International Journal of Automation and Computing 7(1), February 2010, 70-77
DOI: 10.1007/s11633-010-0070-2

Cultural Algorithm for Minimization of Binary Decision Diagram and Its Application in Crosstalk Fault Detection

Zhong-Liang Pan[1] Ling Chen[1] Guang-Zhao Zhang[2]

[1]Department of Electronics, South China Normal University, Guangzhou 510006, PRC

[2]Department of Electronics and Communications, Sun Yat-sen University, Guangzhou 510275, PRC

Abstract: The binary decision diagrams (BDDs) can give canonical representation to Boolean functions; they have wide applications in the design and verification of digital systems. A new method based on cultural algorithms for minimizing the size of BDDs is presented in this paper. First of all, the coding of an individual representing a BDDs is given, and the fitness of an individual is defined. The population is built by a set of the individuals. Second, the implementations based on cultural algorithms for the minimization of BDDs, i.e., the designs of belief space and population space, and the designs of acceptance function and influence function, are given in detail. Third, the fault detection approaches using BDDs for digital circuits are studied. A new method for the detection of crosstalk faults by using BDDs is presented. Experimental results on a number of digital circuits show that the BDDs with small number of nodes can be obtained by the method proposed in this paper, and all test vectors of a fault in digital circuits can also be produced.

Keywords: Digital circuits, binary decision diagrams (BDDs), cultural algorithms, variable order, fault detection.

1 Introduction

Many tasks in combinatorial optimization, mathematical logic, artificial intelligence, digital system design and other areas can be formulated in terms of Boolean functions. The related work is carried out by Boolean manipulations on Boolean functions. The efficiency of Boolean manipulation depends on the form of representation of Boolean functions. The binary decision diagram (BDD) is a graph representation of Boolean functions, where the functions are represented as directed acyclic graphs, with internal vertices corresponding to the variables over which the function is defined and terminal vertices labeled by the function values 0 or 1. The operations on Boolean functions can be implemented as graph algorithms operating on BDDs[1].

One important property of BDDs is that it gives canonical form to Boolean functions, and the size of the graph, i.e., the number of nodes in the graph, is feasible for many practical functions[2]. To date, most applications of BDDs have been in the areas of digital system design, verification, and testing[3, 4].

One feature of BDDs is that they are very sensitive to the variable ordering; the variable ordering used can have a significant impact on the size (i.e., the number of nodes) of the BDD graph, from linear to exponential in the number of variables. Therefore, there is a need to find a variable ordering that minimizes the size of a BDDs. For this, several methods have been presented to determine good orderings in the last few years. For example, the methods were presented which use initial heuristics starting from the structure of a circuit, and gradual improvement heuristics based on the exchange of variables in the BDDs[5]. A minimization method for the BDDs by using the ordered best-first search and branch-and-bound algorithm was presented in [6]. The BDD minimization by scatter search was studied in [7]. The implicit techniques[8], symmetry properties[9], and linear transformations[10], were used in the variable ordering that minimizes the size of a BDD, respectively.

In this paper, first of all, a new method for the minimization of BDDs, i.e., the variable order based on cultural algorithms, is presented. Second, we study the applications of the method for the detection of faults in digital circuits. In digital circuits, a crosstalk fault may be caused by the interference effect between interconnection wires[11]. Two main types of crosstalk faults are crosstalk-induced glitch and crosstalk-induced delay; they are called crosstalk glitch and crosstalk delay, respectively. For two lines in a circuit, if the signal transition of 0 to 1 or 1 to 0 on a line produces coupling effects on another line, then the signal line is called an aggressor line, and the other line is called a victim line. For instance, if the victim line and aggressor line are driven respectively by a static 0 and a fast-rising (0 to 1) transition, then the crosstalk positive glitch is generated in the victim′s output signal. If the height of crosstalk glitch happens to be larger than the upper-threshold value of logic-low voltage for the given technology, it will produce logic failures (functionality problem).

A crosstalk delay is produced when both the aggressor and victim lines have near-simultaneous transitions. If both lines transit in the same direction, their transition times are reduced; hence, the effective delay is reduced. These changes in signal propagation delays can cause incorrect logic operations. Therefore, it is needed to tackle in circuit validation and test for crosstalk[12].

Agarwal[13] proposed an analytical framework to model crosstalk noise in coupled interconnection. The framework is based on transmission-line theory and captures high-frequency effects in on-chip interconnection. Wu and Lee[14] studied the built-in self-test scheme in the crosstalk faults detection for the deep-submicron very large scale integrated (VLSI) circuits; the scheme used a periodic

Manuscript received March 3, 2009; revised May 26, 2009
This work was supported by Natural Science Foundation of Guangdong Provincial of China (No. 7005833).

square wave as test signal which was generated by a linear feedback shift register. It simplifies test generation and test application while obviating the fault occurrence timing issue. Yang and Mourad[15] studied the effects of crosstalk-induced faults due to parameter variation during the manufacture of dynamic random access memories (DRAMs), and the results indicated that there exist worst case data patterns in each physical RAM block and cell arrangement. Liu and Jone[16] studied a test pattern generation method to detect the maximum crosstalk noise for programmable logic arrays (PLA); the method was based on the characteristics of dynamic PLA crosstalk noise. In this paper, a new test method for the detection of crosstalk faults in digital circuits by using BDDs is presented.

2 Binary decision diagrams

A BDD is a directed acyclic graph representation of a Boolean function. With the BDD, the functions can be constructed, manipulated, and compared by simple and efficient graph algorithms.

In a BDD, the set of nodes can be divided into terminal nodes (also called leaf nodes) and nonterminal nodes. Each nonterminal node is labeled with a variable and has edges directed toward two children: the 0-branch (shown as a dashed line) corresponds to the case where the variable is assigned 0, and 1-branch (shown as a solid line) correspond to the case where the variable is assigned 1. Each leaf node is labeled 0 or 1 to correspond to the value of the function. Each path from the root to a leaf node corresponds to a truth table entry where the edges in the path correspond to the assignment of the Boolean variables, and the value of the leaf node is the value of the function under that assignment. For example, the BDD is shown in Fig. 1 for logic function $f = x_1x_2 + x_3$.

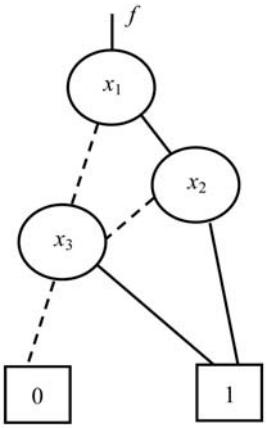


Fig. 1 The BDD of function f

A BDD is called ordered if each variable is encountered at most once on each path from the root to leaf node and if the variables are encountered in the same order on all such paths. We can reduce the size of a BDD by deleting its many isomorphic sub-graphs. Therefore, a BDD is called reduced if it does not contain nodes either with isomorphic sub-graphs or with both edges pointing to the same nodes. The BDD that is reduced and ordered is also called reduced ordered binary decision diagram (ROBDD), with a crucial feature: for any logic function, there is exactly one ROBDD representing it. Due to this uniqueness property, a BDD can be exponentially more compact than its corresponding truth table representation.

For a logic function, the size (i.e., the number of nodes) in its BDD depends on the variable order used. For example, let $g = x_1x_2+x_3x_4+x_5x_6$. The BDD of g is shown in Fig. 2 if the variable ordering is $x_1 < x_2 < x_3 < x_4 < x_5 < x_6$. The number of nodes in the variable ordering is 8. If the variable ordering is $x_1 < x_3 < x_5 < x_2 < x_4 < x_6$, the number of nodes in the variable ordering is 16.

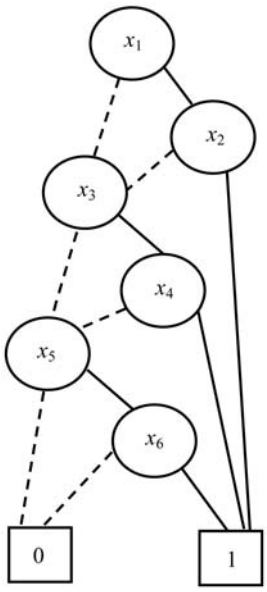


Fig. 2 The BDD of function g

As can be seen from Fig. 2, the choice of the variable order largely influences the size of BDD. It has been shown that finding an optimal variable ordering starting from a given BDD representation is NP-hard[17]. In Section 3, we present a method for variable order based on cultural algorithms.

3 Minimization of BDD by cultural algorithm

3.1 Cultural algorithms

The cultural algorithm is a method adding domain knowledge to evolutionary computation. It is based on the assumption that domain knowledge can be extracted during the evolutionary process, and in return, it makes use of that knowledge to guide the search[18−21]. This process of extraction and use of the domain knowledge has been shown to be very effective in decreasing computational cost.

The cultural algorithm consists of three major components: a population space, a belief space, and a protocol that describes how knowledge is exchanged between the first two components[22]. The population space consists of a set of possible solutions to a given problem. The belief space is the information repository in which the individuals can store their experiences for other individuals to learn indirectly. In cultural algorithm, the information acquired by an individual can be shared with the entire population.

The population space and belief space are linked through a communication protocol, which indicates the type of information to be exchanged between the two spaces, and states the rules about the individuals that can contribute to the belief space (i.e., acceptance function), and the way that the belief space can influence to new individuals in the population space (i.e., influence function).

The acceptance function produces belief information by gleaning the experience of individuals from the population space. The influence function makes domain knowledge guide the evolution of the individuals in population space. The belief space is improved by updating function, the idea being to add new knowledge acquired by the accepted individuals.

The new individuals in population space are produced by a population-based technique, where three functions, generate function, select function and objective function, are used for this purpose. The generate function is an operator to generate new individuals so that it includes the influence of belief space in the generation of offspring; the select function is the operator to select those that will be parent individuals in the next generation. The objective function represents objective function.

There are two main aspects in the implementations of the minimization of BDDs by using cultural algorithms: 1) the representation of individual and the fitness of an individual, and 2) the implementation steps of cultural algorithms. They will be discussed in detail in the following.

3.2 Individuals representation and fitness

Suppose all Boolean functions are expressed over the variables $x_1, x_2, \cdots, x_n$. We use an integer string of length n as the coding of an individual corresponding to a BDD, where n denotes the number of variables of the BDD being considered. Each integer string represents a variable ordering. For example, for a given circuit, we use (1 3 5 2 4 6) as the individual representation corresponding to the BDD under the variable ordering $x_1 < x_3 < x_5 < x_2 < x_4 < x_6$. A population is a set of these individuals. The fitness of an individual is related to the number of nodes of BDD corresponding to the chosen variable ordering. If the number of nodes is small, then the fitness is high.

3.3 BDD minimization by cultural algrithm

In this section, we design a method based on cultural algorithm to minimize the size of BDDs. The method is as follows.

Algorithm 1.

Step 1. Produce initial population to initialize the population space and belief space, respectively.

Step 2. Perform evolutionary operations for each individual in the population space by an integration technique of genetic algorithm and simulated annealing.

Step 3. Carry out the evolutionary operations for each individual in the belief space by the operations of selection, crossover, and mutation in the conventional genetic algorithm.

Step 4. Update the belief space with the accepted new individuals in the population space.

Step 5. If the termination condition is satisfied, then the whole procedure is terminated, else go to Step 2.

In Algorithm 1, the detail implementations for the designs of belief space and population space, acceptance function, influence function, etc. are given as follows.

1) The design of the population space

We use an integration method of genetic algorithm and simulated annealing to produce new individuals in the population space. The simulated annealing is a powerful optimization technique, which can theoretically converge asymptotically to the global optimum solution when the initial temperature is high enough and cooling rate is infinitely slow[23, 24]. The integration method can improve the efficiency of genetic algorithm in a certain degree. The principle of the integration is to produce new individuals with genetic algorithm; then these individuals are processed with simulated annealing, and the results are used as the new individuals of the next generation. The procedure of the integration method is as follows.

Algorithm 2.

Step 1. Initialize the temperature parameter T, i.e., set $T = T_0$, in which T_0 is a large positive number.

Step 2. Produce the initial population made up of n individuals.

Step 3. Compute the fitness of each individual in the initial population.

Step 4. Repeat:

Step 4.1. Carry out evolutionary operations, i.e., selection, crossover, and mutation, for individuals in the current population.

Step 4.2. Let C_j be the child individual produced by a parent individual P_j, $j = 1, \cdots, n$.

Step 4.3. Compute the fitness $E(C_j)$ of new individual C_j, $j = 1, \cdots, n$.

Step 4.4. For $j = 1, \cdots, n$, compute $\triangle E = E(C_j) - E(P_j)$, where $E(P_j)$ is the fitness of parent individual P_j. If $\triangle E \geqslant 0$, then produce a number ε with uniform distribution in interval [0,1]. If $\exp|(-\triangle E/T) > \varepsilon$, then replace individual P_j by child individual C_j. If $\triangle E < 0$, then discard the child individual C_j.

Step 4.5. If the termination condition is satisfied, then the whole procedure is stopped, else the value of temperature T is decreased.

In Algorithm 2, for the selection, crossover, and mutation operators, we adopt the implementations in the conventional genetic algorithms, for example, the roulette wheel selection scheme and two-point crossover.

2) The design of belief space

First of all, the encoding of belief individual in belief space adopts the same encoding used in the population space. At the beginning of the method, all chromosomes of initial individuals are put up to be empty set. Second, the number of belief individuals in the belief space is chosen as 15 % of the number of individuals in the population space. Third, the evolutions of individuals in the belief space are carried out by selection, crossover, and mutation operations in conventional genetic algorithms.

3) The acceptance function

The acceptance function determines which individuals and their behaviors may impact the knowledge of the belief space. It is determined by a percentage of the number of

individuals in the population space. Here, we use the following mode: the individuals with minimal fitness in the belief space are replaced with the individual with maximal fitness in the population space, when the individuals in the population space have evolved K generations, a given number of generations. For example, $K = 10$.

4) The influence operation

When the individuals in the population space have evolved M generations, we use a part of individuals (e.g., 20 %) in the belief space to replace the same number of individuals with lower fitness in the population space. Let M be

$$M = K_0 + L_0 \frac{N_{\max} - N_C}{N_{\max}}.$$

$N_{\max}$ is the maximal number of evolution generations to be set; N_C is the current number of evolution generations; K_0 and L_0 are constants. We choose $K_0 = 10$ and $L_0 = 100$. Using above equation for M, we can achieve that the belief space has little impact on the population space in the initial stage of the whole algorithm. As the algorithm is in progress, the belief space will produce more and more influences for the population space, therefore making the search ranges of the whole algorithm enlarged.

5) The use of belief knowledge

In general, there are a few types of knowledge in the belief space. In our method of minimizing BDDs based on cultural algorithms, normative knowledge is used. The normative knowledge is represented as a set of intervals, and each interval is viewed to be a promising range for good solutions for a parameter. These ranges provide guidelines within which individual adjustments can be made. The normative knowledge provides norms or standards for individual behavior and guidelines for individuals to follow.

The normative knowledge is used in genetic operations such as selection and mutation when we carry out the evolutions for individual in the belief space and make influences on the individuals in the population space.

3.4 Building BDD of a circuit

It is necessary for us to build the BDDs of a circuit when we minimize the BDDs by the method based on cultural algorithms in this paper. In this section, we give the following approach to obtain the BDD corresponding to a circuit.

A digital circuit consists of many circuit blocks. A circuit block implements a specific logic function. The logic function of the whole circuit is expressed as a sequence of operations on the Boolean functions realized by these circuit blocks. This needs a fast method to build the BDD of the whole circuit by using these BDDs of all circuit blocks.

An operator, *ite*, is defined as follows: for logic functions f, g, and h, $ite(f, g, h) = f \cdot g + \overline{f} \cdot h$. The operator $ite(\cdot)$ can be used to realize all Boolean operations with two variables. For example, $f + g = ite(f, 1, g)$, $f \cdot g = ite(f, g, 0)$, and $f \oplus g = ite(f, \overline{g}, g)$, where $\oplus$ denotes an XOR operation.

Let F be the BDD corresponding to logic function f. Let G and H be the BDD corresponding to logic functions g and h, respectively. We use the following method to compute the BDD corresponding to $ite(F, G, H)$. The Shannon's decomposition of F is as follows:

$$F = w \cdot F_w + \overline{w} \cdot F_{\overline{w}} \quad (1)$$

where $w \in \{x_1, x_2, \cdots, x_n\}$, F_w denotes $F|_{w=1}$, and $F_{\overline{w}}$ denotes $F|_{w=0}$. F_w and $F_{\overline{w}}$ are F evaluated at $w = 1$ and $w = 0$, respectively. From (1), we get

$$ite(F, G, H) = ite(v, ite(F_v, G_v, H_v), ite(F_{\overline{v}}, G_{\overline{v}}, H_{\overline{v}})) \quad (2)$$

where $v \in \{x_1, x_2, \cdots, x_n\}$. Equation (2) is true because

$$\begin{aligned} ite(F, G, H) =& FG + \overline{F}H = \\ & v(FG + \overline{F}H)_v + \overline{v}(FG + \overline{F}H)_{\overline{v}} = \\ & ite(v, F_v G_v + \overline{F}_v H_v, F_{\overline{v}} G_{\overline{v}} + \overline{F}_{\overline{v}} H_{\overline{v}}) = \\ & ite(v, ite(F_v, G_v, H_v), ite(F_{\overline{v}}, G_{\overline{v}}, H_{\overline{v}})). \end{aligned}$$

The terminal cases of recursion equation (2) are $ite(1, F, G) = ite(0, G, F) = ite(F, 1, 0) = F$.

Here, an example of operator $ite(\cdot)$ is given in the following. Three BDDs: G_1, G_2, and G_3 are given in Fig. 3. The BDD corresponds to operator $ite(\cdot)$ for G_1, G_2, and G_3, as shown in Fig. 4.

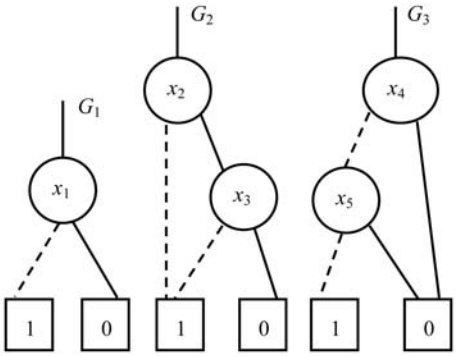


Fig. 3 Three BDDs

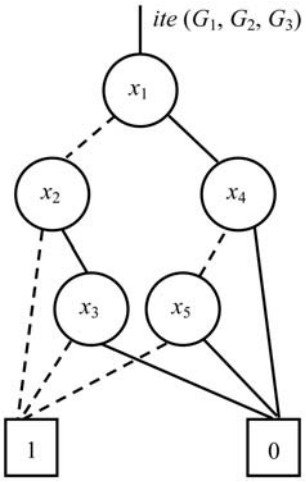


Fig. 4 The BDD example for operator $ite(\cdot)$

The BDD corresponds to that the whole circuit is built by the forward process. For example, for the circuit shown in Fig. 5, we start from the primary inputs of the circuit; each gate output is expressed in terms of primary inputs, and then these BDDs corresponding to the gate outputs are built. The BDDs corresponding to lines d and e are built. The BDDs corresponding to lines a, b, c, d, and e are shown in Figs. 6 (a)-(e), respectively.

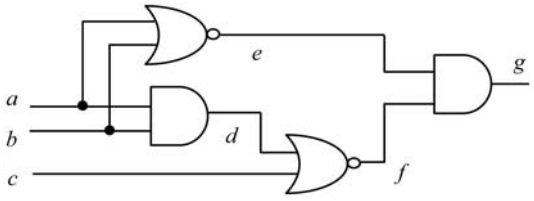
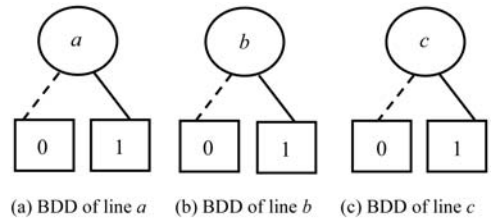
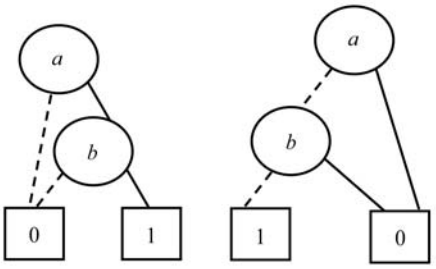


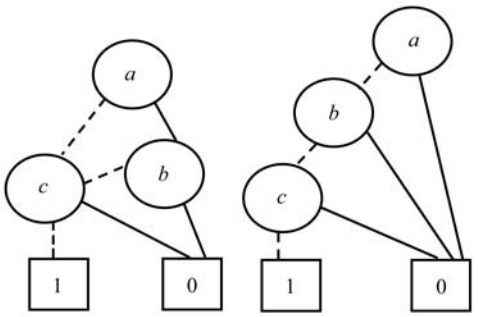
Fig. 5 An example of circuit



(a) BDD of line a (b) BDD of line b (c) BDD of line c



(d) BDD of line d (e) BDD of line e



(f) BDD of line f (g) BDD of line g

Fig. 6 The BDDs correspond to the whole circuit

Then, according to the logic expression of line f, we build the BDD corresponding to line f, which is shown in Fig. 6 (f). Finally, the BDD corresponding to line g is generated, which is shown in Fig. 6 (g). Thus, the BDD of the whole circuit is built. The above method of minimizing BDDs based on cultural algorithms has been implemented in C++ language; the experimental results will be given in Section 5.

4 Faults detection of digital circuit

As the VLSI circuits advance, the test and fault detection have become essential and important parts in the design and manufacturing of digital circuits. One of the main tasks of fault detection is to generate the test vectors of faults, i.e., test vector generation or test generation in short. In the following, the test generation approaches based on BDDs are given for stuck-at faults and crosstalk faults, respectively. The advantage of the approaches is that all test vectors of a given fault in digital circuits can be obtained. First of all, the detection of stuck-at faults is studied. The principle of test generation is as follows. If there is a fault in a circuit, then the test vectors of the fault are the input assignments that cause the faulty circuit and normal circuit (fault-free circuit) to produce different output values.

The test generation approach based on BDDs consists of the following three steps:

Step 1. The BDD for the normal circuit is built, and the BDD is called the normal BDD.

Step 2. The BDD for the faulty circuit is built, and the BDD is called the faulty BDD. The faulty BDD describes the functionality of faulty circuit. For a fault, when its normal BDD and faulty BDD are isomorphic, this shows that the fault is not testable, that is, there are not test vectors for the fault.

Step 3. A BDD called the test BDD is built, which is the XOR operation of the normal BDD and faulty BDD. In the test BDD, each input assignment that leads to the leaf node with attribute value 1 is a test vector of the fault.

We take the fault of line e_4 with s-a-0 in the C17 circuit shown in Fig. 7 as an example to show the test generation procedure. First of all, we build the BDD of y_2 in normal circuit, which is shown in Fig. 8 (a). Second, we build the BDD corresponding to faulty circuit. The BDD of the line $e4$ with s-a-0 is shown in Fig. 9 (a); it only contains a node with attribute value 0. The BDD of line e_3 is shown in Fig. 9 (b). The BDD corresponding to the faulty circuit is shown in Fig. 9 (c), which is obtained by the logic NAND operation of the BDDs for Figs. 9 (a) and (b).

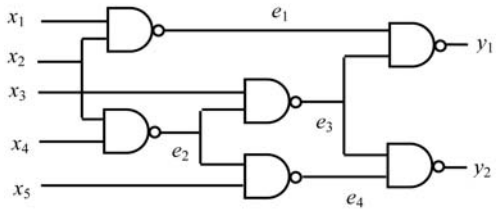
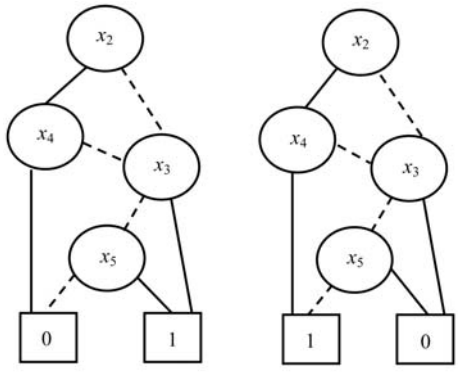


Fig. 7 C17 circuit



(a) BDD of normal circuit (b) The test BDD

Fig. 8 Building the test BDD

The test BDD is given in Fig. 8 (b), which is the XOR operation of the BDD shown in Fig. 8 (a) corresponding to the normal circuit and the BDD shown in Fig. 9 (c) corresponding to faulty circuit.

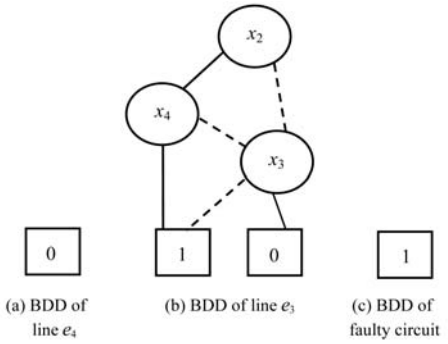


Fig. 9 Building the BDD of fault circuit

In the test BDD shown in Fig. 8 (b), there are three paths from root node to the leaf node with attribute value 1. They are 1) $x_2 = 1$, $x_4 = 1$; 2) $x_2 = 1$, $x_4 = 0$, $x_3 = 0$, $x_5 = 0$; 3) $x_2 = 0$, $x_3 = 0$, $x_5 = 0$. Therefore, the test vectors for fault e_4: s-a-0 are $(* \ 1 \ * \ 1 \ *)$, $(* \ 1 \ 0 \ 0 \ 0 \ )$, and $(* \ 0 \ 0 \ * \ 0)$, and these vectors have the forms of components (x_1 x_2 x_3 x_4 x_5), where * denotes that the value of the component is 0 or 1. In fact, these vectors are all test vectors of fault e_4: s-a-0.

Second, the detection of crosstalk fault is studied. The crosstalk fault is one of the interference effects being caused by parasitic capacitance and inductance coupling. For two lines in a circuit, if the signal transit of 0 to 1 or 1 to 0 on a line produces coupling effects on another line, then the line is called an aggressor line; the other line is called a victim line. Here, for circuit C17 shown in Fig. 7, we give an example for test generation when there is a crosstalk fault between signal lines e_3 and e_4.

Assume that e_4 is an aggressor line, and e_3 is a victim line. If a down transition (1 to 0) in signal line e_4 produces a glitch (1 to 0) in signal line e_3, then there is a crosstalk fault between lines e_3 and e_4. The task of test generation is to search for the inputs vectors of circuit C17 in order to detect the crosstalk fault. For example, a test vector of the crosstalk fault is made up of circuit input vectors V_1=(x_1, x_2, x_3, x_4, x_5)= $(0, 0, 0, 0, 0)$ and V_2 =(x_1, x_2, x_3, x_4, x_5)= $(0, 0, 0, 0, 1)$.

Apply V_1 and V_2 to the circuit sequentially. If the circuit outputs are $y_1 = 0$ and $y_2 = 0$ for V_1, $y_1 = 0$ and $y_2 = 1$ for V_2, then there is not crosstalk. If the circuit outputs are $y_1 = 0$ and $y_2 = 0$ for V_1, $y_1 = 1$ and $y_2 = 1$ for V_2, then there is the crosstalk. Therefore, this test vector can detect the crosstalk fault between lines e_3 and e_4.

In general, a test vector of a crosstalk fault is made up of a pair of circuit input vectors; for instance, for above V_1 and V_2, we call V_1 the first vector and V_2 the second vector in short.

In the following, we present a new approach for the test generation of crosstalk faults, which consists of the following steps:

Algorithm 3.

Produce the first-vector V_1 in the test vector of a crosstalk fault by Steps 1–3.

Step 1. Set the values of aggressor line and victim line to special values.

Step 2. Let C_1 be a sub-circuit of normal circuit; C_1 consists of the signal lines that can produce effects on the values of aggressor line or victim line. Build the BDD corresponding to the sub-circuit C_1. Name the BDD as B_1.

Step 3. In B_1, search for the input assignments that lead to the leaf nodes with the special values of aggressor line and victim line. Then, each of such input assignments is first-vector V_1 in the test vector of a given crosstalk fault.

Produce the second-vector V_2 in the test vector of a crosstalk fault by Steps 4–7.

Step 4. Build the BDD for the normal circuit; name the BDD as B_2.

Step 5. Build the BDD for the faulty circuit; name the BDD as B_3.

Step 6. Build the BDD that is the XOR operation of B_2 and B_3; name the BDD as B_4.

Step 7. In B_4, search for the input assignments that lead to the leaf node with attribute value 1. Then, each of such input assignments is second-vector V_2 in the test vector of a given crosstalk fault.

We take the crosstalk fault between lines e_3 and e_4 in C17 circuit as an example to indicate Algorithm 3. Here, assume that e_4 is an aggressor line and e_3 is a victim line, and that a down transition (1 to 0) in signal line e_4 produces a glitch (1 to 0) in signal line e_3.

For the generation of the first-vector V_1 in the test vector, we set the values of signal lines e_4 and e_3 to be 1 for the normal circuit. Build the BDD B_1, and search for the input assignments that lead to the leaf nodes with the attribute value 1 for aggressor line and victim line. Thus, we obtain the first-vector V_1 in the test vector of the crosstalk fault. For example, we obtain such a vector that is $V_1 = (1, 1, 0, 1, 0)$.

For the generation of the second-vector V_2 in the test vector, we set the value of signal line e_4 as 0 and set the value of signal line e_3 to be 1 for the normal circuit, and build the BDD B_2. We set the value of signal line e_4 to 0 and set the value of signal line e_3 to 0 for the faulty circuit, and build the BDD B_3. Build the BDD that is the XOR operation of B_2 and B_3; name the BDD as B_4. In B_4, each input assignment that leads to the leaf node with attribute value 1 is the second-vector V_2 in the test vector of the crosstalk fault. For example, we obtain such a vector that is $V_2 = (1, 0, 0, 1, 1)$.

Therefore, $T = (V_1 = (1, 1, 0, 1, 0), V_2 = (1, 0, 0, 1, 1))$ is a test vector of the crosstalk fault between lines e_3 and e_4. If the vectors V_1 or V_2 cannot be obtained by Algorithm 3, then it is shown that there is no test vector for the crosstalk fault. Besides, if there are test vectors for the crosstalk fault, then all test vectors of the crosstalk fault can be produced by Algorithm 3.

5 Experimental results

The above methods for the minimization of BDDs based on cultural algorithms and the circuit fault detection by BDDs have been implemented in C++ language and run on a personal computer with a 3.0 GHz clock and 256 M main memory. We have performed a number of experiments for ISCAS′85 benchmark circuits C432, C499, C1355, C1908, and C2670. The total numbers of both lines and single stuck-at faults in these circuits are shown in Table 1.

Table 1 Experimental results for ISCAS85 circuits

Circuit names	In	Out	Lines	Faults	GA	CA
C432	36	7	432	524	1871	1274
C499	41	32	499	758	39118	24683
C1355	41	32	1355	1574	37062	21495
C1908	33	25	1980	1879	6147	5162
C2670	233	140	2670	2747	3952	2637

In Table 1, the column circuit names denotes the names of the benchmark circuits. The two columns "in" and "out" give the numbers of primary inputs and primary outputs in the benchmark circuits, respectively. The column "lines" denotes the total number of signal lines in a circuit. The column "faults" denotes the size of the simplistically reduced equivalent single stuck-at fault set for a circuit. The column "CA" denotes the size of the BDD obtained by the cultural algorithm, and "GA" denotes genetic algorithm. We use the following implementations for the crossover operations and mutation operations in the evolution of individuals in the belief space of cultural algorithms. For the crossover operation, first, two parents are selected; and second, two cut positions are chosen at random. Here, it is valuable to indicate that a simple exchange of the parts between the cut positions may produce an invalid solution. We produce the offspring by choosing the part between the cut positions from one parent and preserving the position and order of as many variables as possible from the second parent. For example, let P_1 =(3 1 4 5 6 2 7) and P_2 =(6 4 2 3 7 1 5). When the cut positions are the fourth and fifth, i.e., we need to exchange the (5 6) in P_1 and (3 7) in P_2, the produced solutions (3 1 4 3 7 2 7) and (6 4 2 5 6 1 5) are invalid if we carry out the simple exchange. The valid offspring for P_1 and P_2 should be (5 1 4 3 7 2 6) and (7 4 2 5 6 1 3).

For the mutation operation in the belief space, we select two component positions of a parent at random and perform the exchange of the values of these two positions. For example, for the individual P_3 =(6 1 2 5 7 4 3), an offspring produced is (6 4 2 5 7 1 3) by the mutation operation.

In these experiments, the following parameters are used in cultural algorithms: the maximal evolution generation is set as 1800; population size of population space is 80. The parameters in the conventional genetic algorithm used in the evolutions of individuals in belief space are as follows: population size of belief space is 20; crossover rate is 0.95; mutation rate is 0.001. The experimental results are shown in Table 1.

For the fault detection of digital circuits by BDDs, we carry out the test vector generation for the ISCAS85 benchmark circuits C17, C432, C499, C1355, C1908, and C2670. We use the variable orders obtained by the cultural algorithm to build the BDDs of normal circuits and faulty circuits.

For the crosstalk faults in the ISCAS85 benchmark circuits, we randomly select 10 crosstalk faults from these circuits to produce the test vectors. The experimental results show that if there are test vectors for a crosstalk fault, then the test vector can be produced by Algorithm 3; the time of producing all test vectors of a crosstalk fault is less than one minute.

For the stuck-at faults in the ISCAS85 benchmark circuits, the numbers of test vectors in the test set obtained for the circuits C432, C499, C1355, C1908, and C2670 are 39, 46, 52, 42, and 251, respectively. The experimental results show that all test vectors of a fault can be produced if the fault is testable.

For the stuck-at faults in the C17 circuit, the test set $Test$ obtained is made up of 11 vectors, i.e., $Test = \{$10000, 10011, 01101, 01110, 01000, 00001, 01111, 10001, 10111, 00101, 00011$\}$. These vectors in test set $Test$ have the forms of components (x_1 x_2 x_3 x_4 x_5); the time of generating the test set $Test$ is less than one minute.

Besides, for the minimization of BDDs, we also carry out many experiments by using conventional genetic algorithms. The following parameters are used in conventional genetic algorithms: the population size is 100; the maximal evolution generation is set as 1800; the roulette wheel selection scheme and two-point crossover are used; the crossover rate is 0.95; mutation rate is 0.001.The experimental results are given in Table 1 in column "GA". The experimental results demonstrate that the sizes of the BDD obtained by using the method based on cultural algorithms in this paper are smaller than the sizes of the BDD obtained by conventional genetic algorithms.

6 Conclusions

For a logic circuit, the size in its BDD depends on the variable order. A good variable order can make the size of BDD small; therefore, the computational efficiency can be enhanced when we use BDDs in the area such as in the design and verification of digital circuits. A new method based on cultural algorithms for the variable order of BDDs is presented in this paper. The application of the method for the detection of stuck-at faults and crosstalk faults in digital circuits is studied. A new method for the detection of crosstalk faults by using BDDs is presented. The experimental results show that the sizes of the BDD obtained by using the method based on cultural algorithms in this paper are smaller than those of the BDD obtained by conventional genetic algorithms, and all test vectors of a fault can be obtained by using BDDs. In order to improve the efficiency of the method in this paper, some work needs to be done in the future, such as the selections of better genetic parameters.

References

[1] K. Osnat. Reduction of average path length in binary decision diagrams by spectral methods. *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 520–531, 2008.

[2] D. Sawitzki. Lower bounds on the OBDD size of two fundamental functions′ graphs. *Information Processing Letters*, vol. 101, no. 2, pp. 66–71, 2007.

[3] M. Matsuura, T. Sasao. BDD representation for incompletely specified multiple-output logic functions and its applications to the design of LUT cascades. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 90, no. 12, pp. 2762–2769, 2007.

[4] R. S. Shelar, S. S. Sapatnekar. BDD decomposition for delay oriented pass transistor logic synthesis. *IEEE Transactions on Very Large Scale Integration Systems*, vol. 13, no. 8, pp. 957–970, 2005.

[5] M. B. Song, H. Chang. A variable reordering method for fast optimization of binary decision diagrams. In *Proceedings of the 6th Asian Test Symposium*, Akita, Japan, pp. 228-233, 1997.

[6] R. Ebendt, W. Gunther, R. Drechsler. Combining ordered best-first search with branch and bound for exact BDD minimization. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 24, no. 10, pp. 1515–1529, 2005.

[7] W. N. N. Hung, X. Song, E. M. Aboulhamid, M. A. Driscoll. BDD minimization by scatter search. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 21, no. 8, pp. 974–979, 2002.

[8] A. L. Oliveira, L. P. Carloni, T. Villa, A. L. Sangiovanni-Vincentelli. Exact minimization of binary decision diagrams using implicit techniques. *IEEE Transactions on Computers*, vol. 47, no. 11, pp. 1282–1296, 1998.

[9] L. Heinrich-Litan, P. Molitor. Least upper bounds for the size of OBDDs using symmetry properties. *IEEE Transactions on Computers*, vol. 49, no. 4, pp. 360–368, 2000.

[10] W. Gänther, R. Drechsler. Efficient minimization and manipulation of linearly transformed binary decision diagrams. *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1196–1209, 2003.

[11] B. K. Kaushik, S. Sarkar. Crosstalk analysis for a CMOS-gate-driven coupled interconnects. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1150–1154, 2008.

[12] T. Ciamulski, W. K. Gwarek. Coupling compensation concept applied to crosstalk cancelling in multiconductor transmission lines. *IEEE Transactions on Electromagnetic Compatibility*, vol. 50, no. 2, pp. 437–441, 2008.

[13] K. Agarwal, D. Sylvester, D. Blaauw. Modeling and analysis of crosstalk noise in coupled RLC interconnects. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 25, no. 5, pp. 892–901, 2006.

[14] M. S. Wu, C. L. Lee. Using a periodic square wave test signal to detect crosstalk faults. *IEEE Design and Test of Computers*, vol. 22, no. 2, pp. 160–169, 2005.

[15] Z. Yang, S. Mourad. Crosstalk induced fault analysis and test in DRAMs. *Journal of Electronic Testing: Theory and Applications*, vol. 22, no. 2, pp. 173–187, 2006.

[16] J. Liu, W. B. Jone, S. R. Das. Crosstalk test pattern generation for dynamic programmable logic arrays. *IEEE Transactions on Instrumentation and Measurement*, vol. 55, no. 4, pp. 1288–1302, 2006.

[17] B. Bollig, I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 993–1002, 1996.

[18] E. P. K. Tsang. Computational intelligence determines effective rationality. *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 63–66, 2008.

[19] C. A. C. Coello, R. L. Becerra. Evolutionary optimization through the use of a cultural algorithm. *Engineering Optimization*, vol. 36, no. 2, pp. 219–236, 2004.

[20] F. G. Zhao, J. S. Sun, S. J. Li, W. M. Liu. A hybrid genetic algorithm for the traveling salesman problem with pickup and delivery. *International Journal of Automation and Computing*, vol. 6, no. 1, pp. 97–102, 2009.

[21] T. K. Liu, C. H. Chen, Z. S. Li, J. H. Chou. Method of inequalities-based multiobjective genetic algorithm for optimizing a cart-double-pendulum system. *International Journal of Automation and Computing*, vol. 6, no. 1, pp. 29–37, 2009.

[22] X. Yuan, Y. Yuan. Application of cultural algorithm to generation scheduling of hydrothermal systems. *Energy Conversion and Management*, vol. 47, no. 15–16, pp. 2192–2201, 2006.

[23] K. I. Smith, E. M. Everson, J. E. Fieldsend. Dominance-based multi-objective simulated annealing. *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 3, pp. 323–342, 2008.

[24] D. J. Chen, C. Y. Lee, C. H. Park, P. Mendes. Parallelizing simulated annealing algorithms based on high-performance computer. *Journal of Global Optimization*, vol. 39, no. 2, pp. 261–289, 2007.

Zhong-Liang Pan received the M. Sc. degree from Tsinghua University, PRC in 1991, and the Ph. D. degree from University of Electronic Science and Technology of China, PRC in 1997. He is currently a professor with the South China Normal University, PRC.

His research interests include circuits and systems, very large scale integrated (VLSI) design and test, design for testability, and image processing.

E-mail: panz@scnu.edu.cn (Corresponding author)

Ling Chen received the B. Sc. degree in communication engineering from South China Normal University, PRC in 2004. She is currently an engineer with the South China Normal University.

Her research interests include circuits design and test, image processing, and intelligent information systems.

E-mail: chen067@126.com

Guangzhao Zhang is currently a professor in the Department of Electronics and Communications at Sun Yat-sen University, PRC.

His research interests include computer networks, optoelectronics, multimedia and communication technology, and circuits and systems.

E-mail: zhangzsu@sina.com