Test Pattern generation for detection of Hardware Trojans based on improved Genetic Algorithm

Sandip Chakraborty
Dept of CSE
NIT Durgapur
sandipch240@gmail.com

Archisman Ghosh
Dept of CSE
NIT Durgapur
archiribhu@gmail.com

Anindan Mondal
Dept of CSE
NIT Durgapur
anindanmondal14@gmail.com

Bibhash Sen
Dept of CSE
NIT Durgapur
bibhash.sen@cse.nitdgp.ac.in

***Abstract*—Hardware Trojans (HT) are minuscule circuits embedded by an adversary for malicious purposes. Such circuits posses stealthy nature and can cause disruption upon activation. To detect the presence of such circuits, appropriate test vectors need to be applied. In this regard, the genetic algorithm (GA) seems to be the most promising technique due to its exploration capability. However, like most of the existing techniques, GA also suffers from exploring the huge search space. In this article a GA based methodology is proposed incorporating the information about potential inputs into it. Experimental results analysis signifies that the identification of the relevant inputs for GA provides an optimal solution. The significance of proposed methodology is endorsed by applying the proposed GA technique on different ISCAS '85 benchmark circuits. A noteworthy improvement on run time is observed while simultaneously providing improved test set quality than the state-of-the art technique.**

***Index Terms*—Hardware Trojan, Genetic Algorithm, SCOAP measurements, Primary Inputs**

I. Introduction

A hardware Trojan (HT) is a small piece of malicious hardware camouflaged inside a standard circuit. Such circuits are activated by a scarce input combination, and provide harmful alterations to the actual operation. Due to the high cost and short manufacturing time, most integrated circuits (IC) design houses do not have their own fabrication facility, hence the manufacturing of ICs is entirely dependent on offshore fabrication facilities [1]. These constraints of an IC design require frequent usage of hardware intellectual property (IP) cores and third-party CAD tools [2]. Since the fabrication process is not under the surveillance of the IP owner, it allows the attacker to perform malicious modifications. An HT can alter the original functionality of an IC in a critical fashion, which can cause destruction of the circuit or leakage of sensitive information [3]. Therefore, it is crucial to detect HTs (if it exists) to maintain the root of trust of an IC [4].

A primary HT consists of two major components - Trigger and Payload. The Trigger activates the HT circuit and the Payload causes malfunctioning within the circuit when triggered [5]. An adversary tries to embed the HT in such a way that the traditional manufacturing tests can not activate it. For these reasons, triggers are generally connected to those nets (rare gates) which do not switch their values in regular operation. One such example of an HT is provided in Figure1.

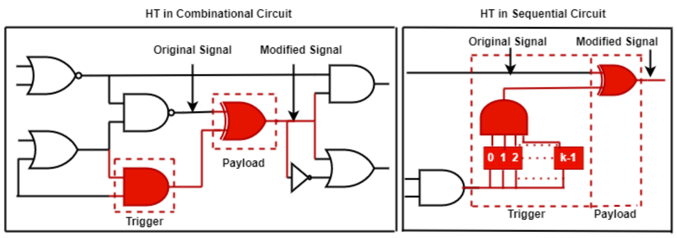

Fig. 1. Two examples showing the insertion of Hardware Trojans in a circuit netlist—(i) HT in a combinational circuit design; (ii) HT in a sequential circuit design. HTs are inserted in specific places of a circuit design that are hard to detect and get triggered only by specific input sequences.

Since defining the characteristics of an HT is difficult, different approaches of Hardware Trojan detection have been introduced so far. These techniques can be broadly classified into two categories: 1. side-channel analysis and 2. simulation-based validation (logic testing). [6] [7].

Side-channel analysis (SCA) based methods, in spite of being performed by sophisticated devices, suffer from background noise. Hence, such techniques are often unreliable. On the other hand, logic testing is primarily based on the idea of generating and applying test vectors in a circuit to activate the HT and confirm its presence. However, finding such vectors is extremely difficult due to the huge search space as well as the unknown location of the HT circuits. In this regard, genetic algorithm (GA) appears to be a promising candidate for efficient test generation. However, traditional GA based HT detection techniques also suffer from inordinately large run times. To overcome this, the search space of GA needs to be reduced. We propose to improve the performance of genetic algorithm for test generation. The following are the key features of the proposed approach:

- We measure the influence of every primary input over each rare net for search space reduction.
- A genetic algorithm based test generation is proposed which uses a modified crossover using the best primary inputs.
- Efficacy of the proposed method is measured using eight different ISCAS '85 benchmark circuits.

- Performance is compared with traditional logic test techniques which validates the proposed method.

An overview of the paper is as follows. Section II introduces the concepts used in this work. The related works are discussed in Section III. Our proposed algorithm and the results are provided in Section IV and V respectively. Section VI concludes the paper.

II. Basics

A. Genetic Algorithm

Genetic Algorithm is a bio-inspired evolutionary algorithm based on the theory of natural selection (i.e., ”survival of the fittest”) for optimising a search problem. An initial population is defined which is the genetic representation of solution domain. From the initial population, a set of new and more adaptable solutions is generated by evaluating the solution domain based on a fitness function. The several operators like selection, crossover and mutation help in shortening the solution domain to reach an optimal solution for the given problem definition after multiple generations. [8]. The built-in parallelism of a GA allows it to explore a search space quickly. [9], making it an ideal candidate for test generation problems, as discussed in Section IV(D). The basic working principle of GA is depicted in Figure 2.

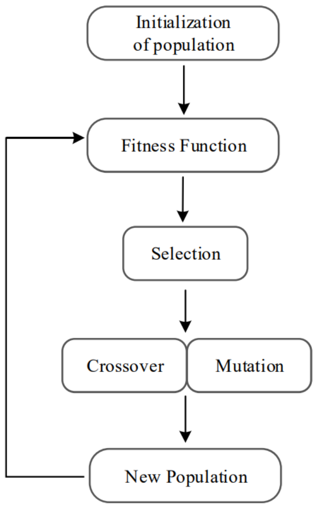

Fig. 2. A flow diagram illustrating the various steps in a Genetic Algorithm. It has been further explained in Section IV.

B. SCOAP Measurements

The Sandia Controllability/ Observability Analysis Program (SCOAP), established by Goldstein in 1980, is one of the most well-known testability analysis techniques [10]. The difficulty of changing a specific logic signal to a 0 or a 1 for a digital circuit is known as controllability. [3]. The rules shown in Table I are used to measure the combinational controllability.

The initial inputs (both for Logic 0 and 1) to a circuit are considered to be most easily controllable and therefore set to 1 and the rest are calculated as per the formulae presented in Table I. SCOAP calculation for a full adder circuit is shown in Figure 3.

TABLE I
Rules for Controllability Calculation

Condition	Output Controllability
If a Logic gate output is produced by setting only one input to a controlling value then	Min(input Controllabilities)+1
If a Logic gate output is produced by setting any input to a non controlling value then	Sum(input Controllabilities)+1

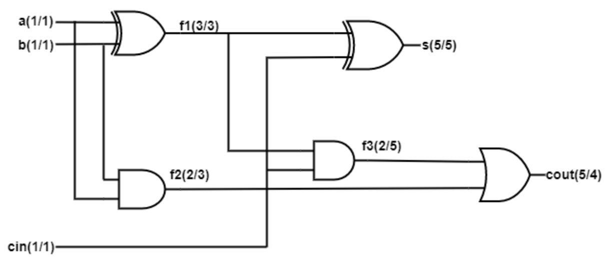

Fig. 3. Controllabilities for individual gates in a Full Adder circuit

III. Background

MERO [2] involves creating a set of minimal test patterns (minimizing test time and expense) while improving the coverage of the Trojan detection. Internal nodes with low probability conditions are identified and repeatedly triggered by a set of vectors such that they reach their rare logic value at least N times, where N be a user-defined parameter. It boosts the likelihood of activating a Trojan compared to purely random patterns by raising the toggling rate of random-pattern resistant nodes. The test patterns generated by *MERO* can be substantially enhanced by combining a Genetic Algorithm with Boolean Satisfiability [9]. The payload of potential HT samples is successfully propagated in an output that is observable. A heuristic scan partition with activity-driven test pattern generation can be used to find Trojans via dynamic power analysis [11]. As a result, the noise detected is significantly reduced. The design is partitioned into regions controlled by scan-chains and test vectors are generated to magnify the activity in the target regions and any anomaly is recorded for Trojan detection. *TRIAGE*, a standard Genetic Algorithm with transitional probability in the fitness operator can be used to generate efficient test vectors for gate-level netlists [12]. The search for efficient test vectors from a huge existing search space translates the problem into a heuristic search operation. With GA being a commonly used optimisation algorithm for conducting such heuristic search operations, it was the main inspiration behind our proposed algorithm which further improves the GA performance.

IV. Proposed Methodology

In this section, we describe the proposed methodology. We have used the multiplication of the SCOAP measurements introduced as the fitness function. Moreover, to optimize the performance of the GA, we find the primary inputs responsible for a transition over rare nets. The detailed approach is described next.

A. *Circuit Representation*

The circuit instances are selected from ISCAS '85 [13] benchmark circuits. Each circuit is represented as a directed graph $DG = \{V, E\}$ where V and E are the set of vertices and edges(Algorithm 1, line 19). We describe each logic gate as a vertex and each wire (or net) as an edge.

B. *Rare Net Selection*

It is well known that HTs are inserted in elusive places in the circuit which are often referred as rare gates (or nets). During standard functioning, rare nets do not have many transitions. To identify these nets, we simulate the circuit using a small number (e.g, 1000) of random test vectors, and select those nets whose rare values are satisfied for less than or equal to a certain threshold value. We represent the set of rare gates as RG(Algorithm 1, line 19).

C. *Effective Input Selection (select_input)*

We take the circuit netlist in the form of a directed graph (as discussed in Section IV (A)) and examine whether a path exists between the input lines(I) and the rare gates(RG) using the Breadth First Search(BFS) algorithm. For all $i \in I$ *and* $g \in RG$, the function call *BFS*(*DG*,*i*,*g*) returns 1 if a path exists between i and g(Algorithm 1, line 23) [14]. Furthermore, all the input lines for the rare gates are combined, and the duplicate input lines are removed to obtain *select_input*(SI) for the triggering condition of the rare gates in the circuit.

An Illustrative Example : We use the smallest benchmark circuit from ISCAS 85 (c17) to demonstrate the idea of *select_input* in Figure 4. Assume the red-coloured NAND gate (gate No. 16) to be a rare gate. The RED-colored nets are the connections from the primary input. Therefore, input $B1$, $B2$, and $B3$ are the effective input lines (*select_input*) whereas $B0$ and $B4$ are considered as don't care (X) bits as they do not affect the triggering of the NAND gate (gate No. 16). The GA is now performed based on input bit $B1, B2, B3$ instead of the entire input set.

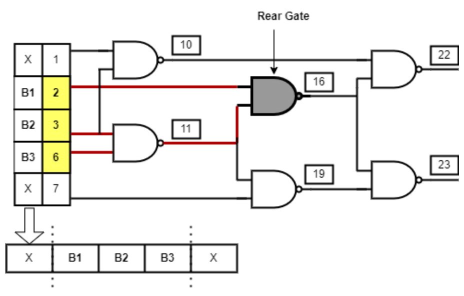

Fig. 4. An example of c17 from ISCAS'85 benchmark circuit, representing our idea of *select_input*. B1, B2, B3 is the *select_input* and the GA is performed using the *select_input* instead of the entire input set to reduce the computation overhead.

Algorithm 1 SelectInput

```
 1: procedure BFS($DG, i, g$)
 2:     $Q \leftarrow Queue$
 3:     $visited \leftarrow i$
 4:     $Q.enqueue(i)$
 5:     while Q do
 6:         $v \leftarrow Dequeue(Q)$
 7:         for $each\ u \in DG.adj(v)$ do
 8:             if $not\ visited$ then
 9:                 $visited \leftarrow u$
10:                 $Q.enqueue(u)$
11:                 if $u\ equals\ g$ then
12:                     return $1$
13:                 end if
14:             end if
15:         end for
16:     end while
17:     return $0$
18: end procedure
19: procedure SELECT($DG = \{V, E\}, RG, I$)
20:     $select\_inputs \leftarrow null$
21:     for $each\ gate\ g \in RG$ do
22:         for $each\ input\ i \in I$ do
23:             if $BFS(DG, i, g)$ then
24:                 $select\_input \leftarrow i$
25:             end if
26:         end for
27:     end for
28:     return $select\_input$
29: end procedure
```

Algorithm 2 GeneticAlgorithm

```
 1: procedure GA($circuitnetlist, RG, SI, I$)
 2:     $n \leftarrow length(SI)$
 3:     $k \leftarrow length(I)$
 4:     Initialise the GA with a random population, $P$ of 1000
        vectors; each vector having size $n$.
 5:     for $i = 1$ to $100$ do
 6:         Compute the fitness of every individual of $P$ by
        simulating the netlist with a pseudovector of size $k$ with
        $n$ bits and $(k - n)$ don't care bits.
 7:         Sort P in descending order of fitness values.
 8:         Selection of parents from the $i^{th}$generation ran-
        domly - one from top 10% and the other from bottom
        90%.
 9:         Two-point crossover to produce children.
10:         Mutation according to mutation rate.
11:         Compute fitness for children $(i + 1)^{th}$generation.
12:         $FP \leftarrow fittest\ individuals\ of\ the\ generation$
13:     end for
14:     return $FP$
15: end procedure
```

D. Proposed Genetic Algorithm

We describe the parameters of our proposed GA based technique here. The pseudocode for the same is provided in Algorithm 2.

1) Initial Population: The initial population is assigned a set of 1000 test vectors that are randomly generated.

2) Fitness Function: The quality of a solution in a GA is determined by its fitness value. As already stated, we have calculated the SCOAP values using the Tesability Measurement Tool [15] and have incorporated it in the fitness function. SCOAP measurements have been extensively used in recent times against HT detection. The authors in [16] proposed a new modified SCOAP value to represent signal probability (termed as difference-amplified controllability, providing an alternative to signal probability calculations with some exceptions [17].). In this measurement, the signal probability of a net is represented as (CC0$\sqrt{CC0/CC1}$, CC1$\sqrt{CC1/CC0}$). Since the multiplication of both signal probabilities is equal to transition probability [18], we multiply both values and denote it as MCO which is actually the product of the combinational controllability values.

$$\therefore MCO = CC0\sqrt{CC0/CC1} \times CC1\sqrt{CC1/CC0}$$
$$= CC0 \times CC1 \qquad (1)$$

The fitness value of an input vector is calculated by simulating the vector on the circuit and taking the sum of MCO values of the rare nets covered by it (r_i).

$$fitness = \sum MCO_{r_i} \qquad (2)$$

For any circuit with a *k-bit input* and an *n-bit select_input*, the simulation is carried out using input vectors with an *n-bit* sub-sequence and *(k-n)* don't care('0's or '1's) bits. In our case, we considered the *n-bit* sub-sequence with *(k-n)* bits '1's (Algorithm 2, line 6).

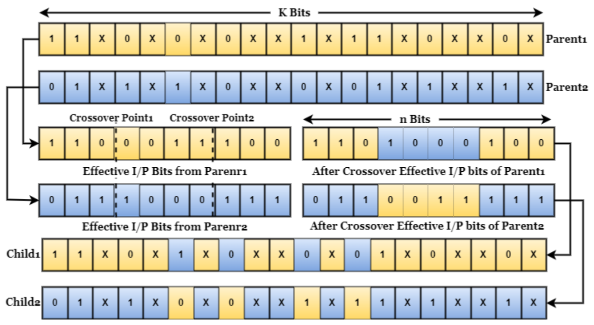
K Bits
Parent1
Parent2
Crossover Point1
Crossover Point2
n Bits
Effective I/P Bits from Parenr1
After Crossover Effective I/P bits of Parent1
Effective I/P Bits from Parenr2
After Crossover Effective I/P bits of Parent2
Child1
Child2

Fig. 5. The diagram illustrates the method used to perform a two-point crossover in our proposed algorithm.

3) Selection: The selection is based on the fitness of vectors. Every vector gets assigned a fitness value, and the entire population is arranged based on the decreasing order of fitness. One vector is selected randomly from the top 10% of the population(elite) and another from the bottom 90% of the population and sent for a two-point crossover.

4) Crossover: We have used a two point crossover operator with a probability of 0.8 on two vectors for our proposed algorithm (Figure 5) to develop the next generation. At first n effective bits ($select\_inputs$) out of total k bits are chosen from two parent vectors, leaving remaining $(k - n)$ input bit positions as don't care. Next, two crossover points are chosen randomly, and crossover is performed only on the selected n bit positions. Finally, the modified n bits along with the remaining $(k - n)$ form the child vectors.

5) Mutation: Mutation (Figure 6) is essential for the convergence of the GA since it explores the search space locally and attempts to avoid a local minima. In our proposed approach, we maintain a mutation probability of 0.1, i.e., the vector (effective n bits only) undergoing mutation will have 10 percent of bits selected randomly and flipped.

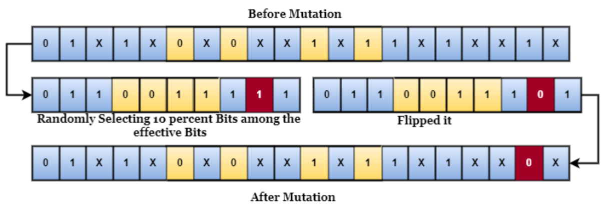

Fig. 6. The diagram illustrates the mutation operator in our proposed algorithm.

The time complexity of our proposed algorithm would be *O(npg)* where *n(size of select_input)* being the size of chromosome, *p* being the population size and *g* being the number of generations.

V. Experimental Results

A. Setup

We have implemented all the algorithms in Python 3 on an Intel Core i5 CPU running @2.40 GHz.The well established MERO algorithm [2] was implemented for comparison along side the standard GA. We used the ISCAS'85 gate-level benchmark circuits for all of our experiments [19].

B. Results

TABLE II
RELEVANT INPUTS OBSERVED IN DIFFERENT ISCAS 85 BENCHMARK CIRCUITS

Circuit Instance	Threshold = 0.1		Threshold = 0.2	
	Total no of Inputs(k)	Reduced no of Inputs(n)	Total no of Input(k)	Reduced no of Input(n)
c432	36	30	36	30
c880	60	27	60	46
c1355	NA	NA	41	41
c2670	233	12	233	18
c3540	50	38	50	38
c5315	178	4	178	7
c6288	32	32	32	32
c7522	207	16	207	16

We observed that a small number of inputs are relevant while controlling the values at rare nets. The reduced number of inputs are reported in Table II. It is well known that random test vectors are not capable of providing good coverage against

HT circuits. To cover such possible HT circuits, it is necessary to cover these rare nets. We used a relatively small threshold of 0.2 and 0.1 for rare net selection in our experiment.

1) Coverage Performance: One of the biggest challenges faced by GA based techniques is the enormous run time. To compare the performance of the proposed modified GA, we also implemented the standard GA with the same fitness function. The comparative results about rare net coverage are shown in Table III, IV, VI, VII (The coverage value reported here is the average rare net covered by a single test vector). It can be seen that, in almost all the cases, the proposed algorithm provides a better coverage when compared to both MERO and standard GA. Additionally, we have shown how our proposed algorithm outperformed the standard GA in V by improving trigger coverage.

TABLE III
COMPARISON BETWEEN MERO AND OUR PROPOSED ALGORITHM (RARENESS THRESHOLD = 0.1)

Circuit	Rare gate coverage		Time(in s)	
Instance	MERO	Proposed Algorithm	MERO	Proposed Algorithm
c432	0.79	2	43.30	93.15
c880	2.54	4	154.31	168.51
c1355	NA	NA	NA	NA
c2670	2.71	4	2463.17	557.18
c3540	9.97	22	1892.88	571.85
c5315	1	1	7104.79	896.34
c6288	35.96	33	3899.52	1202.07
c7552	3.69	6	66457.07	991.11

TABLE IV
COMPARISON BETWEEN MERO AND OUR PROPOSED ALGORITHM (RARENESS THRESHOLD = 0.2)

Circuit	Rare gate coverage		Time(in s)	
Instance	MERO	Proposed Algorithm	MERO	Proposed Algorithm
c432	1.65	3	44.31	88.46
c880	14.67	24	155.10	177.43
c1355	7.89	8	232.97	258.29
c2670	3.48	7	2532.48	452.98
c3540	28.8	34	1667.00	596.30
c5315	1.93	2	5948.34	694.12
c6288	201.62	186.07	3283.55	1114.73
c7552	3.69	6	66457.07	991.11

2) Improvement in execution time: Any circuit with k no of inputs would have a possible 2^k input combinations. Therefore, instead of searching the entire space, our proposed method optimizes the GA to run using a smaller search space, and the observed improvement in run time is significant over a standard GA. To perform a fair comparison, we performed the standard GA and our proposed algorithm using the same fitness function. An initial population of 1000 vectors was generated and both GA were allowed to run for 100 generations. The results are reported in Table VI and VII. For both the threshold values, we observed upto ≈ 50 percent improvement in run time.

TABLE V
AVERAGE TRIGGER COVERAGE IMPROVEMENT OF PROPOSED ALGORITHM OVER A CONVENTIONAL GA

Circuit Instance	# Triggers	Coverage (Standard GA)	Coverage (Proposed GA)
c880	3	0.97	3
c2670	6	5.97	6
c3540	300	202.48	172
c7552	15	14.73	15

TABLE VI
IMPROVEMENT OF PROPOSED ALGORITHM OVER A CONVENTIONAL GA (RARENESS THRESHOLD = 0.1)

Circuit Instance	% improvement in Rare gate coverage	% improvement in Execution Time
c432	0.00	5.38
c880	1.27	17.17
c1355	NA	NA
c2670	0.25	31.31
c3540	1.80	48.05
c5315	0.00	36.85
c6288	1.13	10.20
c7552	1.01	32.65

TABLE VII
IMPROVEMENT OF PROPOSED ALGORITHM OVER A CONVENTIONAL GA (RARENESS THRESHOLD = 0.2)

Circuit Instance	% improvement in Rare gate coverage	% improvement in Execution Time
c432	0	26.08
c880	16.85	1.71
c1355	0.13	15.22
c2670	1.45	46.03
c3540	4.49	44.49
c5315	0.00	51.50
c6288	5.09	22.03
c7552	1.01	32.65

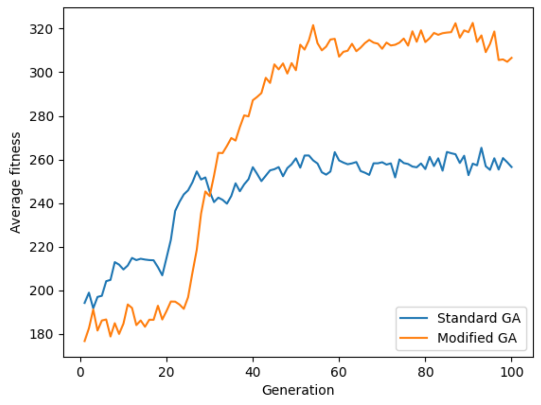

Fig. 7. Comparison of average fitness of the entire population of test vectors between our proposed algorithm and a conventional GA in the c880 netlist of ISCAS '85 benchmarks.

3) Average fitness of test vectors: One of the main advantage of using a smaller subspace during crossover is that

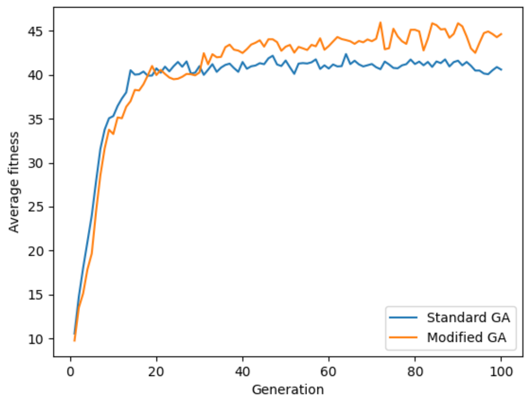

Fig. 8. Comparison of average fitness of the entire population of test vectors between our proposed algorithm and a conventional GA in the c2670 netlist of ISCAS '85 benchmarks.

it helps to converge the final result at a much faster rate. Moreover, the average fitness function observed during the GA operation is also much improved. As shown in Figures 7 and 8, the average fitness of test vectors generated by the proposed algorithm is much better that of the test vectors generated by the conventional GA. In a large circuit, where this run time will be even larger, a faster convergence can allow the tester to set a balance between test generation time and test set quality.

VI. Conclusion

In this work, we have proposed a genetic algorithm (GA) based technique for test generation against hardware Trojans. Well known HT detection methods such as MERO and the standard GA suffers in terms of execution time and are considered to be unsuitable for larger designs. Our proposed algorithm reduces the search space by finding relevant inputs first and uses this information to optimize the performance of GA. Experimental results show that the proposed GA produces equal or better rare net coverage as standard GA while simultaneously requiring a smaller run time. Compared to MERO, we found a drastic improvement ($\approx$ 18 times) in run time, making it viable for larger designs. In future, our proposed technique can be expanded to sequential designs while also providing improved fitness function for hard-to-trigger rare combinations.

References

[1] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, pp. 1–23, 2016.

[2] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "Mero: A statistical approach for hardware trojan detection," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 396–410.

[3] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, pp. 10–25, 2010.

[4] C. S. Sruthi, M. Lohitha, S. Sriniketh, D. Manassa, K. Srilakshmi, and M. Priyatharishini, "Genetic algorithm based hardware trojan detection," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1. IEEE, 2021, pp. 1431–1436.

[5] A. Mondal, M. H. Mahalat, S. Mandal, S. Roy, and B. Sen, "A novel test vector generation method for hardware trojan detection," in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2019, pp. 80–85.

[6] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 408–413.

[7] N. Karimian, F. Tehranipoor, M. T. Rahman, S. Kelly, and D. Forte, "Genetic algorithm for hardware trojan detection with ring oscillator network (ron)," in *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, 2015, pp. 1–6.

[8] D. Hermawanto, "Genetic algorithm for solving simple mathematical equality problem," 2013. [Online]. Available: https://arxiv.org/abs/1308.4675

[9] S. Saha, R. S. Chakraborty, S. S. Nuthakki, D. Mukhopadhyay *et al.*, "Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 577–596.

[10] L. H. Goldstein and E. L. Thigpen, "Scoap: Sandia controllability/observability analysis program," in *Proceedings of the 17th Design Automation Conference*, 1980, pp. 190–196.

[11] X. Mingfu, H. Aiqun, and L. Guyue, "Detecting hardware trojan through heuristic partition and activity driven test pattern generation," 2014.

[12] M. Nourian, M. Fazeli, and D. Hély, "Hardware trojan detection using an advised genetic algorithm based logic testing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 461–470, 2018.

[13] F. Brglez, "A neutral netlist of 10 combinatorial benchmark circuits and a target translator in fortran," in *Int. Symposium on Circuits and Systems, Special Session on ATPG and Fault Simulation, June 1985*, 1985, pp. 663–698.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[15] Samimi, "Testability measurement tool." [Online]. Available: https://sourceforge.net/projects/testabilitymeasurementtool/

[16] K. Huang and Y. He, "Trigger identification using difference-amplified controllability and dynamic transition probability for hardware trojan detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3387–3400, 2020.

[17] A. Mondal, R. K. Biswal, M. H. Mahalat, S. Roy, and B. Sen, "Hardware trojan free netlist identification: A clustering approach," *Journal of Electronic Testing*, vol. 37, no. 3, pp. 317–328, 2021.

[18] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware trojan detection and reducing trojan activation time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 112–125, 2012.

[19] "ISCAS'85, available at http://www.pld.ttu.ee/~maksim/benchmarks/."