Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2021.015462

ARTICLE

An Evolutionary Algorithm for Non-Destructive Reverse Engineering of Integrated Circuits

Huan Zhang[1,2], Jiliu Zhou[1,2,*] and Xi Wu[2]

[1]College of Computer Science, Sichuan University, Chengdu, 610065, China

[2]School of Computer Science, Chengdu University of Information Technology, Chengdu, 610225, China

[*]Corresponding Author: Jiliu Zhou. Email: zhoujl@cuit.edu.cn

Received: 20 December 2020 Accepted: 09 February 2021

ABSTRACT

In hardware Trojan detection technology, destructive reverse engineering can restore an original integrated circuit with the highest accuracy. However, this method has a much higher overhead in terms of time, effort, and cost than bypass detection. This study proposes an algorithm, called mixed-feature gene expression programming, which applies non-destructive reverse engineering to the chip with bypass detection data. It aims to recover the original integrated circuit hardware, or else reveal the unknown circuit design in the chip.

KEYWORDS

Hardware Trojans; Trojan detection; mixed-feature; gene expression programming

1 Introduction

The term hardware Trojan refers either to a particular circuit module deliberately implanted or changed during the process of designing or manufacturing an integrated circuit (IC), or to an unintentional design defect in the IC [1–5]. Once activated, the Trojan may change the function or specifications of the IC, which may cause leaking of sensitive information, a decrease in performance, or even irreversible damage to the IC [6–9].

In the global semiconductor supply chain, the traditional security strategy of implementing protection based on the underlying hardware, is no longer valid [1]. As shown in Fig. 1 [2], untrusted entities participate, directly and indirectly, in all stages of the lifecycle of electronic devices and ICs. This leads to hardware security vulnerabilities in the chain of design, manufacturing, testing, deployment, and application [1,2]. In the design and production processes, attackers are capable of implanting Trojans on target chips at any stage, through various methods. The vulnerabilities further enable the attackers to tamper with the original design, reduce circuit performance, monitor, control, and wage denial of service, and disclose confidential information that may cause irreversible damage to IC [3,4].

In today's hardware Trojan detection technology [1–9], destructive reverse engineering [1,3, 10–12] restores the original IC to be detected with the highest accuracy, but it incurs a great

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

deal of time, effort, and money, and the chip will be destroyed after detection. Therefore, such application of detection technology has been limited to sampling or replication of a specific type of chip. Bypass detection [13–19] determines if an IC contains a Trojan by analyzing the bypass signals, such as timing, power, electromagnetics, and heat. This detection technology is considered as perhaps the most effective because it does not damage the chip and only a relatively small amount of data and resources are required.

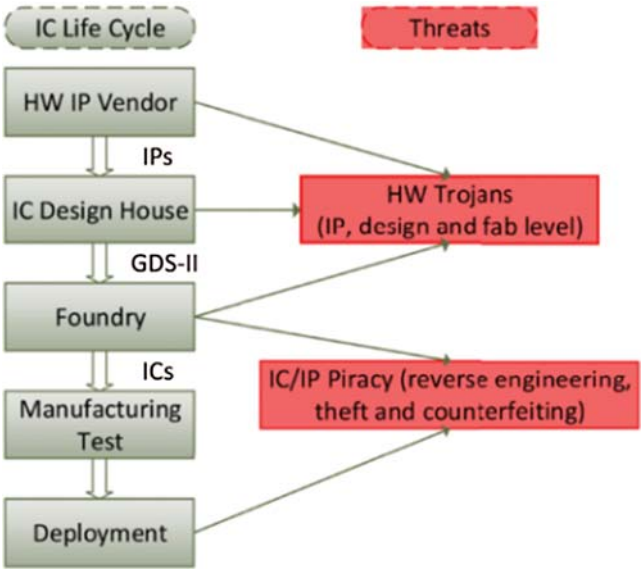


Figure 1: Threats in the IC and IP supply chain

Unlike some previous work that researched evolvable hardware [20–26] using evolutionary algorithms, this paper proposes an evolutionary algorithm called mixed-feature gene expression programming (MF-GEP) and attempts to find the original IC. The algorithm can be evolved by using a single circuit component or a group of circuit structures as the node and mixing multiple features into a single operator.

2 Related Works

2.1 Hardware Trojan

2.1.1 Composition and Attack Mechanism of Hardware Trojan

A hardware Trojan primarily consists of two components: the trigger logic and the payload. A model of the structure of a hardware Trojan is shown in Fig. 2. In this model, the payload is activated by the trigger logic through monitoring the input signal, data/control bus, register status, or a set work time. The payload is responsible for executing the attack.

2.1.2 Destructive Detection Methods

Destructive detection methods usually use destructive reverse engineering to decapsulate an IC and obtain images of each layer, so as to reproduce and verify the trusted design of the final product [1,3,10–12]. However, the process of reverse engineering is irreversible, which means an IC can no longer be used once the intrusion process is completed, and only the information of a single IC sample is available [7–9].

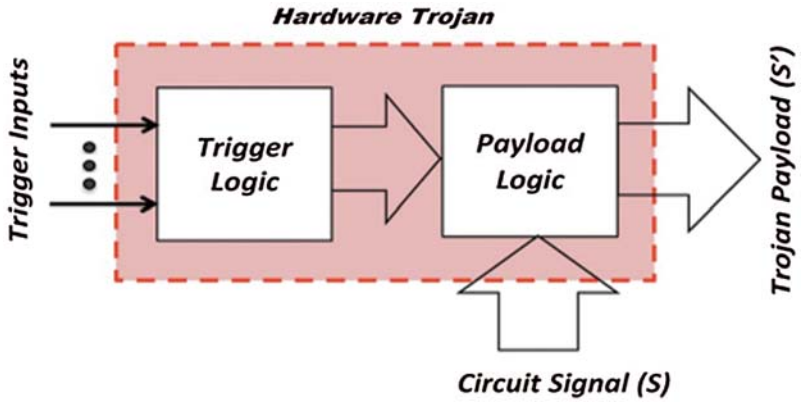


Figure 2: A hardware Trojan modifies signal S to S' when triggered [3]

Courbon et al. [10] proposed a fast-intrusive technique, which divided the IC to be tested into regions, and extracted images of each region by scanning before multiple images were stitched to reproduce the original design. Although this invasive method only works for hardware Trojans implanted by modifying functional units at the manufacturing stage, it significantly decreases the time and cost of the reverse process. Bhasin et al. [9,10] proposed the use of machine learning methods, such as support vector machine (SVM) and clustering (K-means), to identify an IC with no hardware Trojans.

The specific operation process of the SVM method is depicted in Fig. 3. This has been modified from [11].

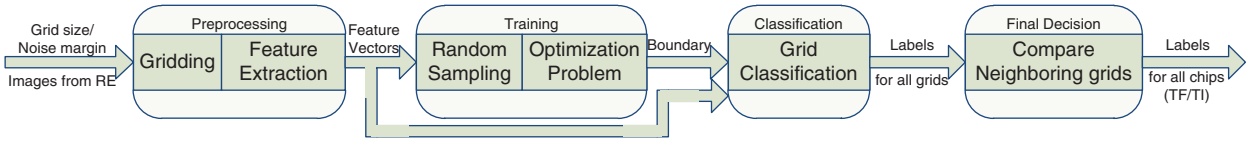


Figure 3: Hardware Trojan detection based on SVM

The author uses the golden layout of N chips to classify parameter values with grid size and noise margin d_{nm} as input. First, the images of each layer of N chips are obtained. Then, each layer of the chips is divided into a non-overlapping grid based on the obtained images. The size and noise margin d_{nm} of each grid are tested in the classification conditions. These grids are used to train the classifier and are subsequently classified as Trojan-free (TF) or Trojan-inserted (TI) through SVM. Finally, the classification results are used to identify whether the chip contains a Trojan.

2.1.3 Logic Test and Bypass Analysis Method

The main goal of the logic test method is to activate a hardware Trojan by applying function, structure, or random test vectors, and then compare the response result with the correct one [1,7]. However, by devising rare triggering conditions, attackers can escape traditional functional and structural tests during production testing. In addition, it is not practical to list all of the state nodes and gate circuits inside the IC. Methods based on logic testing cannot detect Trojans that do

not aim at tampering with the data or functions of the original circuit, but can detect an attempt to disclose confidential information through antennas or to modify design specifications [6].

The bypass analysis method detects hardware Trojans through the parameters of an IC during normal operation, such as delay [14], power consumption [13,15], thermal radiation [16], and electromagnetic [17,18]. The method makes full use of the features of changed bypass information that are caused by additional circuits or hardware Trojans, which compensates for the lack of logic test [1,3,9,19]. However, one of the biggest challenges of this method is that bypass information analysis techniques assume a comparable "golden model," which is often difficult to achieve in practical applications.

2.2 Evolvable Hardware

The concept of evolvable hardware (EHW) was initially proposed in 1993 by Hugo de Garis while at the Advanced Telecommunications Research Institute in Japan and scientists from the Swiss Federal Institute of Technology. EHW aims to make use of the reconfigurable internal structure of programmable devices, as well as the capabilities of the evolutionary algorithm in combinatorial optimization and universal search. The algorithm can help to locate the structure of the bit string combination of programmable devices for specific tasks, which obtains the hardware circuit with expected functions [21–27] through programming and configuration during the programmable periods.

For example [22], Fig. 4 illustrates the array of logic cells, and Fig. 5 shows its genotype.

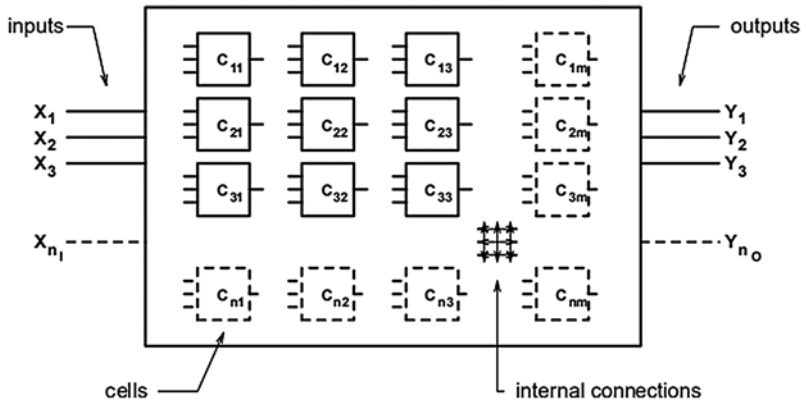


Figure 4: A digital circuit encoded within a genotype by an array of logic cells

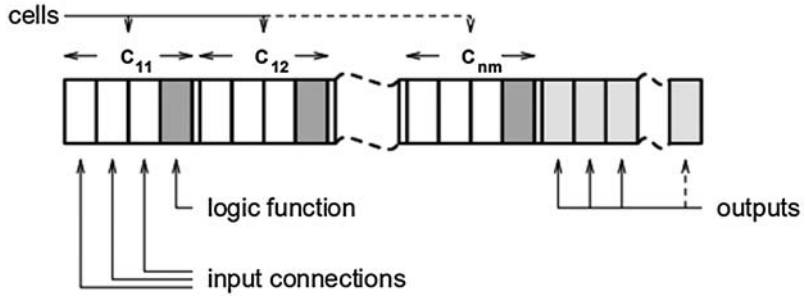


Figure 5: The genotype with respect to the rectangular array of logic cells

Xie et al. [27] proposed a two-phase EHW framework, which consists of an evolution phase based on an express tree genetic programming (ETGP) algorithm and an optimization phase based on a mining frequency digital circuit (MFDC) algorithm. It is believed that the application of GEP to EHW can rapidly generate the optimal solution of an evolutionary circuit.

2.3 Introduction to GEP

Gene expression programming (GEP) [28] follows the basic steps of evolutionary computation (EC), which combines the advantages of genetic algorithms and genetic programming. Genetic material consists of two kinds of symbols: terminators and functions. A gene consists of a linear, fixed-length string of symbols and the code for expression trees, in different sizes and shapes. A chromosome can be composed of a single gene or multiple ones, decoded to map as a candidate solution for problem response.

F is the set of functions, and T is the set of terminals. GEP's gene contains a head and a tail. The head contains symbols that represent both F and T, whereas the tail only contains T. For each problem, the length of the head h is chosen, and the length of the tail e is a function of h. The number of arguments of the function with more arguments n (also called maximum order) is evaluated by expression 1:

$$e = h \times (n - 1) + 1. \tag{1}$$

For example, consider a gene for the set of functions $F = \{+, -, *, /, Q\}$ and the set of terminals $T = \{a, b\}$. In this case, $n = 2$, and if $h = 10$, then $e = 11$, and the length of the gene is 21. String 1 describes such a gene (the tail is shown in bold).

$+Q-/b*abQb$***abababbaaab*** String 1

Fig. 6 shows the expression tree (ET) decoded from String 1.

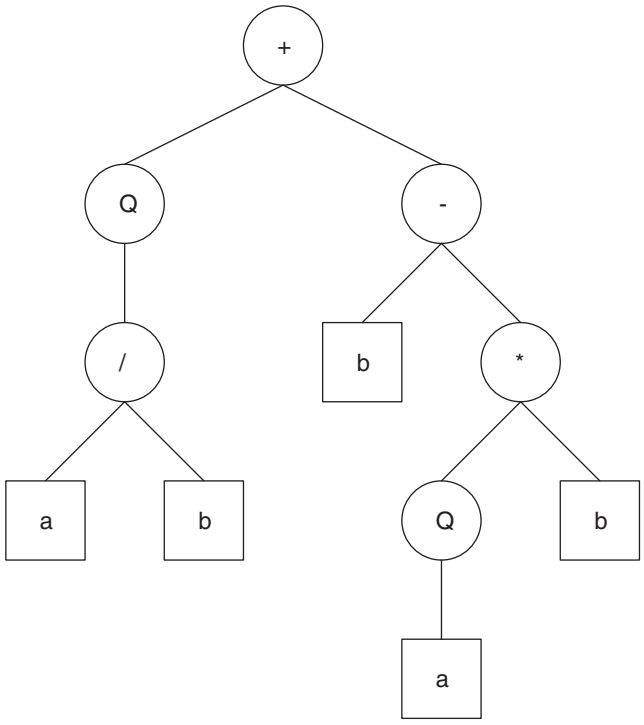


Figure 6: The expression tree decoded from String 1

The algebraic expression represented by Fig. 6 is

$$\sqrt{a/b}+b-\sqrt{a}*b. \tag{2}$$

The redundant symbols in the tail are discarded directly. Then GEP can use fixed-length encoding to express different sizes and shapes of expression trees.

3 Using GEP to Represent Circuit

GEP has performed well in mining association rules, clustering, classification rules, time-series predictions, and sunspot predictions [29–32].

As the structure shows, GEP performs well in resolving tree-structured problems. In other words, if one circuit can be represented as a tree-shaped structure with n leaf nodes, it can be described directly with GEP. The logic circuit of the 6-in/1-out in Fig. 7a can be easily represented in an ET in Fig. 7b, which replaces the logic gate function with the logic symbol. The effective gene in GEP is shown as String 2.

And, *And*, *And*, *A*, *Not*, *And*, *And*, *B*, *C*, *Not*, *E*, *F*, *D* String 2

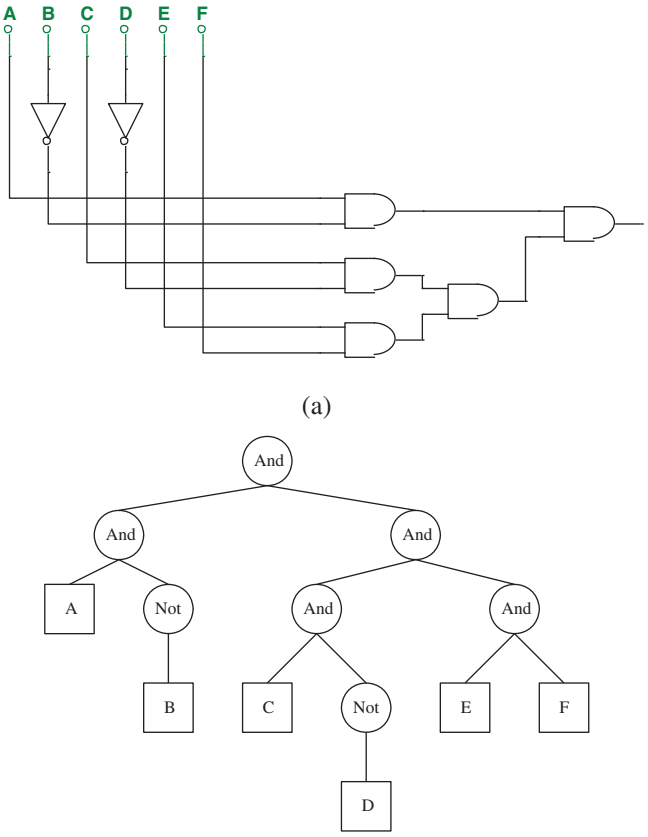


Figure 7: (a) The 6-in/1-out circuit represented by String 2 and (b) ET of the circuit shows in (a)

In GEP, uppercase letters represent operators and lowercase letters represent terminators. In this paper, however, we use the name string of the logic gate to represent logic gate, and uppercase letters to represent input parameters. The purpose of this modification is to facilitate the use of circuit simulation software.

However, if only logical values are used to represent the circuit, there would be too many isomorphic situations, For example, the circuit and its ET in Fig. 7 can be replaced with the circuit and the ET in Fig. 8.

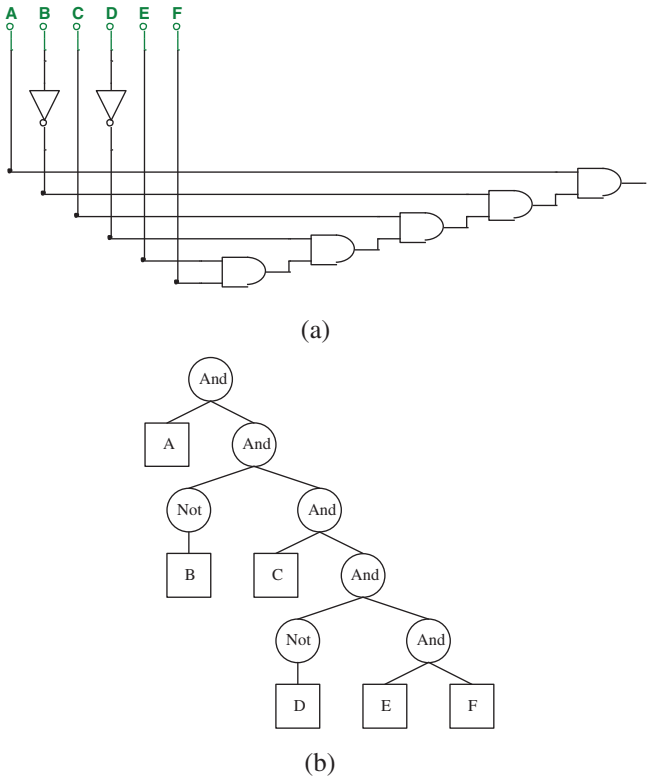


Figure 8: (a) A circuit isomorphic to Fig. 7 and (b) ET of the circuit shows in (a)

String 3 describes the corresponding effective gene of Fig. 8.

And, *A*, *And*, *Not*, *And*, *B*, *C*, *And*, *Not*, *And*, *D*, *E*, *F* String 3

The two circuits are fully equivalent in terms of logic values, and both are shown in expression (3):

$$Y = AB'CD'EF. \tag{3}$$

This example is only for the logic value of the circuit. A similar situation exists in terms of bypass information detection, and many different circuit structures could result from any single bypass information, which is referred to as *Isomorphic* in this paper.

Therefore, if a circuit is described only by logical values or a certain kind of bypass information, the circuit structure cannot be confirmed because of so much isomorphism. This paper

proposes an algorithm called mixed-feature GEP (MF-GEP), which represents multiple circuit features by using the same structure in GEP. This algorithm can reduce the number of isomorphic circuits and can be used to detect hardware Trojans.

4 Formal Definition of Trojan Circuit

For a circuit R, we provide the following symbols:

Let $A = \{A_k \mid k = 1,\ldots, c\}$ be the set of all c feature values that can be tested on R.

$i_k = [i_1, i_2,\ldots, i_n]$ is the input value vector corresponding to feature value A_k, and n is the number of input parameters of the circuit.

$o_k = [o_1, o_2,\ldots, o_m]$ is the output value vector of input vector i_k corresponding to circuit R, denoted as $o_k = R(i_k)$, and m is the number of output values of R.

$I_k = \{i_k \mid k = 1, 2,\ldots, d\}$ is the domain of i_k, $O_k = \{o_k \mid k = 1, 2,\ldots, d\}$ is the value domain of I_k.

We provide the following definitions:

Def 1

For two circuits $R \neq S$, if there is a feature value set $A_k \in A$:

For $\exists\, i_k \in I_k$, there is $R(i_k) = S(i_k)$.

Then we call circuits R and S the same-valued isomeric circuits (**SVIC**) on feature A_k for input vector i_k.

Def 2

For $\forall\, i_k \in I_k$, there is $R(i_k) = S(i_k)$.

Then we call circuits R and S the full same-valued isomeric circuits (**FSVIC**) on feature A_k.

Def 3

For $\exists\, I_k \subset I_k$, there are

$\forall\, i'_k \in I'_k, R\left(i'_k\right) = S\left(i'_k\right)$, and $\forall\, i_k \in I_k - I'_k, R(i_k) \neq S(i_k)$.

Then we call circuits R and S the part same-valued isomeric circuits (**PSVIC**) on feature A_k.

Although Strings 2 and 3 represent two different circuits, they are exactly the same in the logical value test, and both represent the circuits in expression (2). According to Def 2, Strings 2 and 3 are FSVIC in logical values. However, it is always possible to find a certain feature (such as current) that makes Strings 2 and 3 PSVIC. Therefore, Strings 2 and 3 are considered as PSVIC.

A formal definition of a Trojan circuit is as follows:

Def 4: Definition of Trojan Circuit

If the circuits R and S meet the following conditions, we call circuit R a **Trojan circuit** with a Trojan added to circuit S if:

(1) R claims it is circuit S.
(2) In fact, R is one of the PSVICs of S.

For the input domain I:

(1) There are two sets of X and Y, $X \cap Y = \Phi$ and $X \cup Y = I$.
(2) For $\forall\, i \in X$, there is $R(i) = S(i)$, and a Trojan will not be triggered.

(3) For $\forall i \in Y$, there is $R(i) \neq S(i)$, and a Trojan will be triggered.

The smaller the scale of Y, the more difficult it is to detect the difference between R and S, and the deeper the Trojan is.

5 Mixed-Feature GEP

In order to make set Y in Def 4 easier to find, multiple features of the circuit were detected at the same time. Although the detection results of each feature value can build many FSVICs, different FSVICs of different features will form some PSVICs, of which the same part is possible in the real circuit.

Fig. 9 shows a Not gate designed by a triode. It is used as an example to illustrate the concept of a mixed-feature GEP algorithm.

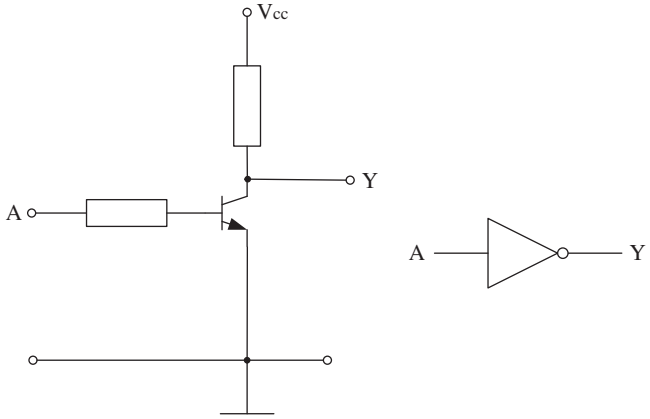


Figure 9: A Not gate designed by triode

- Feature Value 1:

In digital logic terms, the description of this circuit is:

$$Y = \ !A \tag{4}$$

- Feature Value 2:

In terms of voltage, the description of this circuit is:

$$V_y = V_A < V_{SH} \ ? \ V_{CC} : V_{CES}, \tag{5}$$

where V_{SH} represents the higher threshold and lower potential limit of "1" in the circuit, and V_{CES} represents a saturation voltage drop of the triode.

- Feature Value 3:

In terms of current, the description of this circuit is:

$$C_y = N \cdot C_A, \tag{6}$$

where N is the magnification of the current.

In addition, there are a variety of other bypass information detection items, such as delay and spectrum.

With any of the above three feature values, the circuit structure cannot be determined. However, it is possible to determine the circuit structure if the three features are combined.

In GEP, an operator represents only one computation. A GEP individual only represents a description of one feature value and obtains an unlimited number of isomorphic circuits. It is impossible to determine the actual circuit structure.

The MF-GEP proposed in this paper combines the test results of multiple feature values on a circuit into a single function representation, which integrates them into a composite function, i.e., a function that represents multiple calculations at the same time. Then it evolves representations close to the original circuit by using GEP's functional evolutionary ability.

Detection values are included in a function. The input comes from multiple feature values and the output is a vector, like the Not-gate circuit. In GEP's ET, it is still represented by "*Not*," but the implication becomes the calculation of the vector below:

$$Not\,(A_1, A_2, \ldots, A_n) = [F_1(A_1), F_2(A_2), \ldots, F_n(A_n)], \tag{7}$$

where $A_k\ (k = 1,\ldots, n)$ is the input value of a type of feature detection, and $F_k\ (A_k)\ (k = 1,\ldots, n)$ is the result of this feature value detection corresponding to input value A_k.

However, in an electronic component, each feature value should have a different importance. So, each member of this vector should be multiplied by a weight:

$$Not(A_1, A_2, \ldots, A_n) = [c_1, c_2, \ldots, c_n] \cdot [F_1(A_1), F_2(A_2), \ldots, F_n(A_n)] \tag{8}$$

For normalization, it can be specified as

$$\sum_{i=1}^{n} c_i = 1. \tag{9}$$

For example, corresponding to this Not gate circuit, the symbol Not indicates the following meaning:

$$Not(A, V_A, C_A) = [0.4, 0.3, 0.3] \cdot [!\,A, V_A < V_{SH}?V_{CC} : V_{CES}, N \cdot C_A], \tag{10}$$

where A represents the logical value (1 or 0) by the voltage entered at point A, V_A represents the input voltage of point A, and C_A represents the input current of point A. *[0.4, 0.3, 0.3]* is the weight vector.

Therefore, when using GEP evolution, a single symbol **Not** also represents multiple feature values that are not associated with each other.

6 Experiments

6.1 Experimental Settings

In this paper, four groups of experiments (Experiments 1–4) were designed initially. Later, in order to verify the new problems in Experiments 1–4, Experiment 5 was added. As comparison experiments, most of the parameters and fitness functions are exactly the same.

6.1.1 Parameters

The circuit has multiple inputs and only one output. Experiment 1 used one feature, Experiments 2 and 3 used two features, Experiment 4 used three features, and Experiment 5 used two

features. The three features are logical value, voltage value and current value. The GEP parameters of all the experiments are exactly the same. Tab. 1 lists the GEP parameters.

Table 1: Experiment parameters

Parameter	Value
Fitness	=1
Selection mode	Tournament, size = 3
Population size	10000
Head length	20
Tail length	21
Chromosome length	1
Mutation rate	0.05
Insert rate	0.1
Root insert rate	0.01
One-point cross rate	0.1
Two-point recombination rate	0.1
Number of inputs	4
Number of outputs	1
Function set	Not, And, Or

6.1.2 Fitness Functions

The feature data includes logical data, voltage data, and current data, and each has its respective fitness function:

(1) Logical fitness function:

$$F_1 = 1 - \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N}, \tag{11}$$

where N is the amount of test data, $\hat{y}_i$ is the logical value calculated from the expression represented by the individual, and y_i is the actual logical value in the test data. The range of F_1 is now discussed as follows:

a) If all $\hat{y}_i$ are exactly equal to y_i, there is

$$\frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N} = 0. \tag{12}$$

Then

$$F_1 = 1 - \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N} = 1. \tag{13}$$

Then the maximum value of F_1 is 1.

b) Because of the logic value, the worst case is that every $\hat{y}_i$ is only against y_i, then every $|\hat{y}_i - y_i|$ $(i = 1,\ldots, N)$ equals 1. There is:

$$\sum_{i=1}^{N} \left|\hat{y}_i - y_i\right| = N. \tag{14}$$

Then

$$F_1 = 1 - \frac{\sum_{i=1}^{N} \left|\hat{y}_i - y_i\right|}{N} = 0. \tag{15}$$

Then the minimum value of F_1 is 0.

c) Suppose there are two GEP individuals, G_1 and G_2. For the same set of test data,

For G_1, there are m $\hat{y}_i$s equal to $y_{i..}$

For G_2, there are n $\hat{y}_i$s equal to $y_{i.}$ $(0 < m < n < N)$.

For G_1, a rearrangement of $(\hat{y}_i, y_i)$ leads to $\hat{y}_i = y_i$ when $i = 1,\ldots, m$, then

$$\begin{aligned} F_1(G_1) &= 1 - \frac{\sum_{i=1}^{N} \left|\hat{y}_i - y_i\right|}{N}. \\ &= 1 - \frac{\sum_{i=1}^{m} \left|\hat{y}_i - y_i\right| + \sum_{i=m+1}^{N} \left|\hat{y}_i - y_i\right|}{N}, \\ &= 1 - \frac{N-m}{N}, \\ &= \frac{m}{N}. \end{aligned} \tag{16}$$

For G_2, a rearrangement of $(\hat{y}_i, y_i)$ leads to $\hat{y}_i = y_i$ when $i = 1, \ldots, n$, then

$$F_1(G_2) = \frac{n}{N}. \tag{17}$$

Because $m < n$, so $F_1(G_1) < F_1(G_2)$.

Therefore, the conclusion can be drawn as follows:

(i) The range of F_1 is in $[0, 1]$.
(ii) With the improvement of the matching degree between the calculated value $\hat{y}_i$ and the tested value y_i, the value of F_1 increases monotonically.

(2) Voltage fitness function

$$F_2 = 1 - \frac{\sum_{i=1}^{N} \left|\hat{y}_i - y_i\right|}{N(V_{CC} - V_{DD})} \tag{18}$$

N is the amount of test data. $\hat{y}_i$ is the voltage value calculated from the expression represented by the individual. y_i is the actual voltage value in the test data. $(V_{CC} - V_{DD})$ is a fixed value. If $V_{mx} = (V_{CC} - V_{DD})$, then F_2 can be defined as

$$F_2 = 1 - \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N \cdot V_{mx}} \tag{19}$$

The range of each $|\widehat{y_i} - y_i|$ is $[0, V_{mx}]$. The function forms of F_2 are discussed as follows:

a) If all $\hat{y}_i$ are exactly equal to y_i, there is

$$\frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N} = 0. \tag{20}$$

Then

$$F_2 = 1 - \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N \cdot V_{mx}} = 1. \tag{21}$$

Then the maximum value of F_2 is 1.

b) The worst case is that every $\hat{y}_i$ has the max distant to the y_i, which means that every $|\hat{y}_i - y_i| = V_{mx}$ $(i = 1, \ldots, N)$. Then

$$\begin{aligned} F_2 &= 1 - \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N \cdot V_{mx}}, \\ &= 1 - \frac{N \cdot V_{mx}}{N \cdot V_{mx}} = 0. \end{aligned} \tag{22}$$

Then the minimum value of F_2 is 0.

c) Suppose there are k $(0 \le k < N)$ $\hat{y}_i$s equal to y_i in a GEP individual G. A rearrangement of $(\hat{y}_i, y_i)$ leads to $\hat{y}_i = y_i$ when $i = k + 1, \ldots, N$, then

$$\begin{aligned} F_2(G) &= 1 - \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N \cdot V_{mx}}, \\ &= 1 - \frac{\sum_{i=1}^{N-k} |\hat{y}_i - y_i| + \sum_{i=k+1}^{N} |\hat{y}_i - y_i|}{N \cdot V_{mx}}, \\ &= 1 - \frac{\sum_{i=1}^{N-k} |\hat{y}_i - y_i|}{N \cdot V_{mx}}, \\ &(|\hat{y}_i - y_i| \in (0, V_{mx}], \quad i = 1, \ldots, N-k). \end{aligned} \tag{23}$$

Let us define E_A $(k, \hat{y}_i, y_i)$ *(Absolute Error for* $(k, \hat{y}_i, y_i)$*)*.

$$E_A(k, \hat{y}_i, y_i) = \sum_{i=1}^{N-k} |\hat{y}_i - y_i| \tag{24}$$

Then,

$$F_2(G) = 1 - \frac{E_A}{N \cdot V_{mx}}. \tag{25}$$

Because $|\hat{y}_i - y_i| \in (0, V_{mx}]$, so when $|\hat{y}_i - y_i| \in (0, V_{mx}]$ is input into Eq. (23), the following may result:

$$|\hat{y}_i - y_i| \overset{lim}{\to} 0 \tag{26}$$

Then

$$E_A \overset{lim}{\to} 0, \quad F_2(G) \overset{lim}{\to} 1. \tag{27}$$

When

$$|\hat{y}_i - y_i| = V_{mx}, \tag{28}$$

Then

$$E_A = (N - k) \cdot V_{mx} \tag{29}$$

$$F_2(G) = 1 - \frac{(N-k) \cdot V_{mx}}{N \cdot V_{mx}} = \frac{k}{N}. \tag{30}$$

d) Suppose there are two GEP individuals, G_1 and G_2.

For G_1, $k = m$, and for G_2, $k = n$, $(0 < m < n < N)$. The function forms of $F_2(G_1)$ *and* $F_2(G_2)$ are shown in Fig. 10.

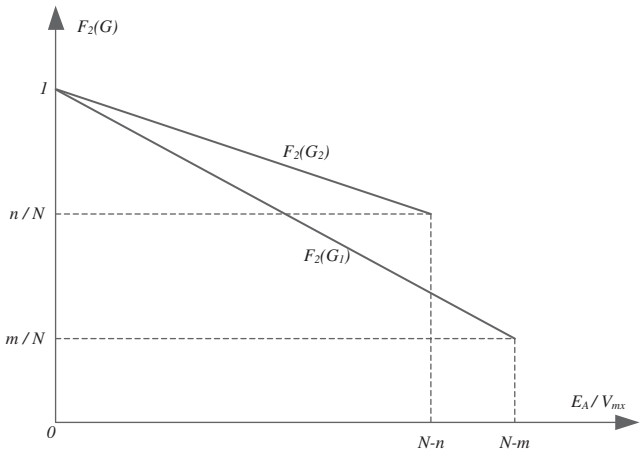


Figure 10: The function forms of $F_2(G_1)$ *and* $F_2(G_2)$

Therefore, we can draw the conclusion that:

(i) The range of F_2 is in $[0, 1]$.
(ii) If individual G_2 is more similar to the destination circuit than G_1, there is $F_2(G_2) > F_2(G_1)$.

(3) Current fitness function

$$F_3 = 1 - \frac{SSE}{SST},$$

$$SSE = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2,$$

$$SST = \sum_{i=1}^{N} (y_i - \overline{y}_i)^2. \tag{31}$$

N is the amount of test data. $\hat{y}_i$ is the current value calculated from the expression represented by the individual. y_i is the actual current value in the test data. $\overline{y}_i$ is the average of y. SSE is residual sum of squares. SST is sum of squares of deviations. $F3$ is the square of the multiple correlation coefficient in statistics, whose value is in [0, 1].

(4) Individual's fitness

According to the previous description of the algorithm, the individual's fitness should be a combination of the three fitness functions, and therefore the individual's fitness is defined as:

$$F = [c_1 \quad c_2 \quad c_3] \cdot \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}. \tag{32}$$

c_i is the weight of corresponding F_i in final fitness. The sum of c_i is 1:

$$1 = \sum_{i=1}^{3} c_i. \tag{33}$$

If we set the weight vector

$$C = [c_1 \quad c_2 \quad c_3], \tag{34}$$

then

$$F = C \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}. \tag{35}$$

6.2 Experimental Results

Fig. 11 shows a circuit that has no Trojan.

Its Boolean expression is:

$$Y = A + BC + BD. \tag{36}$$

The circuit makes the computation:

$$Y = \begin{cases} 0, & (ABCD)_2 < 5 \\ 1, & else \end{cases} \tag{37}$$

After several logic gates are added to the circuit of Fig. 11, it becomes a circuit with Trojan (Fig. 12). In the new circuit, Y will also get a value of 0 when $(ABCD)_2 = 7$:

$$Y = \begin{cases} 0, & (ABCD)_2 < 5 \ or \ (ABCD)_2 = 7 \\ 1, & else \end{cases}. \tag{38}$$

Its Boolean expression becomes:

$$Y = A + BCD' + BC'D. \tag{39}$$

String 4 describes the corresponding effective gene of Fig. 12.

Or, *A*, *Or*, *And*, *And*, *B*, *And*, *B*, *And*, *Not*, *D*, *C*, *Not*, *C*, *D* String 4

Only a portion of the values can be tested if there are too many pins. The input value used to activate the Trojan $(ABCD)_2 = (0111)_2$ may be missed at this time. In the following experiment, the input value $(0111)_2$ will not be provided, and the output will be determined by the evolved circuit.

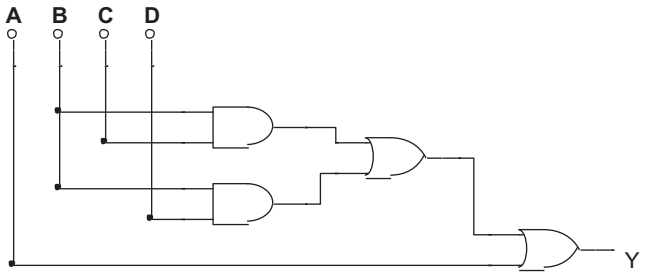


Figure 11: A circuit has no Trojan

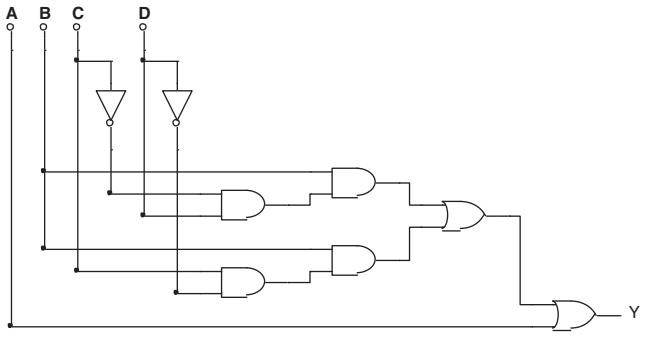


Figure 12: The circuit formed by injecting Trojan into the circuit shown in Fig. 11

Four groups of experiments were designed using different provided values and weight vectors. Considering that the logic values are first required to be correct in the circuit, the voltage and current values must be based on the correct logic values in order to make sense. The individual's fitness is a combination of several data, in which the proportion of logical value is larger.

6.2.1 Experiment 1

Tab. 2 lists the setting of Experiment 1. None of the 100 exercises were able to discover the Trojan circuit if only logical values were provided. Fig. 13 shows some typical results.

Table 2: Setting of Experiment 1

Parameter	Value
Logic gate	And, Or, Not
Values provided	Logic values
Weight vector	C = [1, 0, 0]
Exercise count	100
Trojans discovered	0

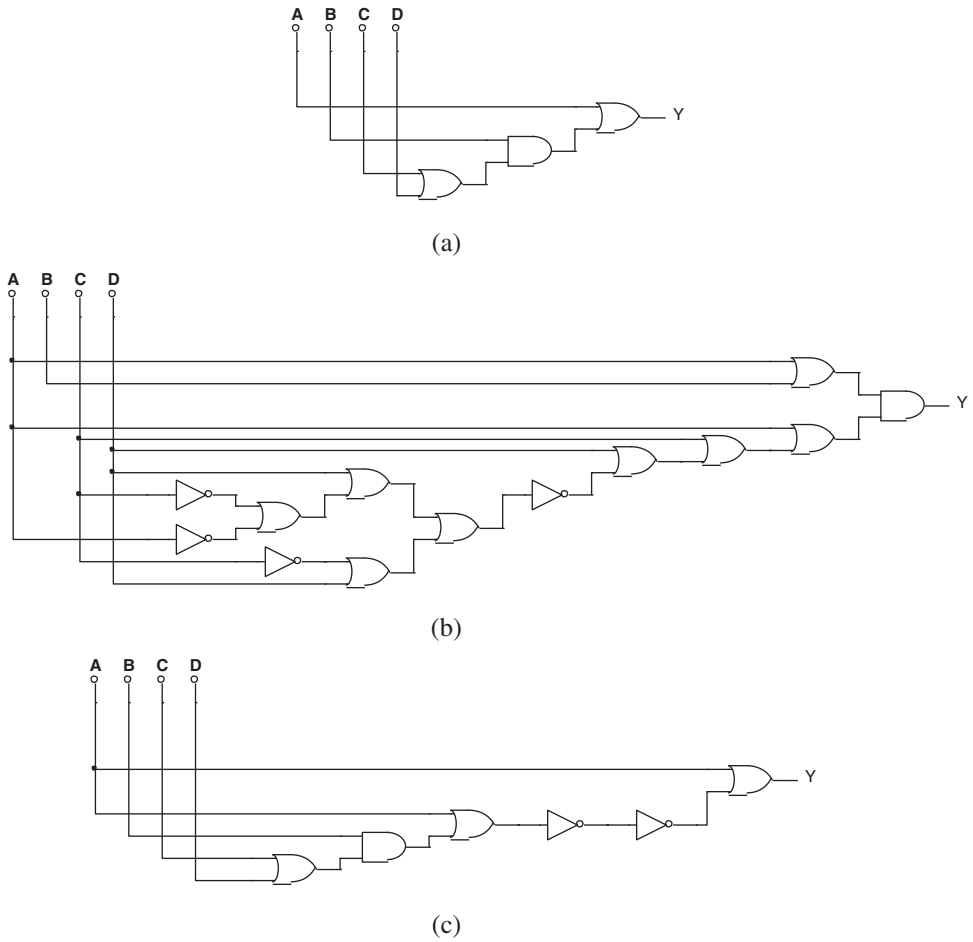


Figure 13: (a) One of the circuits evolved by mixed-feature GEP in Experiment 1, (b) Another circuit evolved by mixed-feature GEP in Experiment 1, and (c) The 3rd circuit evolved by mixed-feature GEP in Experiment 1

The simplified Boolean expression of all the circuits above is:

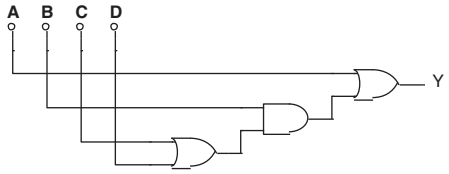
$$Y = A + BC + BD \tag{40}$$

6.2.2 Experiment 2

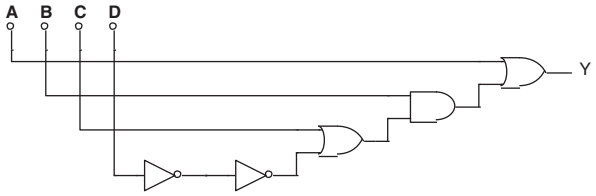
Tab. 3 lists the setting of Experiment 2. None of the 100 exercises were able to discover the Trojan circuit if both logical values and voltage values were provided. Fig. 14 shows some typical results.

Table 3: Setting of Experiment 2

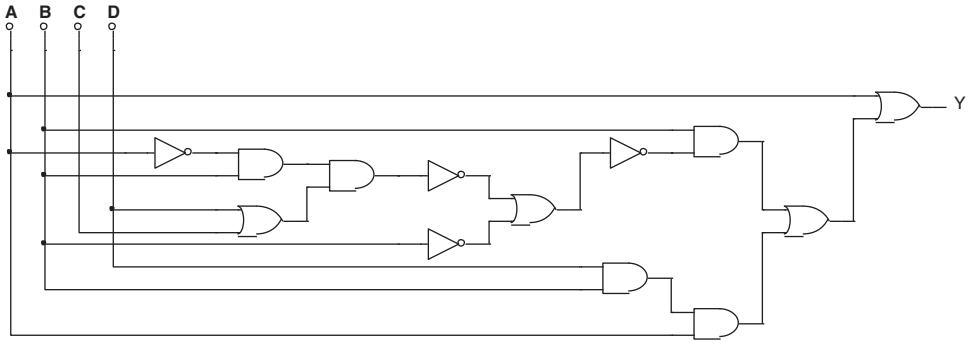
Parameter	Value
Logic gate	And, Or, Not
Values provided	Logic values, voltage values
Weight vector	C = [0.8, 0.2, 0]
Exercise count	100
Trojans discovered	0



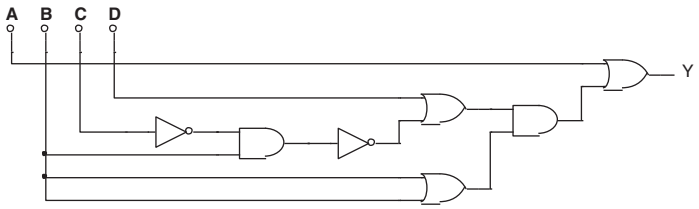
(a)



(b)



(c)



(d)

Figure 14: (a) One of the circuits evolved by mixed-feature GEP in Experiment 2, (b) Another circuit evolved by mixed-feature GEP in Experiment 2, (c) The 3rd circuit evolved by mixed-feature GEP in Experiment 2, and (d) The 4th circuit evolved by mixed-feature GEP in Experiment 2

The simplified Boolean expression of all the circuits above is:

$$Y = A + BC + BD. \tag{41}$$

A Trojan still cannot be found, although both the logical value and the voltage value were provided. The reason is that in digital circuits, the logic value is expressed in the form of voltage values. For example, a voltage value less than $3V$ is considered to be 0, and one that is greater than $3V$ is considered to be 1. Therefore, the choice of logical value and voltage value as feature values on this issue is as same as if only the logical value were provided.

6.2.3 Experiment 3

Tab. 4 lists the setting of Experiment 3. During the 100 exercises, the Trojan circuit was discovered 72 times when logical values and voltage values were provided. Fig. 15 shows some typical results.

Table 4: Setting of Experiment 3

Parameter	Value
Logic gate	And, Or, Not
Values provided	Logic values, current values
Weight vector	C = [0.8, 0, 0.2]
Exercise count	100
Trojans discovered	72

The simplified Boolean expression of both Figs. 15a and 15b is:

$$Y = A + BCD' + BC'D. \tag{42}$$

The equivalent circuit has been discovered, although the original circuit was hidden.

Fig. 15c is a failed result in finding the right circuit. Its Boolean expression is:

$$Y = A + BC + BD. \tag{43}$$

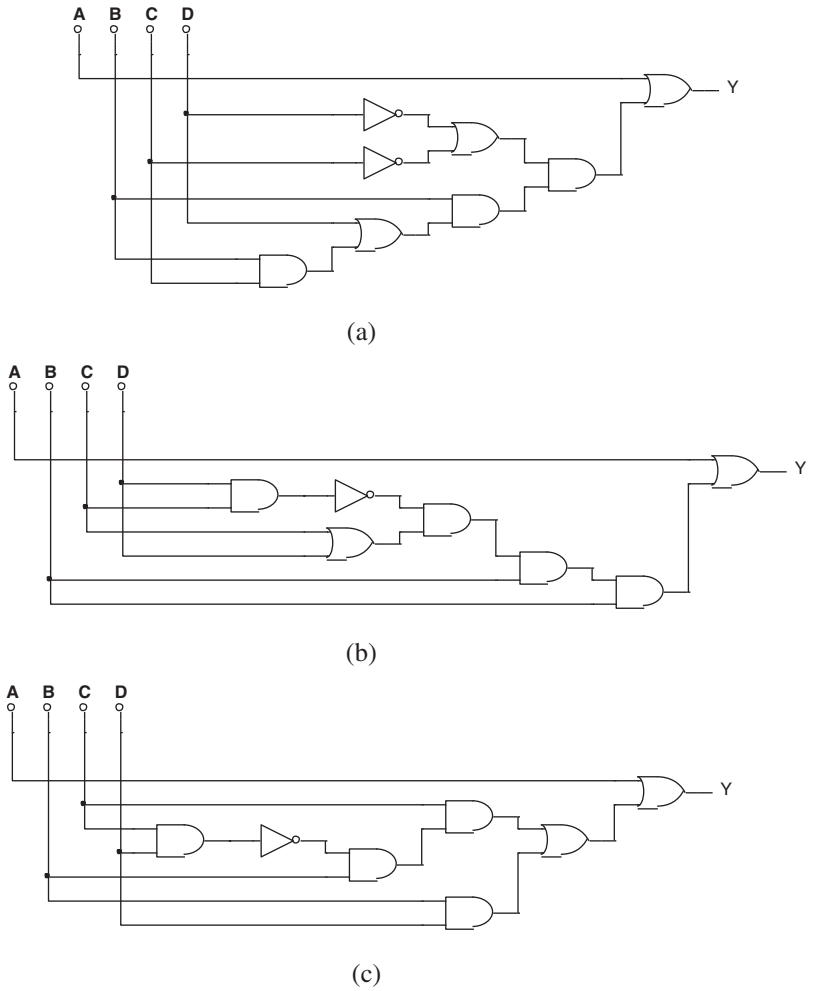


Figure 15: (a) One of the circuits evolved by mixed-feature GEP in Experiment 3, (b) Another circuit evolved by mixed-feature GEP in Experiment 3, and (c) The 3rd circuit evolved by mixed-feature GEP in Experiment 3

6.2.4 Experiment 4

Tab. 5 lists the setting of experiment 4. During the 100 exercises, the Trojan circuit was discovered 67 times when all three feature values were provided. Fig. 16 shows a different result.

Table 5: Setting of Experiment 4

Parameter	Value
Logic gate	And, Or, Not
Values provided	Logic values, voltage values current values
Weight vector	C = [0.6, 0.2, 0.2]
Exercise count	100
Trojans discovered	67

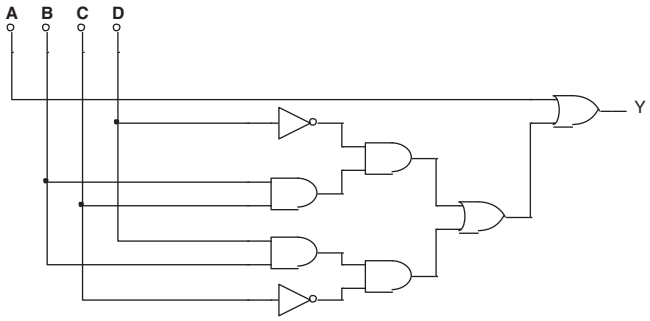


Figure 16: Typical circuit evolved by mixed-feature GEP in Experiment 4

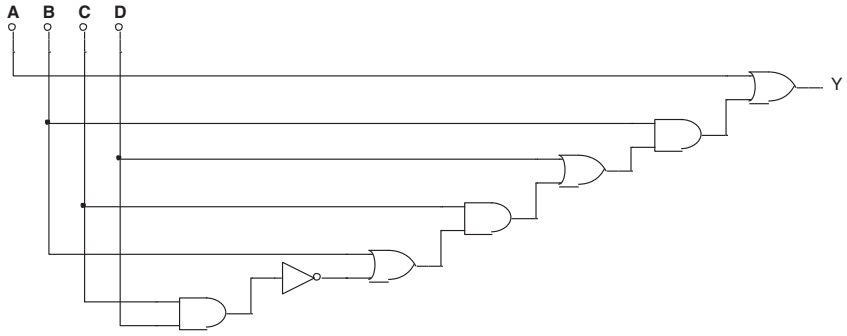
Its simplified Boolean expression is:

$$Y = A + BCD' + BC'D. \tag{44}$$

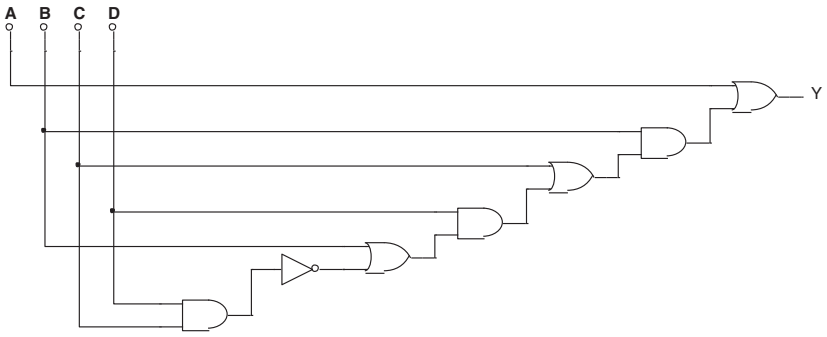
A circuit equivalent to the original circuit has been found. The equivalent circuit has been discovered even though the original circuit was hidden.

In this group of experiments, some wrong circuits were found by the failed evolution. As shown in Fig. 17, their Boolean expressions are:

$$Y = A + BC + BD. \tag{45}$$



(a)



(b)

Figure 17: (a) One of the false results in Experiment 4 and (b) Another false result in Experiment 4

Although three feature values were used in this group of experiments, the efficiency in discovering the Trojan was similar to that in Experiment 3, which used only two feature values. This is because the logical value itself is expressed in the form of voltage values and the three feature values are equal to the two feature values.

6.2.5 Experiment 5

As discussed in Experiments 2 and 4, in digital circuits, the logical value is expressed in the form of voltage value, e.g., a voltage value less than 3V is referred to as 0, while a voltage value 3V and above is considered to be 1. Therefore, the result of using the logical value as the feature value should be similar to that of the voltage value. To verify this hypothesis, Experiment 5 is designed as a control test to Experiment 3. Tab. 6 shows the parameters and results. It differs from Experiment 3 in the replacement of the logical value by the voltage value, and the modification of the fitness function page accordingly.

Table 6: Setting of Experiment 5

Parameter	Value
Logic gate	And, Or, Not
Values provided	Voltage values, current values
Weight vector	C = [0, 0.8, 0.2]
Exercise count	100
Trojans discovered	53

We can see that in 53 of the 100 experiments, results equivalent to Trojan circuits were obtained, but these circuits failed to make breakthrough findings in Experiments 3 and 4. However, several false circuits were found, as shown in Fig. 18. The Boolean expressions are:

$$Y = A + BC + BD. \tag{46}$$

Experiment 5 shows a lower probability in detecting Trojan circuits than Experiment 3. Nevertheless, the result is acceptable considering the randomness of evolutionary calculation. Results indicate that voltage data and logical value data in the mixed-feature GEP algorithm produce similar results. This verifies the hypothesis in Experiments 2 and 4 that two feature values are inter-replaceable in the algorithm if there is a simple correlation between them. Therefore, the maximization of variance between feature values helps to improve the effectiveness of the algorithm when the simple and direct correlation is uncertain.

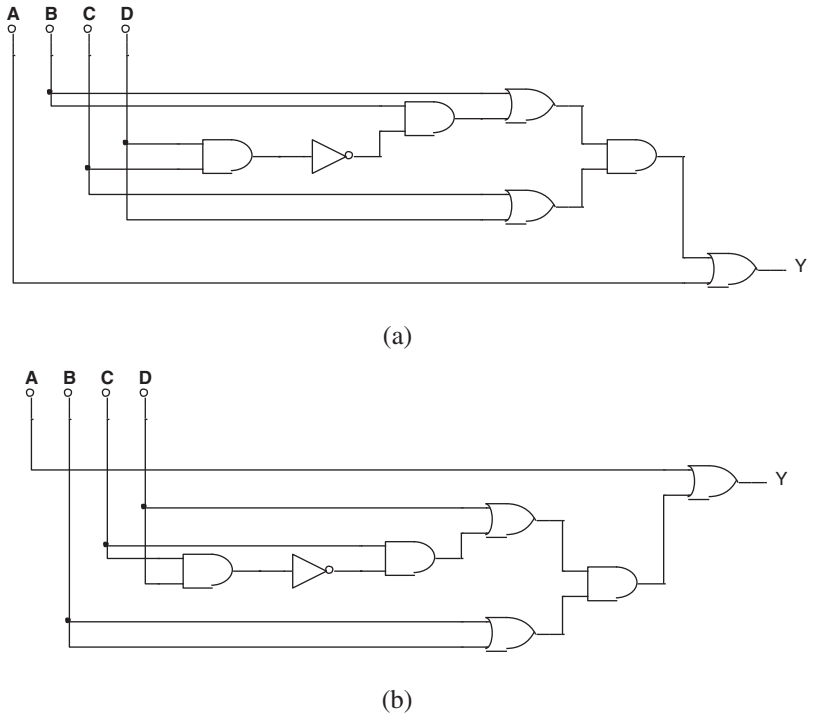


Figure 18: (a) One of the false results in Experiment 5 and (b) Another false result in Experiment 5

7 Conclusions

This paper proposes a mixed-feature GEP (MF-GEP) algorithm in which multiple feature values were fused into the same operator. There is a specific probability that Trojan circuits could be detected by MF-GEP, which automatically discovers the evolutionary power of mathematical formulas. The fewer features that are used, the higher the efficiency of the GEP evolution, but the conclusion is in wider disparity from the real circuit. At the same time, as the number of features used increases, the efficiency of GEP evolution decreases, but the conclusion drawn gets closer to the real circuit. However, if there is a direct conversion relationship between the multiple feature values used, these values can be considered as one and the accuracy of MF-GEP evolution will not be increased.

Funding Statement: This work was supported by the National Key Research and Development Program of China (Grant No. 2018YFB1502803).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Xiao, K., Forte, D., Jin, Y., Karri, R., Bhunia, S. et al. (2016). Hardware Trojans: Lessons learned after one decade of research. *ACM Transactions on Design Automation of Electronic Systems, 22(1),* 1–23. DOI 10.1145/2906147.
2. Antonopoulos, A., Kapatsori, C., Makris, Y. (2017). Trusted analog/mixed-signal/RF ICs: A survey and a perspective. *IEEE Design & Test, 34(6),* 63–76. DOI 10.1109/MDAT.2017.2728366.

3. Bhunia, S., Hsiao, M., Banga, M., Narasimhan, S. (2014). Hardware Trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE, 102(8),* 1229–1247. DOI 10.1109/JPROC.2014.2334493.
4. Lv, Y. Q., Zhou, Q., Cai, Y. C., Qu, G. (2014). Trusted integrated circuits: The problem and challenges. *Journal of Computer Science & Technology, 29(5),* 918–928. DOI 10.1007/s11390-014-1479-9.
5. Tehranipoor, M., Wang, C. (2012). *Introduction to hardware security and trust*. USA: Springer Publishing Company.
6. Wang, X., Tehranipoor, M., Plusquellic, J. (2008). Detecting malicious inclusions in secure hardware: Challenges and solutions. *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 15–19. Anaheim, CA, USA.
7. Tehranipoor, M., Koushanfar, F. (2010). A survey of hardware Trojan taxonomy and detection. *IEEE Design and Test of Computers, 27(1),* 10–25. DOI 10.1109/MDT.2010.7.
8. Zhao, J. F., Shi, G. (2017). A survey on the studies of hardware Trojan. *Journal of Cyber Security, 2(1),* 74–90. DOI 10.19363/j.cnki.cn10-1380/tn.2017.01.006.
9. Bhasin, S., Regazzoni, F. (2015). A survey on hardware Trojan detection techniques. *Proceedings of the 2015 IEEE International Symposium on Circuits and Systems*, pp. 2021–2024. Lisbon, Portugal.
10. Courbon, F., Loubet-Moundi, P., Fournier, J. J. A., Tria, A. (2015). SEMBA: A SEM based acquisition technique for fast invasive hardware Trojan detection. *Proceedings of the 2015 European Conference on Circuit Theory and Design*, pp. 1–4. Trondheim, Norway.
11. Bao, C. X., Forte, D., Srivastava, A. (2014). On the application of one-class SVM to reverse engineering-based hardware Trojan detection. *Proceedings of the 15th International Symposium on Quality Electronic Design*, pp. 47–54. Santa Clara, CA, USA.
12. Bao, C. X., Forte, D., Srivastava, A. (2015). On reverse engineering-based hardware Trojan detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35(1),* 49–57. DOI 10.1109/TCAD.2015.2488495.
13. Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P., Sunar, B. (2007). Trojan detection using IC fingerprinting. *Proceedings of the 2007 IEEE Symposium on Security and Piracy*, pp. 296–310. Berkeley, CA, USA.
14. Xiao, K., Zhang, X., Tehranipoor, M. (2013). A clock sweeping technique for detecting hardware Trojans impacting circuits delay. *IEEE Design & Test, 30(2),* 26–34. DOI 10.1109/MDAT.2013.2249555.
15. Aarestad, J., Acharyya, D., Rad, R., Plusquellic, J. (2010). Detecting Trojans through leakage current analysis using multiple supply pad I_{DDQ}s. *IEEE Transactions on Information Forensics and Security, 5(4),* 893–904. DOI 10.1109/TIFS.2010.2061228.
16. Nowroz, A. N., Hu, K., Koushanfar, F., Reda, S. (2014). Novel techniques for high-sensitivity hardware Trojan detection using thermal and power maps. *IEEE Transactions on Computer—Aided Design of Integrated Circuits and Systems, 33(12),* 1792–1805. DOI 10.1109/TCAD.2014.2354293.
17. Zhou, B. Y., Adato, R., Zangeneh, M., Yang, T. Y., Uyar, A. et al. (2015). Detecting hardware Trojans using backside optical imaging of embedded watermarks. *Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference*, pp. 1–6. San Francisco, CA, USA.
18. He, J., Zhao, Y., Guo, X., Jin, Y. (2017). Hardware Trojan detection through chip-free electromagnetic side-channel statistical analysis. *IEEE Transactions on Very Large Scale Integration Systems, 25(10),* 2939–2948. DOI 10.1109/TVLSI.2017.2727985.
19. Jacob, N., Merli, D., Heyszl, J., Sigl, G. (2014). Hardware trojans: Current challenges and approaches. *IET Computers & Digital Techniques, 8(6),* 264–273. DOI 10.1049/iet-cdt.2014.0039.
20. Higuchi, T., Murakawa, M., Iwata, M., Kajitani, I., Liu, W. X. et al. (1997). Evolvable hardware at function level. *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pp. 187–192. Indianapolis, IN, USA.
21. Higuchi, T., Iwata, M., Kajitani, I., Iba, H., Hirao, Y. et al. (1996). Evolvable hardware and its application to pattern recognition and fault-tolerant systems. *Towards evolvable hardware,* pp. 118–135. Berlin: Springer Verlag.

22. Vassilev, V., Job, D., Miller, J. (2000). Towards the automatic design of more efficient digital circuits. *Proceedings of the 2nd NASA/DOD Workshop on Evolvable Hardware*, pp. 151–160. Palo Alto, CA, USA.
23. Timothy, G. W., Peter, J. B. (2002). Towards development in evolvable hardware. *Proceedings of the 3rd NASA/DOD Workshop on Evolvable Hardware*, pp. 241–250, Pasadena.
24. Erbo, M., Rossi, R., Liberali, V., Tettamanzi, A. G. B. (2001). Digital filter design through simulated evolution. *Proceedings of the European Conference on Circuit Theory and Design*, pp. 389–393. Espoo, Finland.
25. Hemmi, H., Mizoguchi, J., Shimohara, K. (1994). Development and evolution of hardware behaviors. *Towards Evolvable Hardware*, pp. 250–265. Berlin: Springer Verlag.
26. Hounsell, B., Arslan, T. (2000). A novel evolvable hardware framework for the evolution of high performance digital circuits. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 525–529. Las Vegas, Nevada, USA.
27. Xie, F. J., Tang, C. J., Yuan, C. A., Zuo, J., Cheng, A. L. (2005). An algorithm of evolution and optimization for evolvable hardware. *Journal of Computer—Aided Design and Computer Graphics, 17(17),* 1415–1420. DOI 10.3321/j.issn:1003-9775.2005.07.006.
28. Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems, 13(2),* 87–129. DOI 10.1007/3-540-32849-1.
29. Ferreira, C. (2002). *Gene expression programming: Mathematical modeling by an artificial intelligence*. USA: Springer Verlag.
30. Zuo, J., Tang, C. J., Zhang, T. Q. (2002). Mining predicate association rule by gene expression programming. *Proceedings of the 3rd International Conference for Web-Age Information Management*, pp. 281–294. Beijing, China.
31. Zuo, J., Tang, C. J., Li, C., Yuan, C. A., Chen, A. L. (2004). Time series prediction based on gene expression programming. *Proceedings of the 5th International Conference for Web-Age Information Management*, pp. 55–64. Dalian, China.
32. Zhou, C., Xiao, W., Tirpak, T. M., Nelson, P. C. (2003). Evolution accurate and compact classification rules with gene expression programming. *IEEE Transaction on Evolutionary Computation, 7(6),* 519–531. DOI 10.1109/TEVC.2003.819261.