 \in X$. Given $k=2$, the k-distance signature of f w.r.t. x_1 and anchor function f is a six-integer vector shown below:

$[|f_{x_1\bar{x_2}}|, f_{x_1\bar{x_3}}|, f_{x_1\bar{x_4}}|, |f_{x_1x_2}|, f_{x_1x_3}|, f_{x_1x_4}|] = [0,0,0,2,0,1]$ ■

According to Equation (2), the time complexity of computing k-distance signature is $O(p^{k+1} \times q^{k+1})$, where p and q are the sizes of OBDD's representing f and g, respectively. It is time consuming to compute k-distance signature w.r.t. a large number k. Therefore, we propose five k-distance signatures w.r.t. $k=1$ and use $ksig(f,g)$ to denote $ksig(f,g,k=1)$ for short. It is clear that $ksig(f,g)$ is a $2n$-integer vector, where each tuple corresponds to the number of pairs of minterms which only involve one different literal x_i or $\bar{x_i}$.

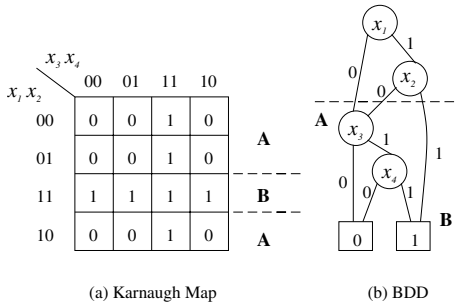
- **Type 0:** $g = 1$

  In fact, type-0 k-distance signature is equivalent to the cofactor satisfy-count signature. However, it can compute the cofactor-satisfy-count signatures w.r.t. all inputs simultaneously.

- **Type 1:** $g_1 = f$ and $g_2 = \bar{f}$
- **Type 2:** $g = S_i^n$ for $i = 0, 1, \cdots, n$

  S_i^n is a totally symmetric function involving all minterms with weight i.

- **Type 3:** $g = x_i$ for $i = 1, \cdots, n$



(a) Karnaugh Map (b) BDD

Figure 1: Karnaugh map and BDD of $f = x_1x_2 + x_3x_4$.

Below, we will describe a new type of k-distance signature based on communication complexity. Suppose that X_b is the set of inputs (or referred as *bounded set*) which had been distinguished by some signatures. The *communication complexity* of f w.r.t. X_b is defined as the number of distinct cofactors of f w.r.t. m_i, where m_i is a minterm in the Boolean space spanned by X_b. For example, consider $f = x_1x_2 + x_3x_4$ and bound set $X_b = \{x_1, x_2\}$. Fig. 1(a) and 1(b) show the Karnaugh map of f w.r.t. the partition $X_b/X - X_b$ and OBDD of f with order $x_1 < x_2 < x_3 < x_4$, respectively. W.r.t. the partition $X_b/X - X_b$, there are two distinct cofactors **A** and **B** corresponding to row patterns 0010 and 1111, respectively. So the communication complexity is 2. Take a look at this OBDD. Under the dotted cut-line, it is easy to identify two sub-BDD's which correspond to cofactors **A** and **B**, respectively. W.r.t. such an input partition, f can be decomposed as $f = b_1 \cdot u_1 + b_2 \cdot u_2$, where $b_1 = \bar{x_1} + \bar{x_2}$, $b_2 = x_1x_2$, $u_1 = x_3x_4$ and $u_2 = 1$.

Given a Boolean function $f(X)$ and a bounded set X_b including all distinguished inputs, let m be the communication complexity of f w.r.t. X_b. Then f can be decomposed as $f = \sum_{i=1}^{m} b_i(X_b) \cdot u_i(X - X_b)$, where $f_{m_j} = u_i$ for each minterm $m_j \in b_i$. Type 4 k-distance signature is thus defined as below.

- **Type 4:** $g = b_i$ for $i = 1, \cdots, m$

THEOREM 3.1. Type 0 k-distance signature is equivalent to the cofactor-satisfy-count signature .

< **Proof** :> As we state for k-distance signature w.r.t. $k=1$, it is an integer vector of $2n$ tuples corresponding to literals x_i's or $\bar{x_i}$'s. Consider any minterm $m_a \in f$. Let $m_a = \prod_{i=1}^{n} l_i$, where l_i is x_i or $\bar{x_i}$. For each literal $l_i \in m_a$, since g is a tautology, it is clear that there exists a minterm $m_b \in g$, where $Mark(m_a, m_b) = l_i$. So the tuple on l_i in type 0 k-distance signature will increase accordingly. After checking all minterms in f in such a way, then each tuple is equivalent to the cofactor-satisfy-count signature w.r.t. literal x_i or $\bar{x_i}$. It is proved. ■

THEOREM 3.2. Type 1 k-distance signature is equivalent to the partner signature.

< **Proof** :> By Def. 2.2, the partner signature is a 4-tuple integer vector $[p_0, p_1, p_2, p_3]$, where p_0, p_1, p_2 and p_3 are $|\bar{f}_{\bar{x_i}} \cdot \bar{f}_{x_i}|$, $|\bar{f}_{\bar{x_i}} \cdot f_{x_i}|$, $|f_{\bar{x_i}} \cdot \bar{f}_{x_i}|$, and $|f_{\bar{x_i}} \cdot f_{x_i}|$, respectively. Consider any input x_i in X. W.r.t. $g = f$, The tuple of $ksig(f,f)$ corresponding to x_i is equal to p_3. The same observation can be applied to $g = \bar{f}$. The tuples in $ksig(f,\bar{f})$ corresponding to x_i and $\bar{x_i}$ are equal to p_1 and p_2, respectively. Moreover, p_0 can be computed as $2^{n-1} - (p_1 + p_2 + p_3)$. It is proved. ■

THEOREM 3.3. Type 2 k-distance signature is equivalent to the cofactor-breakup signature.

< **Proof** :> By Def. 2.3, the cofactor-breakup signature of f w.r.t. x_i shows the distribution of minterms involving the literal x_i in terms of minterm weight. It is an integer vector $[b_1, \cdots, b_j, \cdots, b_n]$, where b_j is the number of minterms with weight j. Consider $g = S^n_{j-1}$. Consider the domain $x_i = 1$. For each minterm $m_a \in f$ with weight j, there must exist a minterm $m_b \in g$ and $Mark(m_a, m_b) = x_i$. So b_j can increase 1 because of this minterm pair. W.r.t. different S^n_j's, all b_j's can be computed in the same way. It is proved. ■

THEOREM 3.4. Type 3 k-distance signature is equivalent to the cross signature.

< **Proof** :> It can be proved directly by their definitions.■

DEFINITION 3.3. Consider a Boolean function $f(X)$ and a signature type s. The *distinct factor* of f w.r.t. s, denoted as $df(f, s)$, is defined as the ratio of the number of inputs distinguished by s to the number of inputs in X. ■

Let us consider a Boolean function $f(X)$ with symmetric inputs first. Without loose of generality, let $MSS = \{x_1, x_2, \cdots, x_m\}$ be a maximal symmetric set of f. In our *AID* algorithm, x_1 will be used to represent MSS. While computing the distinct factor of f w.r.t. any signature type, the remaining $m-1$ inputs can be viewed as the distinguished inputs of f.

EXAMPLE 3.4. Let $f = x_1x_2 + x_3x_4 + x_5$. Type 0 *k-distance signature* s_0 of f is shown below:

	x_1	x_2	x_3	x_4	x_5
0	10	10	10	10	7
1	13	13	13	13	16

It is clear that $\{x_1, x_2\}$ and $\{x_3, x_4\}$ are symmetric sets of f. Consequently, x_2 and x_4 are included in the distinguished input set DI. x_5 is also added into DI because of $|f_{x_5}| = 16 \neq 13$. So $df(f, s_0) = 3/5 = 0.6$. ■

4. DETECTION OF $\mathcal{G}$-SYMMETRY

4.1 Check of Group Symmetry

There are two phases in our heuristic to detect g-symmetry of Boolean functions.

Phase 1: Search of Candidate Swapping Pairs

Suppose f is g-symmetric to $X_1 = [x_{1_1}, x_{1_2}, \cdots, x_{1_k}]$ and $X_2 = [x_{2_1}, x_{2_2}, \cdots, x_{2_k}]$. Let $A = [a_1, a_2, \cdots, a_i, \cdots, a_k]$ and $B = [b_1, b_2, \cdots, b_i, \cdots, b_k]$ be constructed by the following rule: if $a_i = x_{1_i}$ (or x_{2_i}), then $b_i = x_{2_i}$ (or x_{1_i}). It is easy to prove that f is also g-symmetric w.r.t. A and B. Therefore, it doesn't matter to find any specific ordered sets A and B. We only need to search the candidate pairs of variables for swapping. In our method, we consider the swapping of two indistinguishable inputs x_i and x_j and use type 4 k-distance signature to find their mutual dependency which can help search the other candidate swapping pairs. Consider the function $f = x_1(x_2 + \bar{x_3}) + x_4(x_5 + \bar{x_6})$. It has three indistinguishable input groups $X_1 = \{x_1, x_4\}$, $X_2 = \{x_2, x_5\}$, and $X_3 = \{x_3, x_6\}$. The following matrix shows the positive phase of type 4 signatures w.r.t. all input variables. Each entry corresponds to the cofactor count of $f_{x_ix_j}$.

		X_1		X_2		X_3	
		x_1	x_4	x_2	x_5	x_3	x_6
X_1 :	x_1	0	**14**	**16**	14	**10**	13
	x_4	**14**	0	14	**16**	13	**10**
X_2 :	x_2	**16**	14	0	**12**	**11**	10
	x_5	14	**16**	**12**	0	10	**11**
X_3 :	x_3	**10**	13	**11**	13	0	**7**
	x_6	13	**10**	10	**11**	**7**	10

	x_1	x_2	x_3	x_4	x_5
x_1	0	8	7	7	8
x_2	8	0	8	7	7
x_3	7	8	0	8	7
x_4	7	7	8	0	8
x_5	8	7	7	8	0

(a) M_1

	x_1	x_2	x_4	x_5	x_3
x_1	0	8	7	8	7
x_2	8	0	7	7	8
x_4	7	7	0	8	8
x_5	8	7	8	0	7
x_3	7	8	8	7	0

(b) M_2

Figure 2: Type 4 Signatures with Different Input Orders.

Consider the input group X_1 w.r.t. the above matrix. If we swap the inputs x_1 and x_4, two pairs (x_2, x_5) and (x_3, x_6) must be also swapped to make this matrix invariant. The same situation can happen to swapping the inputs in X_2 and X_3. With such an observation, we can obtain three swapping candidate pairs. Then f can be checked if it is invariant under the swapping of these candidate pairs. Our heuristic chooses the inputs from the smallest group as the initial swapping candidate pair.

Phase 2: Check of Group Symmetry with Transpositional Operator

Consider a function $f(X)$ and two k-input ordered subsets X_1 and X_2 of X. In the article [9], the authors proposed a straightforward method using $2^{2k-1} - 2^{k-1}$ cofactors' equivalence checking which is not feasible for large number k. Instead, we check $\mathcal{G}$-symmetry of f w.r.t. X_1 and X_2 by *transpositional operator* which can swap two inputs of f with ITE operator [11]. The time complexity of transpositional operator is $O(p^2)$, where p is the size of OBDD representing f. Let f^k be the function obtained from f by applying k transpositional operators to f. If $f^k = f$, then f is $\mathcal{G}$-symmetric w.r.t. X_1 and X_2. Let $f^0 = f$ and f^i be the function after applying the i'th transpositional operator. The time complexity of computing f^k from f is $O(\sum_{i=0}^{k-1} {p_i}^2)$, where p_i is the size of OBDD representing f^i.

4.2 Check of Rotational Symmetry

Consider the function $f = x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_1$ which is invariant under rotating the variables of f with the ordered set $[x_5, x_1, x_2, x_3, x_4]$. Fig. 2(a) and (b) show the matrices of type 4 k-distance signature of f with input orders $[x_1, x_2, x_3, x_4, x_5]$ and $[x_1, x_2, x_4, x_5, x_3]$, respectively. In the first matrix, each left-to-right diagonal has the same value, while the second matrix doesn't own this property. According to this important observation, given an initial type 4 k-distance signature matrix, we have developed a prune-and-search algorithm to find all reordered matrices with all diagonals having the same value. If there exists such a matrix, we can further check if f is r-symmetric w.r.t. this input order. The detail is however omitted because of space limitation.

We have implemented the matrix reordering algorithm mentioned above and performed an experiment to check r-symmetry of $f = x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n + x_nx_1$. The experimental result is shown in Table 1. The rows labeled n, **|BDD|**, and **CPU Time** show the input size, the OBDD size, and the run time in second, respectively. It shows our matrix reordering algorithm is very promising to check r-symmetry of Boolean functions.

Table 1: Results of Detecting R-Symmetry

n	5	8	16	32	64	128	256
|BDD|	10	20	52	116	244	500	1012
CPU Time	0.00	0.00	0.01	0.04	0.33	2.44	21.02

```
Algorithm AID(f, X)
Input: A Boolean function f w.r.t. the input set X;
Output: Ψ is an ordered set of distinguished inputs;
Begin
    MSS = Maximal-Symmetric-Set(f, X);
    Generate the initial Ψ based on MSS;
    if Ψ = X return Ψ;
    Type-0 signature ksig(f, 1) is used to update Ψ;
    if Ψ = X return Ψ;
    Type-1 signature ksig(f, f) is used to update Ψ;
    if Ψ = X return Ψ;
    for i = 0 to n do      /* optional */
        Type-2 signature ksig(f, S_i^n) is used to update Ψ;
        if Ψ = X return Ψ;
    endfor
    Type-3 signature ksig(f, x_i) is used to update Ψ;
    if Ψ = X return Ψ;
    Generate the bounded set X_b by MSS and Ψ;
    Reorder the OBDD of f with transpositional operator so
    the order of inputs in X_b is before the inputs in X − X_b;
    Let f = Σ_{i=1}^{m} (b_i(X_b) · u_i(X − X_b)) by Type-4 signature;
    for i = 1 to m do
        Ψ = Ψ + AID(u_i, X − X_b);
        if Ψ = X return Ψ;
    endfor
    return (Ψ);
End
```

Figure 3: The AID Algorithm.

5. ALGORITHM FOR INPUT DIFFERENTIATION (AID)

5.1 AID

Our *AID* algorithm starts by detecting all maximal symmetric sets MSS of $f(X)$ using the algorithm we proposed in [12]. Then we classify the MSS in terms of set size. For each maximal symmetric set $P_i \in MSS$, if P_i is unique in size $|P_i|$, then we throw the inputs of P_i into the initial ordered set Ψ; otherwise, we take an input x_i of P_i to represent P_i and throw the remaining inputs into Ψ. Take $f = x_1x_2x_3 + x_4x_5 + x_6x_7$ for example. It is clear $P_1 = \{x_1, x_2, x_3\}$, $P_2 = \{x_4, x_5\}$ and $P_3 = \{x_6, x_7\}$ are maximal symmetric sets. The inputs of P_1 are added into Ψ due to P_1 is unique on set size 3. Since P_2 and P_3 have the same set size, x_4 and x_6 are used to represent P_1 and P_2, respectively. Consequently, x_5 and x_7 are thrown into Ψ.

After generating the initial Ψ, we apply type 0, 1, 2, and 3 signatures in turn to add more inputs into Ψ. Once all inputs are distinguished, the algorithm returns Ψ immediately. If there still exist some inputs that can't be distinguished, the bounded set X_b is constructed based on Ψ and MSS. One thing we need to address is that the bounded set is not always equal to Ψ. Take the function $f = x_1x_2x_3+x_4x_5+x_6x_7$ for example. After applying all k-distance signatures except for type 4, we get $\Psi = [x_1, x_2, x_3, x_5, x_7]$ and two indistinguishable inputs x_4, x_6. So, x_5 and x_7 must be removed from Ψ to generate the bounded set $X_b = \{x_1, x_2, x_3\}$. Then w.r.t. X_b, we reorder the OBDD of f using *transpositional operator* [11] and decompose f as $\sum_{i=1}^{m}(b_i(X_b)\cdot u_i(X-X_b))$ based on type 4 k-distance signature. Then *AID* procedure is recursively applied to subfunctions u_i's to further distinguish the inputs in $X - X_b$. By the above description, the *AID* algorithm is shown in Fig 3.

5.2 Boolean Matching Algorithm

Our Boolean matching method is mainly based on *AID*, $\mathcal{G}$-symmetry detection, and structural transformation of OBDD's. To check if two functions $f(X)$ and $g(Y)$ are equivalent under input permutation, our algorithm starts by applying *AID* algorithm to f and g to find their distinguishable inputs as many as possible. If all the inputs of f and g can be distinguished and have the same signature for each ordered input pair, then the OBDD representing f can be reordered to check if the transformed OBDD is *structural equivalent* [11] to the OBDD representing g. For the situation that some inputs can't be distinguished by *AID*, we may apply our $\mathcal{G}$-symmetry detection method to find $\mathcal{G}$-symmetry of f and g, respectively. Combining the result by *AID* and detected $\mathcal{G}$-symmetry, an assignment ψ mapping X to Y can be identified. Lastly, the same reordering and structural equivalence checking scheme can be applied to check whether f and g are matched or not. The above Boolean matching algorithm is shown in Fig. 4.

```
Algorithm Boolean-Matching(f(X), g(Y))
Input:  f(X) and g(Y)
Output: π is a feasible assignment;
Begin
    if |f| ≠ |g| return (∅);
    Ψ_1 = AID(f, X);     Ψ_2 = AID(g, Y);
    if Ψ_1 ≠ Ψ_2 return (∅);
    G_1 = G-Symmetry of f(X) utilizing Ψ_1;
    G_2 = G-Symmetry of g(Y) utilizing Ψ_2;
    Let ψ be an assignment w.r.t. (Ψ_1, G_1) and (Ψ_2, G_2);
    Reorder f w.r.t. ψ;
    if f ≡ g return (ψ);     /* ≡: structural equivalent */
    return (∅) ;
End
```

Figure 4: The Boolean Matching Algorithm.

6. EXPERIMENTAL RESULTS

The proposed *AID* algorithm and $\mathcal{G}$-symmetry detection method have been implemented in C language on SUN-Blade 1000 station. 128 circuits from MCNC and ISCAS benchmarking sets have been tested. For each circuit, each output is tested individually. Overall, there are 17, 28, 63, 71, and 78 out of 128 circuits with all inputs that can be distinguished by type 0, 1, 2, 3 k-distance signatures, and *AID*, respectively. The ratios are 13%, 22%, 49%, 55%, and 61%, respectively.

Table 2 shows the partial experimental results. The columns labeled **#in**, **#out**, and |**BDD**| are the numbers of inputs, the number of outputs, and the size of OBDD, respectively. The columns with labels **0**, **1**, **2**, **3** and **AID** show the average distinct factors of type 0, 1, 2, 3 k-distance signatures, and our *AID* algorithm, respectively. The **CPU Time** column shows the running time in second. The rows with labels **average** and **ratio** show the average distinct factor and runtime normalization, respectively. The average distinct factors of type 0, 1, 2, 3, and *AID* are 0.60, 0.67, 0.78, 0.76, and 0.81, respectively. It shows that *AID* is the most effective signature for being used in Boolean matching.

Table 2 also shows the result of testing our $\mathcal{G}$-symmetry detection method. The column **#O** show the number of outputs with indistinguishable inputs after running *AID*. The **+GSD** column shows the result of $\mathcal{G}$-symmetry detection. The columns labeled **#HS** and **#GS** show the number of outputs with *h-symmetry* and with *g-symmetry*, respectively. The result of *r-symmetry* is skipped since no r-symmetry exists for all circuits. The experimental result shows that the $\mathcal{G}$-symmetry of all tested circuits except for C1355 can be detected by our heuristic with little run time overhead (7%). In summary, our heuristic is very effective and efficient to check $\mathcal{G}$-symmetry of Boolean functions.

Table 2: Benchmarking Results for K-Distance Signatures and AID

circuit	#in	#out	|BDD|	K-Distance Signature (k=1)					+GSD			CPU Time (sec.)						
				0	1	2	3	AID	#O	#HS	#GS	0	1	2	3	AID	+GSD	BM
alu4	14	8	637	0.18	0.42	0.90	0.90	0.90	1	0	1	0.00	0.00	0.00	0.00	0.00	0.01	0.02
apex3	54	50	854	0.73	0.79	0.92	0.99	0.99	1	1	0	0.08	0.12	2.13	0.59	0.58	0.63	1.20
apex5	117	88	1121	0.88	0.93	0.93	0.94	0.95	8	8	0	0.10	0.12	0.20	0.11	0.13	0.18	0.37
apex6	135	99	569	0.80	0.88	0.89	0.98	0.99	5	0	5	0.01	0.02	0.07	0.04	0.05	0.05	0.10
clma	415	115	1304	0.89	0.93	0.98	0.98	0.99	3	0	3	0.05	0.07	0.52	0.29	0.21	0.27	0.50
cm150a	21	1	33	0.05	0.05	0.14	0.14	0.14	1	0	1	0.00	0.00	0.01	0.01	0.01	0.01	0.02
cm151a	12	2	17	0.08	0.08	0.25	0.25	0.25	2	0	2	0.00	0.00	0.00	0.01	0.00	0.00	0.01
cordic	23	2	63	0.83	0.83	0.83	0.83	0.83	2	2	0	0.00	0.00	0.01	0.00	0.00	0.01	0.02
dalu	75	16	804	0.35	0.39	0.57	0.63	0.72	13	0	13	0.18	0.21	2.64	1.10	1.15	1.47	3.01
des	256	245	2922	0.72	0.81	0.97	0.95	1.00	-	-	-	0.20	0.23	0.62	0.57	0.60	-	1.17
dsip	452	421	2460	0.73	0.82	0.91	1.00	1.00	-	-	-	0.11	0.11	0.60	0.18	0.18	-	0.35
ecc	127	130	2276	0.91	0.92	0.99	0.99	0.99	2	0	2	0.19	0.27	0.36	0.30	0.17	0.18	0.34
frg2	143	139	975	0.79	0.80	0.84	0.84	0.87	33	0	33	0.03	0.04	0.20	0.21	0.20	0.30	0.58
gcd	78	85	1200	0.92	0.94	0.97	0.99	0.99	11	11	0	0.07	0.11	0.61	0.37	0.16	0.19	0.41
i10	257	224	30375	0.94	0.96	0.98	0.98	0.98	16	8	8	4.23	7.09	16.70	7.22	17.56	17.71	38.59
mm9a	39	36	1113	0.42	0.54	0.66	0.66	0.66	16	0	16	0.04	0.09	0.14	0.12	0.13	0.16	0.30
pair	173	137	2842	0.70	0.92	0.97	0.97	0.97	9	8	1	0.13	0.31	0.78	0.40	0.33	0.41	0.89
rot	135	107	2871	0.86	0.91	0.97	0.98	0.98	11	2	9	0.13	0.28	1.87	0.60	0.39	0.58	1.23
s1423	91	79	1823	0.58	0.73	0.91	0.89	0.94	27	26	1	0.10	0.17	1.63	0.80	0.62	0.65	1.32
s38584.1	1464	1730	14539	0.83	0.93	0.97	0.97	0.98	208	171	37	2.07	2.68	24.81	5.57	24.76	26.46	54.10
s4863	153	120	56483	0.56	0.58	0.64	0.63	0.85	47	1	46	1.50	3.42	3.93	4.39	2.96	4.22	8.81
s6669	322	294	21803	0.72	0.86	0.87	0.87	0.87	47	2	45	0.37	1.01	3.18	2.85	2.79	4.23	8.15
seq	41	35	1202	0.94	0.97	0.99	0.99	1.00	-	-	-	0.12	0.10	0.17	0.11	0.08	-	0.15
t481	16	1	21	0.00	0.00	0.00	0.00	0.00	1	0	1	0.00	0.00	0.00	0.01	0.00	0.01	0.03
term1	34	10	81	0.61	0.64	0.70	0.65	0.70	8	5	3	0.01	0.01	0.03	0.02	0.02	0.03	0.06
x3	135	99	562	0.80	0.88	0.89	0.98	0.99	5	0	5	0.02	0.03	0.07	0.04	0.04	0.05	0.11
C1355 *	41	32	25874	0.02	0.05	0.07	0.05	0.07	32	0	0	0.55	1.45	43.02	32.52	47.23	47.50	130.24
C1908	33	25	6461	0.18	0.30	0.97	0.40	1.00	-	-	-	0.17	0.52	7.28	5.57	2.71	-	5.40
C2670	233	140	1937	0.80	0.81	0.87	0.86	0.88	32	0	32	0.14	0.34	12.38	3.58	27.26	33.50	70.06
C3540	50	22	24129	0.64	0.86	0.93	0.95	0.98	1	1	1	2.22	4.12	23.42	5.36	5.76	6.35	13.10
C432	36	7	1226	0.06	0.06	0.06	0.06	0.06	7	0	7	0.05	0.11	1.05	0.98	0.13	1.00	2.10
C5315	178	123	2064	0.43	0.48	0.95	0.76	0.96	8	0	8	0.17	0.43	3.53	7.57	3.48	3.71	7.96
C7552	207	108	2299	0.64	0.68	0.99	0.79	0.99	4	4	0	0.51	0.71	18.38	12.88	18.01	18.50	38.28
C880	60	26	4626	0.99	0.99	0.99	0.99	0.99	3	3	0	0.17	0.45	0.74	0.48	0.21	0.24	0.50
average				0.60	0.67	0.78	0.76	0.81				0.41	0.72	5.03	2.79	4.72	5.06	11.42
ratio									1.00	0.45	0.50	0.08	0.15	1.07	0.59	1.00	1.07	2.42

Table 3: Boolean Matching with Cell Libraries

circuit	#in	#out	|BDD|	CPU Time (ms.)	
				total	avg.
33-4	9	87	72	20	0.23
43-5	12	396	900	80	0.20
44-3	16	625	1588	150	0.24
44-6	16	3503	8556	800	0.23

We also conduct an experiment to test our Boolean matching algorithm. For each benchmarking circuit, we match it with a new circuit generated by randomly permuting its inputs variables. In Table 2, the column with label **BM** shows the running time for Boolean matching. The experimental result demonstrates our algorithm is very efficient to solve Boolean matching problem. Moreover, we test several large cell libraries with the same experimental flow. The result is summarized in Table 3. In this table, the columns **#in** and **#out** show the input number of the largest cell and the number of cells in the library, respectively. The columns **total** and **avg.** show the matching time of all cells and the average matching time for each cell in millisecond (ms.). The experimental result shows the average matching time is less than 0.25 ms. It is very promising to encourage us to apply our Boolean matching technique for technology mapping.

7. CONCLUSION

In this paper, we had defined the k-distance signature which is a general form of many existing signatures. Based on k-distance signature, an *AID* algorithm which can distinguish almost all inputs of Boolean functions without $\mathcal{G}$-symmetry was also proposed. Moreover, we addressed the issue of detecting $\mathcal{G}$-symmetry and shown a heuristic to solve this problem. The experimental results on a set of benchmarks show that our *AID* algorithm and $\mathcal{G}$-symmetry detection method are not only effective but also efficient for being applied in Boolean matching for logic verification and technology mapping.

8. REFERENCES

[1] F. Mailhot and G. De Micheli, "Technology mapping using Boolean matching and don't care sets," in *Proc. European Design Automation Conf.*, pp. 212-216, 1990.

[2] U. Hinsberger and R. Kolla, " Boolean matching for large libraries," in *Proc. Design Automation Conf.*, pp. 206-211, 1998.

[3] D. I. Cheng and M. Marek Sadowska, "Verifying equivalence of functions with unknown input correspondence," in *Proc. European Design Automation Conf.*, pp. 81-85, Feb. 1993.

[4] J. Mohnke and S. Malik, "Permutation and phase independent Boolean comparison," *INTEGRATION - the VLSI Journal*, Vol. 16, no. 2, pp. 109-129, 1993.

[5] J. Ciric and C. Sechen, "Efficient canonical form for Boolean mathcing of complex functions in large library," *IEEE Trans. Computer-Aided Design*, Vol. 22, No. 5, pp. 535-544, May 2003.

[6] A. Abdollahi and M. Pedram, "A new canonical form for fast Boolean matching in logic synthesis and verification," in *Proc. Design Automation Conf.*, pp. 379-384, June 2005.

[7] J. Schlichtmann, F. Brglez, and P. Schneider, "Efficient Boolean matching based on unique variable ordering," in *Proc. Int. Workshop on Logic Synthesis*, May 1993.

[8] J. Mohnke, P. Molitor, and S. Malik, "Limits of using signatures for permutation independent Boolean comparison," in *Proc. ASP Design Automation Conf.*, pp. 459-464, 1995.

[9] I. Pomeranz and S. M. Reddy, "On determining symmetries in inputs of logic circuits," *IEEE Trans. Computer-Aided Design*, Vol. 13, No. 11, pp. 1428-1434, Nov. 1994.

[10] V. N. Kravets and K. A. Sakallah, "Generalized symmetries in Boolean functions," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 526-532, Nov. 2000.

[11] K. H. Wang, T. Hwang, and C. Chen, "Restructuring Binary Decision Diagrams Based on Functional Equivalence," in *Proc. European Design Automation Conf.*, pp. 261-265, 1993.

[12] K. H. Wang and J. H. Chen, "K-disjointness paradiagm with application to symmetry detection of incompletely specified functions," in *Proc. ASP Design Automation Conf.*, pp. 994-997, Jan. 2005.