Journal of Electronic Testing (2023) 39:555–570
https://doi.org/10.1007/s10836-023-06088-1

MATLAB-Open Source Tool Based Framework for Test Generation for Digital Circuits Using Evolutionary Algorithms

Priyajit Bhattacharya[1] · **Rahul Bhattacharya**[1] · **Himasree Deka**[1]

Received: 29 March 2023 / Accepted: 6 October 2023 / Published online: 24 October 2023

Abstract

This paper proposes an automated framework for test generation for digital circuits, using evolutionary algorithms (EAs) such as Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) targeting single stuck-at faults. The framework is built in MATLAB environment in conjunction with an open-source fault simulator HOPE. Finding the best test pattern with maximum fault coverage from a very large search space of digital tests is proven to be time effective in VLSI circuits using EAs. This work emphasizes upon finding the best test pattern with maximum fault coverage in fewer iterations. This indeed can significantly reduce the number of required test patterns to achieve a good fault coverage quickly. Unlike open source ATPG tool ATALANTA which can generate tests only for combinational circuits, our EA based test generation framework is able to generate test pattern for combinational as well as sequential circuits. The proposed framework has introduced test generation for sequential circuits before and after their "netlist cutting". In this context, an automated tool which can perform "netlist cutting" for sequential circuits without altering the structure of combinational logic has been devised. The quality of generated test patterns is verified through fault simulation of some of the ISCAS'85 combinational and ISCAS'89 sequential benchmark circuits. The work has also explored the efficacy of the best test pattern through variations of genetic operators like crossover and mutation rate. Results show that the proposed GA and PSO outperforms the commonly known open source ATPG tool ATALANTA.

Keywords Evolutionary algorithm · ATPG · Genetic algorithm · Particle swarm optimization · Stuck-at faults · Netlist cutting

1 Introduction

The progress of VLSI technology brings with it several challenges in testing a digital circuit. Chips can develop errors as a result of flaws and imperfections in the manufacturing process, and it is crucial to find those flaws before the ICs are mounted on printed circuit boards. The number of flaws/defects has grown over time as a result of the planned chip's increased integration. Therefore, it is crucial to find the problem at the device level in order to maximize the reliability of the created chips. Additionally, numerous independent studies have demonstrated that the cost of detecting a failure in a circuit is far less expensive than detecting the same failure inside a bigger circuit that contains the same circuit as a subpart. High fault coverage must be attained so that the fault level in chips is kept below a predetermined value. However, to evaluate every potential defect in a chip, it is practically not possible to produce test vectors. Therefore, to facilitate the test generation process, defects are modelled as faults. Due to its resemblance to real defects and the computational options it provides for producing test vectors, the stuck-at fault model is among the existing fault models that is most widely recognized [16].

An Automatic Test Pattern Generator (ATPG) is a system that analyses circuits and creates a series of test patterns. As depicted in Fig. 1, there are two parts to it: a test vector generator and a simulator for simulating faults. To put it another way, ATPG is a technique used to identify an input sequence, that when applied to a digital circuit, lets us to tell the difference between a correct circuit and a faulty circuit.

Responsible editor: E. Amyeen

Rahul Bhattacharya
rahul@iitism.ac.in

Priyajit Bhattacharya
priyajitdbhattacharya@gmail.com

Himasree Deka
himasreedeka2000@gmail.com

[1] Dept. of Electronics Engg, Indian Institute of Technology (Indian School of Mines), Dhanbad, India

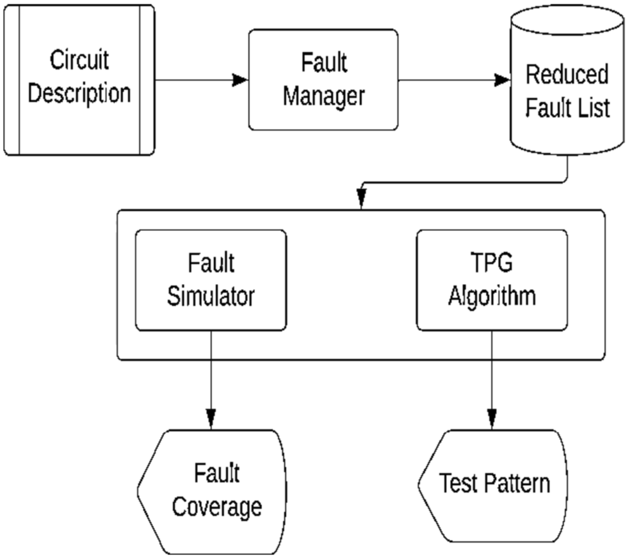


Fig. 1 ATPG Architecture

1.1 Open Source CAD Tools

ATPG tools like ATALANTA [38, 41] and FAULT [1], which are open source, are available for education and research purposes. The tools offer a variety of options, making them excellent for use in teaching and the code can be modified as per user requirements [22].

1.2 Licensed CAD Tools

Industry standard ATPG tools like “Modus" from Cadence [9], "TestMAX" by Synopsys [39], and “Tessent FastScan" by Siemens [40] are vastly used. These tools provide high fault coverage in minimum time and have highly efficient hardware utilization for ATPG.

In this paper, it has been discussed how digital circuit testing can be done using evolutionary methods, which is used to describe search, learning and optimization algorithms generally. Among these methods, the key examples include: *genetic algorithms (GA)*, *genetic programming (GP)*, *evolutionary strategy (ES), particle swarm optimization (PSO)* and *ant colony optimization (ACO).* The self-adaptation and robustness of the evolutionary algorithms are well recognized. These algorithms are widely utilized in a variety of applications, including physical design, VLSI testing, and many more that call for a planned, controlled introduction of randomness throughout the solution-finding process [10]. When there are no heuristic answers or when they typically produce unconvincing results, the EA might be used. Due to this, interest in evolutionary algorithms has lately increased, especially in the areas of practical solving of problems. EA’s offer satisfactory answers frequently and are simple to apply in comparison to other global optimization techniques.

As stated in [22], “Universities cannot always use professional ATPG tools in education due to high cost of licenses". As found in literature, the test generation algorithms in [12, 14, 22] were written in languages like C, C++, Python etc. But the MATLAB environment has not been fully investigated. Engineers and scientists can use the programming environment in MATLAB to study, create, and test systems and technologies that will change the world. Due to the command line/graphical user interface, easy random number generation and its ability to interact with various CAD tools, MATLAB proves to be an efficient environment for test generation in digital circuits using GA [13].

2 Related Work and Major Contributions

2.1 Existing Approaches on Test Generation Using Traditional ATPG Algorithms

Numerous studies have tackled the issue of creating a test vector generator with elevated fault coverage and limited test size. Socrates [33], Smart & Fast [2], Compactest [29] were one of those earlier developed ATPG which focused on identifying redundant faults and usage of dynamic compaction techniques. A. Raghuraman [30] has made a suggestion that intends to create a method for getting a single test pattern which can find all or the majority of defects in a particular combinational circuit. After identifying faults which are essential and independent, a test vector reduction based on faults is suggested in [20]. Essential test vectors can find at least one distinct fault that no other test vector can find. Then child test vectors which are essential are filtered out from essential test vectors by a technique as proposed in [23]. This resulted in an elevated number of faults being detected by a compacted test set. The simplest approach to determine the most perfect test sequence having maximum fault coverage in VLSI circuits is exhaustive testing. But it is not possible to produce all the 2^n test combinations for a circuit with n inputs. An alternative to exhaustive tests is random pattern generation which uses less test patterns overall. However, both methods generate unnecessary tests because multiple vectors can detect the same fault. Malaiya [24] presented an innovative method of testing termed antirandom testing. In this method, a test vector is selected so that it has a distance as large as possible from all the prior vectors. More faults can be found by using two test vectors with a large hamming distance between them. Although this method reduces test vectors, but it is computationally intensive. For single stuck-at (s–s-a) faults deterministic approach (for e.g., D Algorithm, PODEM, FAN) follows random test generation, but due to its huge storage requirement and increased test vector switching activity, it does not fulfill the criteria. This calls for the creation of a system that generates optimum number of vectors for testing which can detect maximum number of faults.

2.2 Existing Approaches on Test Generation Using Evolutionary Algorithms

With the advent of evolutionary algorithms [36], test patterns having better fault coverage compared to other techniques are generated. GA being one of the evolutionary algorithms is used to find optimum solution for a problem. In [32] and [37], GA was originally applied as a framework for test vector generation based on simulation, however only combinational circuits were taken into account in [37]. The test vectors generated by the CRIS test pattern generator [32] had decreased fault coverages as compared to that produced by a test vector generator based on deterministic approaches because it used a logic simulator to analyze potential test sequences. One defect was targeted at a time by another innovative method of GA-based test generation [28], which was then expanded to target 64 faults at once [11].

In [15], a genetic algorithm-based test pattern generating strategy was created to enable direct comparison with random methods. It was observed that, although the random generator gets more fault coverage quickly, but in the end, GA detects additional faults. When dealing with circuits having large number of inputs, the effectiveness of GA comes into picture. GATEST [31], a test pattern generator based on GA was accomplished around a fault simulator for sequential circuit PROOFS [27]. Huge fault coverages and compacted test vectors were achieved for combinational circuits, while deterministic techniques would yield even better results. One of the limitations was that GATEST could not identify untestable faults. In [5], results were obtained for ISCAS’89 benchmark combinational circuit (C17) using GA. One single stuck-at-fault is injected in C17 circuit and it was checked how fast GA was able to develop a test pattern to detect that fault. In [4], a genetic algorithm-based test pattern generator was developed which detects both delay and single stuck at faults for combinational circuits. In [3], the sequential circuits are tested using the evolutionary algorithms GA, PSO and Differential Evolution (DE) and the performance comparison is made among them. In [17], in order to perform ATPG for sequential circuits, a symbolic fault simulator is merged into a Genetic Algorithm (GA) environment.

2.3 MATLAB based Approaches for Test Generation for Digital Circuits

In [26], research has been made to develop an algorithm based on test pattern generation using black box and some deterministic approach in MATLAB. This algorithm achieves high fault coverage with reduced test vector count such that the vectors have large distance between them using NQ1 rule 90 and NQ1 rule 150 [42]. The test patterns are generated in MATLAB and their fault simulation are performed through ATALANTA. The results for circuits C432, C3540 and C6288 show that a smaller number of vectors achieve high fault coverage ($>85\%$). In [34], a MATLAB based D-Algorithm and PODEM algorithm has been developed for testing VLSI circuits. It is very tedious to implement these algorithms for complex circuits, so a library based on Simulink has been developed. The paper [25] explains a method for estimating a combinational circuit's nets' testability, which can either be described at the gate-level or at the component level. For each net in the network, three weight functions—CC0 (combinational controllability 0), CO (combinational observability) and CC1 (combinational controllability 1) are evaluated. The results are used in an existing ATPG technique to speed up test creation. MATLAB 7.0 has been used to implement the algorithm. The method's efficacy is demonstrated by experimental findings on the ISCAS’85 benchmark circuits.

2.4 Major Contributions

This work makes the following contributions, taking into account numerous studies previously described in the literature for test pattern generation of digital circuits:

- This work emphasizes on searching the single best test pattern with maximum fault coverage rather than a test set, for a particular digital circuit. Knowing the best test vector aids in compacting the test set size for achieving a high fault coverage, since maximum of the faults are detected by the best test vector and the remaining faults require a small number of vectors to detect them. The test generation framework for ATALANTA which uses a traditional ATPG (FAN) algorithm for test generation finds the compact test set that can detect maximum number of stuck-at faults in combinational circuits only, whereas our EA based test generation framework is able to generate test pattern for combinational as well as sequential circuits and can find the best test pattern with maximum fault coverage rather than a test set. Moreover, the proposed framework has been proven to provide fault coverage better than the open-source tool ATALANTA for the best test pattern in nearly same number of iterations.
- Unlike other works as in [31, 35], where the GA implementation has been made in languages like C and C++, in this work a GA and a PSO based test pattern generator has been devised using the well- established engineering tool MATLAB to test benchmark combinational [7] as well as sequential circuits [6].

- In this work the open-source fault simulator HOPE [21] has been used which is an improvised version of the PROOFS fault simulator used in [31]. The linking of MATLAB with the HOPE simulator justifies the ability of MATLAB for interacting with other CAD tools and opens up possibilities for exploration on test pattern generation involving MATLAB as the main tool.
- The proposed framework has introduced test generation for sequential circuits before and after their "netlist cutting". In this context, a MATLAB based EDA tool which can perform "netlist cutting" for sequential circuits without altering the structure of combinational logic has been incorporated in the framework.

The arrangement of the remaining portion of paper is as follows. Section 3 provides the entire test generation framework of GA as well as PSO in MATLAB along with an approach for test generation for sequential circuits. The tabulation of the results for the benchmark combinational and sequential circuits are presented in Sect. 4. Finally, the paper is concluded in Sect. 5.

3 Test Generation Framework: Methodology and Theoretical Background

3.1 Test Generation Framework-An Overview

Figure 2 shows the overall framework of EA that has been implemented in MATLAB for test generation for digital circuits. The inputs to the algorithm are:

- CktName – Name of the circuit to be simulated
- MaxIt – The maximum number of iterations through which the EA iterates itself
- nPop – Initial number of random individuals to be generated
- mutrate – Mutation Rate in case of GA
- nbits – Number of input bits in the circuit
- TotFaults – Total number of faults in the circuit
- TotFlops – Total number of D flip flops in the circuit

An EA begins with generation of random individuals of a population, which are then passed to the succeeding evolutionary stages. Finally, the EA framework finds the best test pattern which can detect the maximum number of s–s-a faults as an optimal or sub-optimal solution. The efficacy of the test pattern generated depends on the type of EA under consideration. HOPE simulates the single test vector generated in each run and finds the number of faults detected by that pattern, which contributes to the fitness evaluation of a test vector, deciding its ability to proceed to the next generation. The output for this proposed framework after each iteration is the best test vector with maximum fault coverage.

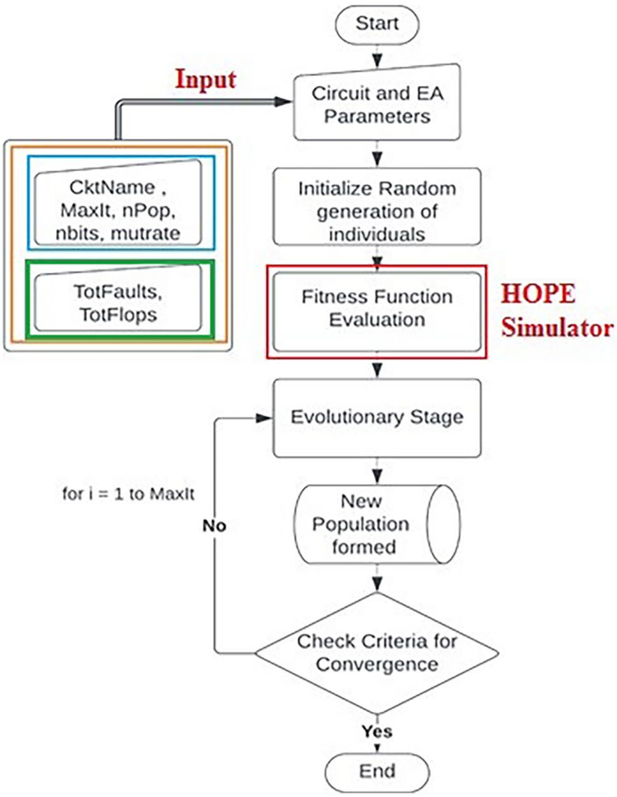

Start
Input
Circuit and EA Parameters
CktName , MaxIt, nPop, nbits, mutrate
TotFaults, TotFlops
Initialize Random generation of individuals
Fitness Function Evaluation
HOPE Simulator
Evolutionary Stage
for i = 1 to MaxIt
No
New Population formed
Check Criteria for Convergence
Yes
End

Fig. 2 Overall EA framework in MATLAB for test generation for digital circuits

The common single stuck-at (s–s-a) fault model [8, 21, 38] is considered in this framework for fault simulation using HOPE fault simulator.

3.1.1 Combinational Circuit Testing

For combinational test generation, the inputs to the framework are highlighted in only blue box in Fig. 2. The portion highlighted in red indicates the fitness function evaluation which requires the fault simulation of the generated test vectors using HOPE simulator.

3.1.2 Sequential Circuit Testing

The main problem of testing sequential circuits is that it is hard to initialize sequential circuits from primary inputs (PIs) as shown in Fig. 3. On application of test vectors, it is not possible to determine the output response of the circuit, unless the flip -flops are initialized. In this framework, test generation for the sequential circuit is performed before and after cutting its netlist.

A. Sequential Circuit Testing before Netlist Cutting

For sequential circuits before netlist cutting, the test vectors have been applied to HOPE simulator for fault

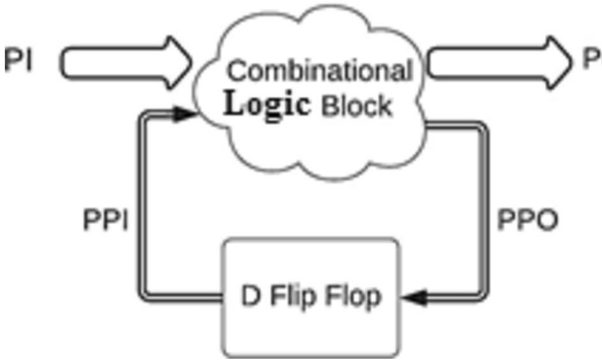


Fig. 3 Synchronous sequential circuit

simulation, with the condition that all the flip-flops are in reset state, and then HOPE finds out the number of faults detected by a single test vector generated by EA framework. The entire set of inputs for sequential circuits without netlist cutting are highlighted in orange colour.

Apart from the inputs for combinational test generation, some extra inputs are also fed for test generation for sequential circuits without netlist cutting, as follows:

- TotFaults – Total number of faults in the circuit
- TotFlops – Total number of D flip-flops in the circuit

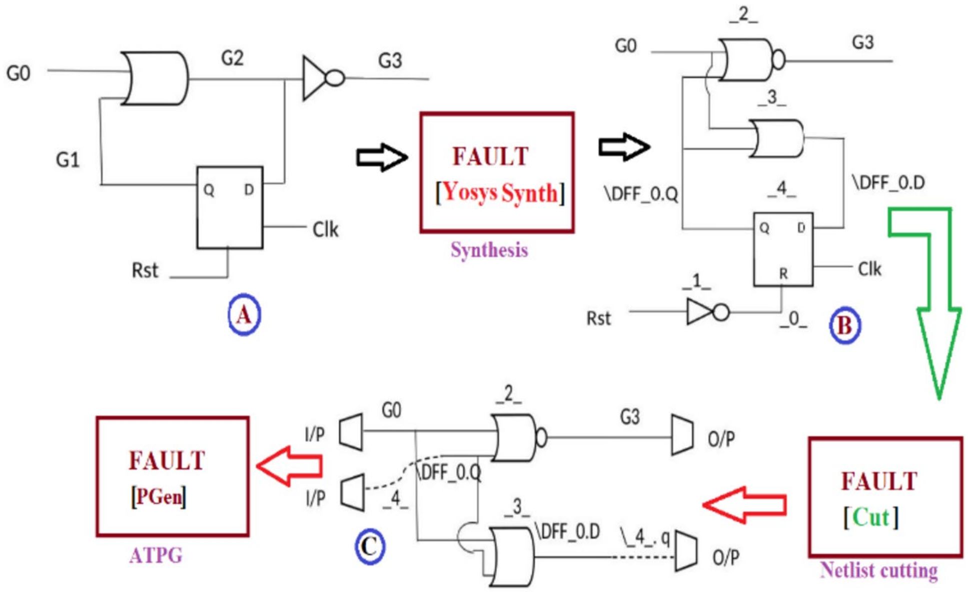
B. Sequential Circuit Testing after Netlist Cutting

Synchronous sequential circuits are usually tested using well known techniques such as *Design for Testability (DFT)* [8] which makes the flip-flops easily controllable and observable. A synchronous sequential circuit can be decomposed into flip-flops and a combinational logic block (CLB) as shown in Fig. 3. The outputs of the flip-flops are called pseudo primary inputs (PPI's) and the inputs of the flip-flops are called pseudo primary outputs (PPO's). Using the concept of "netlist cutting", flip-flops are removed from the netlist of a sequential circuit. Thus, a sequential circuit is turned into a combinational circuit. In this framework, a sequential circuit after netlist cutting is treated as a CLB and PPIs and PPOs to the CLB become PIs and POs respectively. As a result, the number of inputs and outputs as well as the length of the test pattern for each circuit increase after performing netlist cutting. This is also clearly justified in Table 4. Though the test pattern which is generated for sequential circuit after netlist cutting cannot be directly applied to the circuit under test (CUT), it can be useful for scan testing which makes PPIs controllable through scan chain.

The concept of netlist cutting has been recently introduced in FAULT [1], an open-source DFT tool chain that mainly performs ATPG of combinational and sequential circuits, scan chain insertion and scan testing. Any sequential circuit netlist undergoes two stage transformations, namely, netlist flattening and netlist cutting, before it is ready for ATPG. The various steps involved in FAULT for ATPG are shown in Fig. 4. Referring to Fig. 4, the Verilog RTL 'A' is first synthesized into a flatten netlist 'B' using Yosys synth with the help of its own standard cell library. Then, the flatten netlist 'B' is converted into pure combinational design 'C' using the concept of netlist cutting. This modified netlist is used for the ATPG process done by PGen.

In this entire process, it is observed that the structure of the original circuit gets changed in the beginning when synthesis is done. Test patterns are generated based on the transformed netlist structure. As a result, it is difficult to interpret

Fig. 4 Stages of Netlist Conversion for ATPG in FAULT



the generated tests due to augmentation of additional ports and transformation of netlist structure. For structural testing, it is essential to retain the original topology of the circuit netlist intact throughout the entire ATPG process.

To overcome the limitations mentioned above, a MATLAB based EDA tool has been incorporated. The tool can be used to convert the bench file of the original sequential netlist into the cut away version i.e., purely combinational one without altering the topology of the original netlist as shown in Fig. 5.

So, for test generation for the sequential circuits after netlist cutting, the inputs to EA framework are the same as that for the combinational circuits.

3.2 GA based Test Generation in MATLAB

In GA, an individual can be represented by a binary string (e.g., 10,101), which makes it a great choice for test vector generation in VLSI circuits. Using three basic operators, namely, selection, crossover and mutation an individual holds its existence from generation to the other. The performance of GA depends on various operators [13]. Crossover operator is one of them. In this work, we have mainly focused on three different types of crossover operators, namely, single point crossover (SPC), double point crossover (DPC) and uniform crossover (UC).

The initial step involves the generation of random individuals (chromosomes) of a search space. These individuals are analogous to the test vectors of a circuit. In this work the test vectors are selected using Roulette Wheel Selection method, in which the chance of an individual being selected as parent is directly proportional to its fitness. After two fittest individuals are selected as parents, different crossover operations are performed on them to produce two new individuals (offspring) with a large probability P_c. Crossover operation is followed by mutation, where the bits of the generated offspring are flipped in accordance with a predefined mutation probability. In GA, mutation plays the pivotal role of ensuring that some individuals of the population are not lost as the algorithm progresses and that they are tested.

In this paper, the test generation of VLSI circuits in GA has been engineered through the popularly used tool MATLAB as illustrated in Fig. 6. After the generation of individual test vectors, they are simulated for checking their respective fitness using the open-source fault simulator HOPE for both combinational and sequential circuits. Two main components of GA are finding an accurate fitness function and selection criteria.

3.2.1 Fitness Function

One crucial component of GA based test generation is the fitness function, which guides the search process towards finding the most suitable solutions. The fitness function evaluation using the HOPE simulator is as described below:

- Combinational Circuits

For combinational circuits, the fitness simply refers to the number of individual faults detected by that test vector using the parallel fault simulation technique in HOPE.

$$FF = F_d \tag{1}$$

- Sequential Circuits

The fitness function for sequential circuits after netlist cutting is taken the same as that of Eq. (1). The fitness function evaluation for sequential circuits before netlist cutting has been done according to the one performed in the fourth phase test generation in [36], where the fitness of each individual test vector is calculated as:

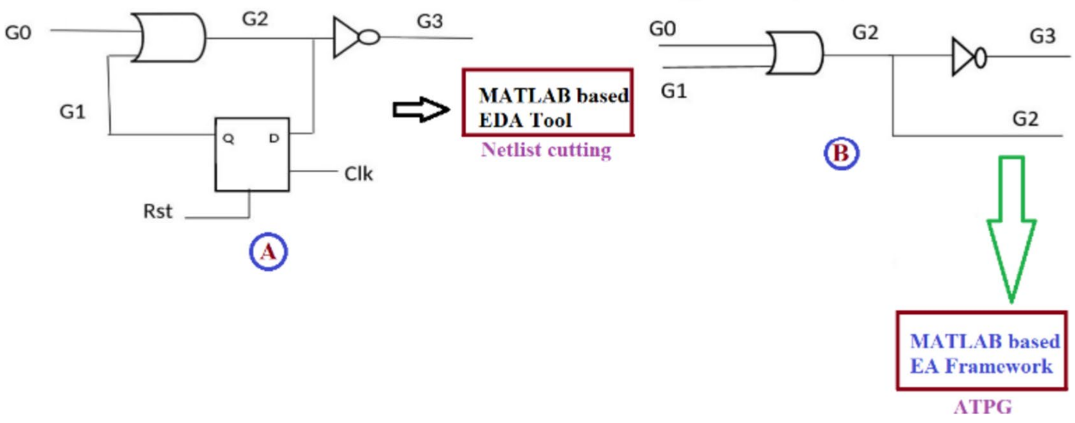


Fig. 5 Stages of netlist conversion for ATPG in EA framework

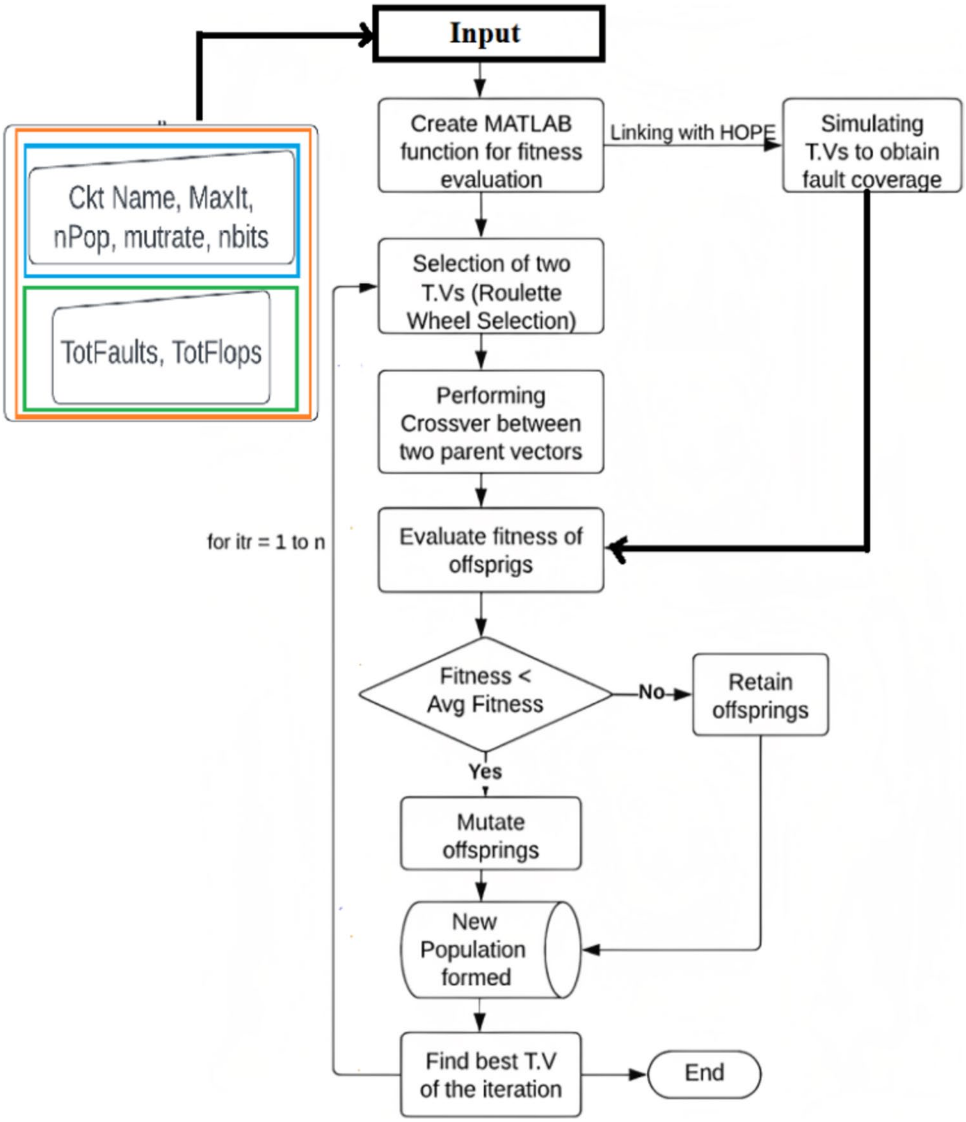

Input
Ckt Name, MaxIt, nPop, mutrate, nbits
TotFaults, TotFlops
Create MATLAB function for fitness evaluation
Linking with HOPE
Simulating T.Vs to obtain fault coverage
Selection of two T.Vs (Roulette Wheel Selection)
Performing Crossver between two parent vectors
for itr = 1 to n
Evaluate fitness of offsprigs
Fitness < Avg Fitness
No
Retain offsprings
Yes
Mutate offsprings
New Population formed
Find best T.V of the iteration
End

Fig. 6 GA implementation in MATLAB – an insight view

$$FF = F_d + F_d/FTL \tag{2}$$

where,

FF represents the fitness function.

F_d represents the number of detected faults.

F represents the total faults in a circuit.

T represents the total flip-flops in a circuit.

L represents the length of test vector.

3.2.2 Roulette Wheel Selection

After the fitness evaluation of an individual is completed, they are mapped to their corresponding probabilities using Eq. (3) and (4). The following probabilities represent the individual slots of the Roulette Wheel.

$$P(j) = e^{-\beta f(j)} \tag{3}$$

$$f(j) = c(j)/avgc \tag{4}$$

where,

P(j) represents the probability corresponding to the fitness of jth test vector. c(j) represents the fitness of jth test vector.

avgc is the average of the fitness of all test vectors of a population.

f(j) represents the normalized fitness of jth test vector.

β is termed as selection pressure, whose value is assumed to be unity in this work.

It is to be noted that the main cost of executing GA involves calculating the fitness function. The GA operators (selection, crossover and mutation) take minimal amount of time.

Although it is essential to have an accurate fitness function to achieve a good result, the large computation cost, especially for complex circuits is better to be avoided.

3.3 PSO based Test Generation in MATLAB

PSO is a potent meta-heuristic technique used in optimization, which is motivated by the nature of swarms as seen in nature, such as fishes and birds. The concept behind PSO was first introduced in [18] where a mathematical modelling of the algorithm has been provided through the following equations. These equations mimic the behavior to be shown by a group of individuals in a population to reach to an optimum solution of a problem.

$$v_{ij} = w * v_{ij} + c_1 * rand * \left(p_{ij} - x_{ij}\right) + c_2 * rand * (g_{ij} - x_{ij}) \tag{5}$$

$$x_{ij} = x_{ij} + v_{ij} \tag{6}$$

where,

x_{ij} represents the position of the particle.

v_{ij} signifies the velocity of the particle.

p_{ij} represents the best position of the particle which it occupied earlier.

g_{ij} represents the location of the best performing particle in the neighborhood.

w, c_1, c_2 are the performance parameters of the PSO. rand is a random number in [0,1], which is uniformly distributed.

In [18], introduction of PSO was made for use in the case of search spaces having real values. Many problems designed in real life features discrete variables in the search space. To expand the usage of PSO in several domains, a binary version of the same was established in [19], as any continuous or discrete value problem can be imitated in binary form. Here, the position and velocities of a particle are elucidated in terms of change in probabilities of a bit being in a particular position or another. The Eq. (5) remains unaltered in the binary version of PSO, with the change that p_{ij} and x_{ij} are now integers in {0,1} and v_{ij} lies in the range [0.0,1.0]. To accommodate v_{id} in this range, a transformation is made using sigmoid function such that

$$S\left(v_{ij}\right) = 1/1 + e^{-v_{ij}} \tag{7}$$

The effective change in position is made by the rule defined below as

$x_{ij} = 1$ if (rand $< S\left(v_{ij}\right)$)

else $x_{ij} = 0$;

The concept of binary PSO has been extended for use in ATPG for digital circuits in this paper, where $x_{ij}(x_{i1}, x_{i2}, \ldots, x_{in})$ represents a test vector as illustrated in Fig. 7.

Due of the variance in fitness values across particles, generating test sequences at random may expand the search space. Since PSO does not have a selection operation, so the velocity of a particle is adjusted in accordance with its own previous best performance and the best performance globally made by all other particles. The algorithm will update the set of previously generated test patterns in each iteration before comparing it with the global or local best solutions.

4 Results and Discussion

The efficacy of the test pattern generated using GA and PSO in our proposed framework has been validated through fault simulation using the fault simulator HOPE. The results are presented for ISCAS'85 combinational circuits and ISCAS'89 sequential circuits. The test generation framework is built on a PC with Windows 10 OS and an Intel Core *i*5 1.6 GHz processor and 8 GB RAM. The results obtained through simulation of the combinational and sequential circuits are presented in the following sections. Let F_{di} be the number of faults detected by a single test vector in ith iteration and f_{ci} be the associated fault coverage. Let F represent the total number of s–s-a faults in a given circuit.

So,

$$f_{ci}(\%) = \frac{F_{di}}{F} * 100 \tag{8}$$

Let N be the number of iterations (runs) required for EA and F_{dmax} be the maximum number of s–s-a faults detected by a single test vector.

So,

$$F_{dmax} = \max\{F_{d1,}F_{d2,} \ldots ., F_{dN}\} \tag{9}$$

and maximum fault coverage,

$$f_{cmax}(\%) = \frac{F_{dmax}}{F} * 100 \tag{10}$$

Average fault coverage for a single test vector after r number of iterations can be given by

$$f_{c_{avg}} = \frac{1}{r} \sum{}_{i=1}^{r} f_{ci} \tag{11}$$

4.1 Results: Simulation of Test Pattern for Combinational Circuits

4.1.1 Simulation of Test Pattern Generated using GA

Test patterns were generated for ISCAS'85 combinational circuits keeping the probability of crossover as unity, and mutation rate of 0.02. Higher is the probability of crossover, faster will be the convergence rate of GA. The results in Table 1 shows the performance of different crossover schemes in terms of maximum number of faults detected

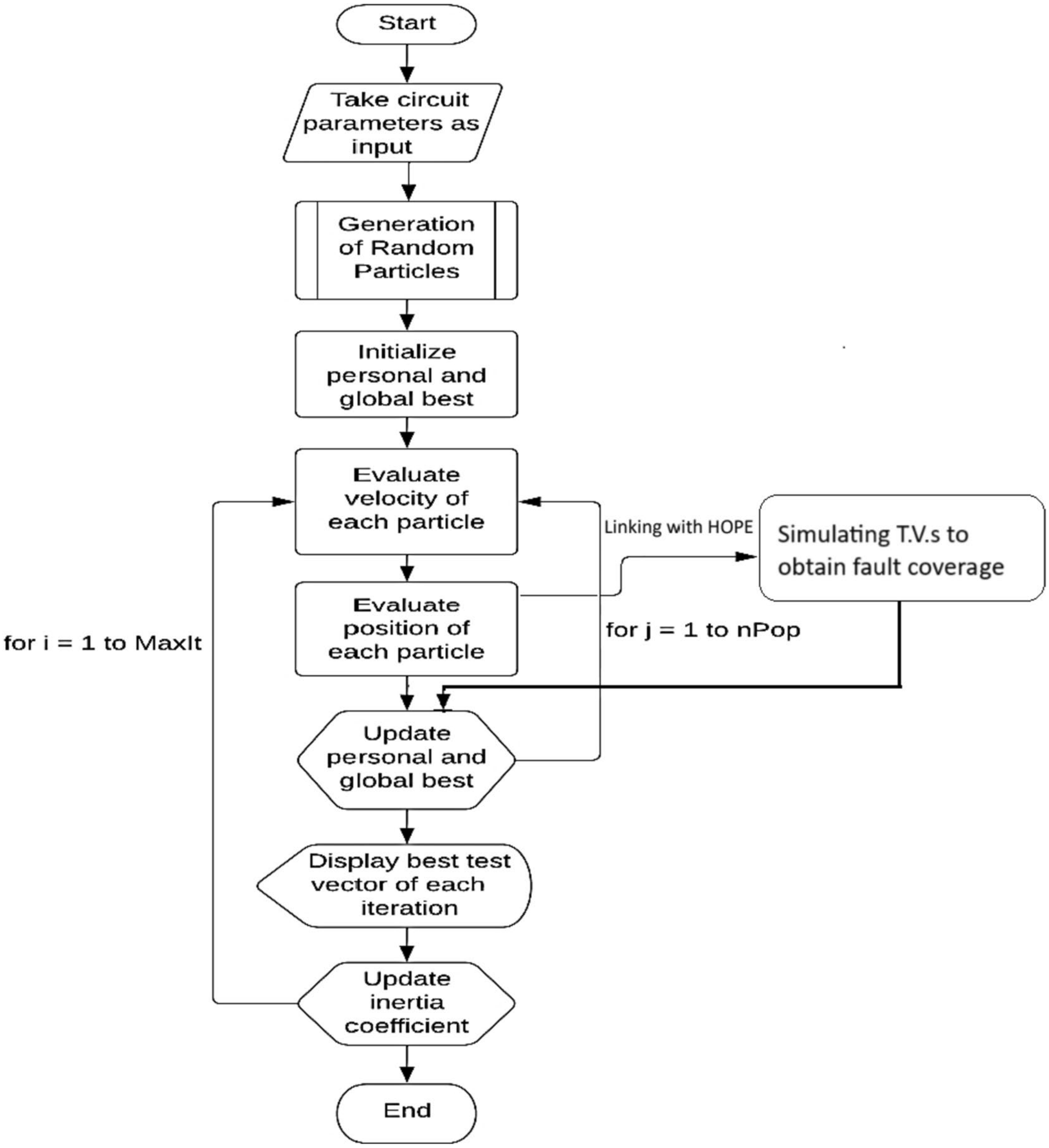

Start
Take circuit parameters as input
Generation of Random Particles
Initialize personal and global best
Evaluate velocity of each particle
Evaluate position of each particle
Linking with HOPE
Simulating T.V.s to obtain fault coverage
for j = 1 to nPop
for i = 1 to MaxIt
Update personal and global best
Display best test vector of each iteration
Update inertia coefficient
End

Fig. 7 PSO implementation in MATLAB – an insight view

by a single pattern and number of iterations for which GA is executed to find such pattern. From Table 1, it may be observed that as the crossover schemes are varied, more efficient single test vector can be obtained in a lesser or same number of iterations. In case of C17 circuit with 7 primary inputs, it has been observed that the fault coverage of a single test remains unchanged across all crossover schemes, but there is a variation in number of iterations in which the best possible result is obtained. Significant improvement in fault coverage of a single test vector is observed in case of the rest of the circuits, in lesser or same number of iterations with the variation in crossover schemes. The genetic operators are more efficient for circuits that have a large number of primary inputs.

Next, the average fault coverage attained by a single test vector along with the number of iterations required to attain this average fault coverage for different mutation rates are tabulated in Table 2. The GA can be distracted from the local minima present in the search space by using mutation. Without crossover operation, the behavior of mutation is like a random search. The crucial task of replacing characters in particular spots, that were previously off-track from the population, is accomplished by mutation when paired with selection and crossover, so that they can be tested in the current context. However, good character combinations can also be destroyed by mutation, so a balance must be maintained. Figure 8 show how GA proves to be efficient in its ability to achieve higher fault coverages in successive iterations for

Table 1 Comparison among different crossover schemes on combinational circuits

ISCAS'85 Circuits	SPC $F_{d_{max}}$ / N^*	DPC $F_{d_{max}}$ / N^*	UC $F_{d_{max}}$ / N^*
C17	9 / 3	9 / 2	9 / 2
C432	109 / 5	109 / 5	121 / 5
C880	243 / 20	248 / 17	250 / 16
C1355	497 / 5	500 / 3	501 / 3
C1908	530 / 7	545 / 7	550 / 5
C2670	639 / 10	688 / 8	698 / 8
C3540	610 / 7	613 / 6	617 / 6
C5315	1020 / 9	1040 / 7	1073 / 6
C6288	2792 / 12	2795 / 12	2804 / 12
C7552	1714 / 6	1716 / 4	1740 / 4

N^*—Number of iterations required to achieve $F_{d_{max}}$

circuits C432 and C5315 respectively. Each generation proves to derive better results than its previous one. It is observed from the below Table 2 that as the mutation rate is lowered, GA requires higher number of iterations to achieve the same or nearly the same f_{cavg}. This happens due to the fact that as the mutation rate is decreased, there is a lesser chance of altering a bit position in the test vector. As a result, some good test vectors may be overlooked from the population and remain untested. As a solution, the number of generations through which the GA iterates itself have to be increased.

4.1.2 Simulation of Test Pattern Generated using PSO and Comparison with GA and ATALANTA

The test patterns generated using PSO for some of the ISCAS'85 benchmark combinational circuits have been validated through fault simulation using HOPE. Table 3 compare the efficiacy of the test patterns generated using GA and PSO uisng our proposed framework with the test patterns generated using open-source ATPG tool ATALANTA. Each row of Table 3 presents the maximum number of single stuck-at faults detected by the best test vector generated by ATALANTA, GA and PSO respectively along with corresponding fault coverage and number of runs required in each case. The best test vector in each case has been found through iterating ATALANTA or EAs for several runs. The results show that GA and PSO are better than ATALANTA in terms of the maximum number of faults detected by a single pattern in lesser or almost same number of iterations for a given circuit. Figure 9 shows the execution of GA and PSO on circuits C432 and C6288 respectively and justifies the slower searching speed of GA in comparison with that of PSO. The results, as tabulated in Table 3, show that PSO results a significant better performance over GA in terms of its ability in finding the best test vector with maximum fault covergae for a given circuit. PSO is highly directed since it allows a particle to depend on both its self experience and the experiene of the entire group. On the contrary, GA based search enables individuals to arrive at optimal solution collectively, which results in gradual increase in time for testing. As a result, PSO outperforms GA in terms of speed and quaility of the solution.

4.2 Results: Simulation of Test Pattern for Sequential Circuits

This section presents the results obtained on simulation of GA and PSO on some of the ISCAS'89 benchmark sequential circuits (1) without performing netlist cutting and (2) after cutting the netlist. The efficacy of the test pattern generated using GA and PSO in our proposed framework

Table 2 Mutation rate comparison for combinational circuits

ISCAS'85 Circuits	Mutation Rate							
	1/16		1/32		1/64		1/128	
	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*
C17	35.45	5	33.18	8	37.95	11	36.36	14
C432	17.14	10	18.29	13	18.35	17	17.73	22
C880	24.12	10	23.92	15	24.3	20	23.68	25
C1355	31.18	15	31.19	20	31.29	25	31.14	30
C1908	24.8	10	23.43	14	24.51	20	24.27	25
C2670	22	10	21.3	15	21.81	25	20.77	30
C3540	16.86	15	16.95	20	16.91	25	16.81	30
C5315	18.94	15	18.90	22	19.05	27	18.60	32
C6288	35.91	10	35.94	15	35.91	20	35.82	25
C7552	20.59	10	20.6	15	20.53	20	20.52	25

R^* – Number of iterations required to achieve the average fault coverage

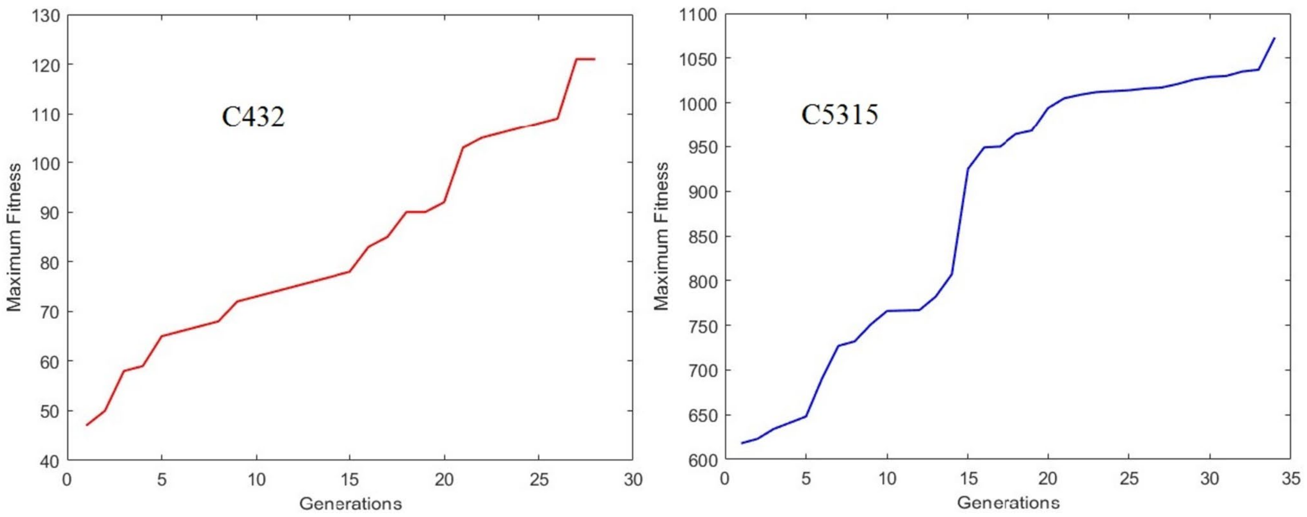


Fig. 8 Simulation of test patterns generated using GA for C432 and C5315 circuits

has been validated through fault simulation using the fault simulator HOPE which supports the fault simulation of sequential circuits. It may be noted that after netlist cutting the number of s-a faults detected in each run of GA or PSO is significantly higher than that before cutting the netlist. Many of the s–s-a faults which are undetected in a sequential circuit (before cutting the netlist) lie on the signal lines in the combinational logic block (CLB). Controllability and observability of these signal lines demand assignment of a specific set of logic vectors to PPIs. In this framework, during fault simulation of the generated test pattern using HOPE, all flip-flops are required to be either set or reset in sequential circuits before cutting the netlist. This make PPIs assignment forced to be either logic 1 or logic 0. Moreover, the faults that are able to propagate only to PPOs but unable to reach POs remain undetected in this case. On the contrary, the sequential circuits after netlist cutting are treated as CLBs during fault simulation of the generated test pattern using HOPE. So, the causes due to which many of the faults are undetected in sequential circuits before cutting the netlist are eliminated after cutting the netlist.

4.2.1 Simulation of Test Pattern Generated using GA

The parameters required for fitness function evaluation of the sequential circuits, are shown in Table 4. Test patterns were generated for ISCAS'89 circuits keeping the probability of crossover as unity, and mutation rate of 0.02.

The results in Table 5 show the performance of different crossover schemes for sequential circuits before and after netlist cutting in terms of maximum number of faults detected by a single pattern and number of iterations for which GA is executed to find such pattern. From Table 5, it may be observed that as the crossover schemes are varied

Table 3 Comparison of GA and PSO with ATALANTA for ISCAS'85 circuits

ISCAS'85 Circuits	**ATALANTA**			**GA**			**PSO**		
	F_{dmax}	f_{cmax}	N^*	F_{dmax}	F_{dmax}	N^*	F_{dmax}	f_{cmax}	N^*
C17	8	36.36	3	9	40.9	2	9	40.9	1
C432	107	20.41	5	121	23.09	5	128	24.42	3
C880	248	26.32	21	250	26.5	16	278	29.5	5
C1355	494	31.38	3	501	31.83	3	505	32.08	2
C1908	532	28.31	5	550	29.27	5	571	30.3	5
C3540	576	16.80	8	617	18.0	6	626	18.2	6
C5315	1041	19.45	7	1073	20.05	6	1118	20.8	4
C6288	2768	35.74	19	2804	36.2	12	2912	37.6	8

N—Number of iterations required to achieve* F_{dmax}

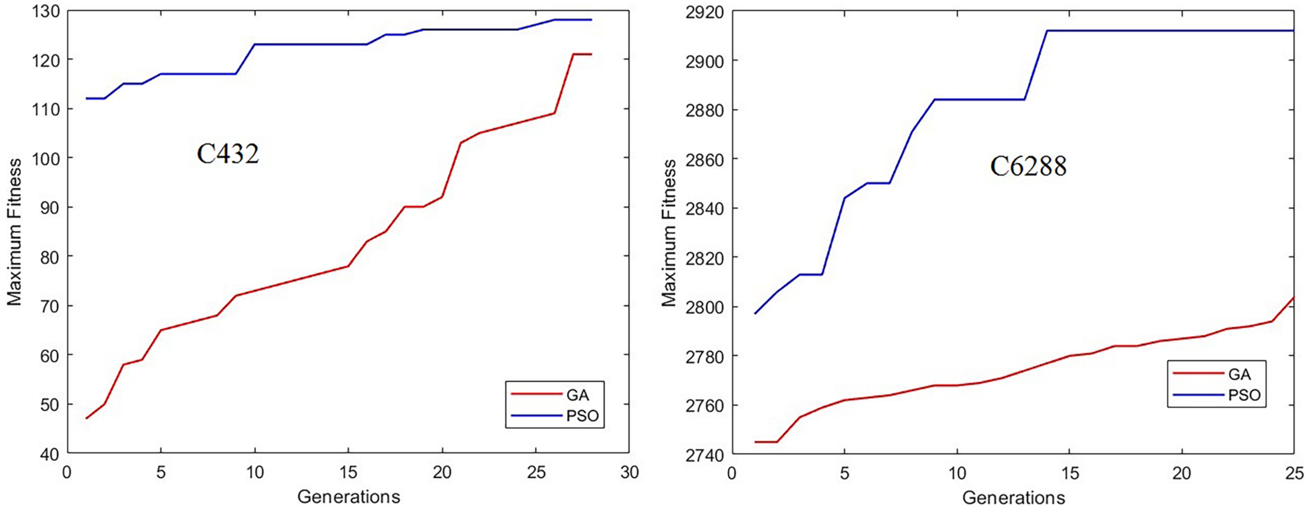


Fig. 9 Simulation of test patterns generated using GA and PSO for C432 and C6288 circuits

from SPC to UC, more efficient single test vector can be obtained in a lesser or same number of iterations. For S27 circuit, it is observed that F_{dmax} remains the same for all three crossover schemes. This may be due to the fact that S27 has only 4 PIs. Significant increase in F_{dmax} is observed for S832, S1196 and S1238 with larger number of PIs. Next, the average fault coverage attained by a single test vector along with the number of iterations required to attain this average fault coverage for different mutation rates are tabulated in Table 6. From Table 6, a similar conclusion can be reached as that of Table 2.

4.2.2 Simulation of Test Pattern Generated using PSO and Comparison with GA and ATALANTA

The test patterns generated using PSO for some of the ISCAS'89 benchmark circuits have been validated through fault simulation using HOPE before and after netlist cutting. Table 7 compare the efficacy of the test patterns generated using GA and PSO using our proposed framework with the test patterns generated using open-source ATPG tool ATALANTA. It may be noted that ATALANTA is a combinational test pattern generator. So, it is not suitable for sequential circuit before cutting its netlist. The results, as tabulated in Table 7, shows that PSO results a significant better performance compared to GA with respect to its ability in finding F_{dmax} specifically for sequential circuits after netlist cutting. However, in case of sequential circuits without netlist cutting, PSO offers a mere improvement in F_{dmax} only for the circuits S1196 and S5378. For the rest of the circuits, the results of PSO are the same as that of GA. This is because of the fact that many of the s-a faults remain undetected in a sequential circuit before cutting the netlist. EAs such as GA and PSO try to explore the possibilities of finding the best test pattern which can detect maximum number of s-a faults. In case of sequential circuits without netlist cutting, there is no scope to explore PPIs for fault detection as the flip-flops are either set or reset during fault simulation. So, the chances of finding the best test pattern

Table 4 ISCAS'89 benchmark circuit parameters

ISCAS'89 Circuits	No. of Flip Flops (T)	Total stuck-at-faults (F)	Before netlist cutting		After netlist cutting	
			No. of Inputs (L)	No. of Outputs	No. of Inputs	No. of Outputs
S27	3	32	4	1	7	4
S820	5	850	18	19	23	24
S832	5	870	18	19	23	24
S1196	18	1242	14	14	32	32
S1238	18	1355	14	14	32	32
S1488	6	1486	8	19	14	25
S5378	179	4603	35	49	214	228

Table 5 Comparison of different crossover schemes for sequential circuits

ISCAS'89 Circuits	Before netlist cutting			After netlist cutting		
	SPC	DPC	UC	SPC	DPC	UC
	F_{dmax} / N^*	F_{dmax} / N^*	F_{dmax} / N^*	F_{dmax} / N^*	F_{dmax} / N^*	F_{dmax} / N^*
S27	8 / 3	8/ 2	8 / 3	13 / 2	13 / 2	13 / 2
S820	52 / 4	53/ 3	54/ 3	74 / 2	79 / 1	81 / 2
S832	52/ 2	53/ 2	54/ 2	70 / 3	78 / 2	86 / 2
S1196	96 / 7	98/ 5	102/ 5	167 / 2	178 / 2	186 / 2
S1238	92/ 5	99/ 4	106/ 4	139 / 2	169 / 2	185 / 2
S1488	143/ 6	144/ 4	144/ 5	135 / 4	204 / 3	183 / 3
S5378	584/ 7	586 / 7	586 / 7	1078 / 2	1080 / 2	1096 / 2

N^*—Number of iterations required to achieve F_{dmax}

Table 6 Mutation rate comparison for sequential circuits

ISCAS'89 Circuits	Before netlist cutting								After netlist cutting							
	Mutation Rate								Mutation Rate							
	1/16		1/32		1/64		1/128		1/16		1/32		1/64		1/128	
	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*	f_{cavg}	R^*
S27	6.84	5	6.84	8	6.84	10	6.84	13	40	5	33.98	8	33.59	10	36.12	13
S820	0.59	8	0.58	10	0.61	15	0.60	17	8.09	8	7.91	10	7.91	15	8.08	17
S832	0.56	8	0.55	10	0.56	15	0.56	20	7.13	8	6.90	10	6.91	15	7.05	20
S1196	0.74	8	0.74	10	0.76	15	0.76	18	11.89	8	12.05	10	11.79	15	12.04	18
S1238	0.71	8	0.75	10	0.68	15	0.72	17	10.60	8	11.09	10	10.45	15	10.70	17
S1488	0.30	5	0.29	8	0.29	10	0.29	13	7.40	5	7.24	8	6.99	10	7.22	13
S5378	6.996	10	6.98	15	7.02	20	6.98	25	14.90	10	14.81	15	14.96	20	14.88	25

R^* – Number of iterations required to achieve the average fitness

Table 7 Comparison of GA and PSO with ATALANTA for ISCAS'89 circuits

Circuits	Before netlist cutting							After netlist cutting								
	ATALANTA	GA			PSO			ATALANTA			GA			PSO		
		F_{dmax}	f_{cmax}	N^*	F_{dmax}	f_{cmax}	N^*	F_{dmax}	f_{cmax}	N^*	F_{dmax}	f_{cmax}	N^*	F_{dmax}	f_{cmax}	N^*
S27	ATALANTA is not suitable for sequential circuits	8	25	3	8	25	3	13	40.62	2	13	40.62	2	14	43.75	1
S820		54	6.35	3	54	6.35	3	85	10	2	81	9.52	2	89	10.47	2
S832		54	6.21	2	54	6.21	1	86	9.89	2	86	9.89	2	89	10.22	1
S1196		102	8.21	5	107	8.61	4	180	14.49	2	186	14.97	2	213	17.14	2
S1238		106	7.82	4	106	7.82	4	190	14.02	2	185	13.65	2	221	16.31	2
S1488		144	9.69	5	144	9.69	5	176	11.84	3	204	13.72	3	225	15.14	3
S5378		586	12.73	7	587	12.75	6	1103	23.96	2	1096	23.81	2	1144	24.85	1

N^*—Number of iterations required to achieve F_{dmax}

with nearly equal fault coverage using either GA or PSO are almost same.

On the other hand, the results in Table 7 show that PSO is better than GA and ATALANTA with respect to the maximum number of faults detected by a single pattern in lesser or almost same number of iterations for sequential circuits after netlist cutting.

5 Conclusion

In this paper, two evolutionary algorithms for test pattern generation for benchmark combinational and sequential circuits are studied. GA and PSO algorithms are implemented in MATLAB and the fault simulation is performed in HOPE simulator. The results derived from both the algorithms were compared with that obtained using the well known open source ATPG tool ATALANTA. The results highlight the better performance of PSO over GA as well as ATALANTA for combinational circuits as well as sequential circuits after netlist cutting. Since PSO provides both global and local searches, it is more efficient than GA, in most optimization problems. The searching speed of GA is slower and several adjustments are needed to its parameters. The double point crossover scheme and uniform crossover scheme increases the fault coverage of a single test vector for maximum circuits. The complete study in this paper recommends using a low value of mutation rate to decrease the rate of randomness in the test generation process. The fact that GA involves numerous parameters that may be changed, implies that it provides users much command over the search technique.

Genetic algorithm always tries to find solution through evolution. Evolution is not a one way processs since natural lives many times evolve from an odd situtation, and do not always experience a favourable situation. Even, it may cause species to reach a dead end. The convergence principle is a significant phenomenon in GA. For instance, when a user initiates a GA search to reach the target, the final outcome may not be always the best possible one, and in reality, better outcomes may exist. GA search may reach a local optimum point, where it gets trapped and is unable to move forward towards global optimum, which is the actual target. Lastly, a method for test generation for sequential circuits before and after their "netlist cutting" has been presented. In this context, a EDA tool which can perform "netlist cutting" for sequential circuits without altering the structure of combinational logic has been introduced.

In this work, the efficacy of the EA based test generation framework is compared with that of well-known open source ATPG tool ATALANTA. As a future extension of the work, the same can be compared with that of commercially available industry standard ATPG tools. In fact, comparison of results with that obtained using state-of-the-art test generation tools may recommend tweaking some of the parameters of evolutionary algorithms or usage of more efficient technique like hybrid PSO (GA-PSO) to make the framework more robust. Apart from this, the EA based test generation framework can be further explored through implementation of other evolutionary algorithms such as Differential Evolution (DE) algorithm.

Funding The work is not funded by any external agency or organization.

Data Availability The resulting data may be available with the corresponding author on reasonable request. No data repository is available with the manuscript.

Declarations

Conflicts of Interests The authors declare that they have no confict of interests.

References

1. Abdelatty M, Gaber M, Shalan M (2021) Fault: Open-Source EDA's Missing DFT Toolchain. In: IEEE Design Test 38(2):45–52. https://doi.org/10.1109/MDAT.2021.3051850
2. Abramovici M, Kulikowski JJ, Menon PR, Miller DT (1986) SMART And FAST: Test Generation for VLSI Scan-Design Circuits. In: IEEE Design Test Compt 3(4):43–54. https://doi.org/10.1109/MDT.1986.294975
3. Alateeq MM, Pedrycz W (2017) Analysis of optimization algorithms in automated test pattern generation for sequential circuits. In Proc. The IEEE Int Conf Sys Man Cybern (SMC). Banff, AB, Canada 1834–1839. https://doi.org/10.1109/SMC.2017.8122883
4. Arslan T, O'Dare MJ (1997) A genetic algorithm for multiple fault model test generation for combinational VLSI circuits. In Proc. Second International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications. Glasgow, UK, 462–466. https://doi.org/10.1049/cp:19971224
5. Baid A, Srivastava AK (2013) Generating test patterns for fault detection in combinational circuits using genetic algorithm". In Proc Students Conf Eng Sys (SCES), Allahabad, India, 1–4. https://doi.org/10.1109/SCES.2013.6547506
6. Brglez F, Bryan D, Kozminski K (1989) Combinational profiles of sequential benchmark circuits. In Proc. The IEEE Int Symp Circuits Syst (ISCAS), Portland, OR, USA, 3:1929–1934. https://doi.org/10.1109/ISCAS.1989.100747
7. Brglez F, Fujiwara H (1985) A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In Proc. IEEE Int Symp Circuits Sys (ISCAS'85), Kyoto, Japan
8. Bushnell M, Agrawal V (2004) Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits. 17. Springer Science & Business Media, ISBN:0–7923–7991–8.
9. Cadence Modus DFT Software Solution from Cadence (2020) [Online]. Available at: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/test/modus-test.html
10. Chandrasekara B (2013) Evolutionary Algorithms for Low Power Test Pattern Generator. In: Int J Comp Appl (0975 – 8887) 66(7):1–6
11. Corno F, Prinetto P, Rebaudengo M, Reorda MS (1996) GATTO: a genetic algorithm for automatic test pattern generation for large synchronous sequential circuits. In: IEEE Trans Comput-Aided Des Integ Circuits Sys 15(8):991–1000. https://doi.org/10.1109/43.511578
12. Fujiwara H (1985) Fan: A fanout-oriented test pattern generation algorithm. In Proc. IEEE Int Symp Circuits and Syst 671–674
13. Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, P.41 ISBN0–201–15767–5
14. Hong SJ (1993) A 15-valued fast test generation for combinational circuits. In Proc. IEEE 2nd Asian Test Symposium (ATS), Beijing, China, 113–118. https://doi.org/10.1109/ATS.1993.398789

15. Ivask E, Raik J, Ubar R (1998) Comparison of Genetic and Random Techniques for Test Pattern Generation. Masters Thesis, Computer Engineering Department, Tallinn Technical University
16. Kalyana RK (1997) Minimizing N-Detect Tests for Combinational Circuits. M.SC Thesis, Auburn University, Auburn, Alabama
17. Keim M, Drechsler N, Drechsler R, Becker B (2001) Combining GAs and Symbolic Methods for High Quality Tests of Sequential Circuits. In: J Electron Test 17:37–51. https://doi.org/10.1023/A:1011193725824
18. Kennedy J, Eberhart R (1995) Particle swarm optimization. In Proc. ICNN'95 - Int Conf Neural Netw, Perth, WA, Australia, 4:1942–1948. https://doi.org/10.1109/ICNN.1995.488968
19. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In Proc. The IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Orlando, FL, USA, 5:4104–4108. https://doi.org/10.1109/ICSMC.1997.637339
20. Khera VK, Sharma RK, Gupta AK (2019) A heuristic fault based optimization approach to reduce test vectors count in VLSI testing. In: J King Saud Univ Comput Inf Sci-Elsevier 31(2):229–234. https://doi.org/10.1016/j.jksuci.2017.02.001
21. Lee HK, Ha DS (1996) HOPE: an efficient parallel fault simulator for synchronous sequential circuits. In: IEEE Trans Comput-Aided Des Int Circuits Syst 15(9):1048–1058. https://doi.org/10.1109/43.536711
22. Lipovský M, J. Švarc J, Gramatová E, Fišer P (2016) A new user-friendly ATPG platform for digital circuits. In Proc. 2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Kosice, Slovakia. 1–4. https://doi.org/10.1109/DDECS.2016.7482474
23. Miyase K, Kajihara S (2004) XID: Don't care identification of test patterns for combinational circuits. In: IEEE Trans Comput-Aided Design Int Circuits Syst 23(2):321–326. https://doi.org/10.1109/TCAD.2003.822103
24. Malaiya YK (1995) "Antirandom testing: getting the most out of black-box testing. In Proc. Sixth International Symposium on Software Reliability Engineering. ISSRE'95, Toulouse, France. 86–95. https://doi.org/10.1109/ISSRE.1995.497647
25. Mehta U, Devashrayee NM, Dasgupta KS (2008) Implementation of Algorithm For Testability Measures Using MATLAB. In Proc. 3rd International Conference on Advanced Computing & Communication Technologies, ICACCT-2008, pp 1–5. Availaible at: https://www.academia.edu/57304615/Implementation_of_Algorithm_For_Testability_Measures_Using_MATLAB
26. Musa EA, Karrar AE, Aain AK (2016) Test Pattern Generation for Integrated Circuit Test Using Distance between Vectors. In: Int J Sci Res (IJSR), ISSN(Online): 2319–7064, 5(5):2480–2485
27. Niermann TM, Cheng WT, Patel JH (1990) PROOFS: a fast, memory efficient sequential circuit fault simulator. In Proc. 27th ACM/IEEE Design Automation Conference, Orlando, FL, USA, 535–540. https://doi.org/10.1109/DAC.1990.114913
28. Prinetto P, Rebaudengo M, Sonza Reorda M (1994) An automatic test pattern generator for large sequential circuits based on Genetic Algorithms. In Proc Int Test Conf, Washington, DC, USA, 240–249. https://doi.org/10.1109/TEST.1994.527955
29. Pomeranz I, Reddy LN, Reddy SM (1993) COMPACTEST: a method to generate compact test sets for combinational circuits. In: IEEE Transact Comput-Aided Design Int Circuits Syst 12(7):1040–1049. https://doi.org/10.1109/43.238040
30. Raghuraman A (2005) To Generate a Single Test Vector to detect all/most number of faults in a given combinational circuit. [Online]. Availaible at: https://view.officeapps.live.com/op/view.aspx?src=https%3A%2F%2Fwww.eng.auburn.edu%2F~agrawvd%2FCOURSE%2FE7250_05%2FREPORTS_PROJ%2FRaghuraman_ATPG.doc&wdOrigin=BROWSELINK
31. Rudnick EM, Patel JH, Greenstein GS, Niermann TM (1997) A genetic algorithm framework for test generation. In: IEEE Trans Comput-Aided Des Int Circuits Sys 16(9):1034–1044. https://doi.org/10.1109/43.658571
32. Saab DG, Saab YG, Abraham JA (1992) CRIS: A test cultivation program for sequential VLSI circuits. In Proc. IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, CA, USA, 216–219. https://doi.org/10.1109/ICCAD.1992.279372
33. Schulz MH, Trischler E, Sarfert TM (1988) SOCRATES: a highly efficient automatic test pattern generation system. In: IEEE Trans Comput-Aided Des Integr Circuits Syst 7(1):126–137. https://doi.org/10.1109/43.3140
34. Singh GP, Lakha BS (2011) Simulink Library Development and Implementation for VLSI Testing in Matlab. In Proc. International Conference on High Performance Architecture and Grid Computing. HPAGC 2011. Communications in Computer and Information Science169. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22577-2_31
35. Singh R, Rajawat A (2012) Implementation of Genetic Algorithm for Automatic Test Pattern Generation. In: Int J Sci Eng Res 3(4):1–6, ISSN 2229–5518
36. Skobtsov YA, Skobtsov VY (2011) 13 Evolutionary Test Generation Methods for Digital Devices. In: M. Adamsk, A. Barkalov., M. Węgrzyn. (eds) Design of Digital Systems and Devices. Lect Notes Electr Eng 79:331–361. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17545-9_13
37. Srinivas M, Patnaik LM (1993) A Simulation-Based Test Generation Scheme Using Genetic Algorithms. In Proc. Sixth International Conference on VLSI Design, Bombay, India 132–135. https://doi.org/10.1109/ICVD.1993.669663
38. Thakar S (1993) On the generation of test patterns for combinational circuits. M.SC Thesis, Virginia Tech
39. TestMAX ATPG from Synopsys (2007) [Online]. Available at:https://www.synopsys.com/implementation-andsignoff/test-automation/testmax-atpg.html
40. Tessent FastScan from Siemens (2019) [Online]. Available at:https://eda.sw.siemens.com/en-US/ic/tessent/test/fastscan/
41. User's Guide for ATALANTA (1991) Virginia Polytechnic & State University. Available at: https://github.com/hsluoyz/Atalanta
42. Wolfram S (1983) Statistical Mechanics of Cellular Automata. Rev Mod Phys 55:601–644. https://doi.org/10.1103/RevModPhys.55.601

Priyajit Bhattacharya graduated with a B.Tech. degree in Electronics and Communication Engineering in 2021 from Institute of Engineering& Management (IEM), Kolkata, India. He received his M.Tech. degree in Electronics and Communication Engineering in 2023 from Indian Institute of Technology (Indian School of Mines) Dhanbad, India. He worked as a Research and Development Intern in Siemens EDA (India) Pvt. Ltd. from July 2022 to December 2022. Currently he is working as a Member of Technical Staff in Siemens EDA (India) Pvt. Ltd. His research interests include Test & Verification of VLSI circuits, FPGA based Software-Hardware Co-Simulation, Fault Emulation and Electronic Design Automation (EDA).

Rahul Bhattacharya is an Assistant Professor in the Department of Electronics Engineering at IIT(ISM) Dhanbad. He received the B.E. degree in Electronics & Communication Engineering from NIT Durgapur in 2005 and M.S.(R) degree in Electrical Engineering from IIT Kharagpur in 2012, respectively, and the Ph.D. degree from IIT(ISM) Dhanbad in 2019. Prior to joining IIT(ISM) Dhanbad, he spent some years in ST-Ericsson India Pvt. Ltd. as a Senior Design Designer in the High-Speed Link Validation group and as a research consultant in the Advanced VLSI Design Laboratory at IIT Kharagpur. His research focuses on FPGA Emulation of mixed-signal circuits, Fault modeling and diagnosis and VLSI circuits testing. Bhattacharya is the recipient of ADI Innovation Fellowship Grant from Analog Devices India Pvt. Ltd. for his exceptional research in the field of modelling, emulation and verification. He also acts as a reviewer of various National and International journals such as IEEE Open journal of the Solid State Circuits Society, IEEE Transactions on VLSI Systems, IEEE Transactions on Device and Materials Reliability, IEEE Transactions on Industrial Electronics, IEEE Access, the Proceedings of the National Academy of Sciences, India, Section A: Physical Sciences, Springer etc.

Himasree Deka graduated with a B.Tech. degree in Electronics and Tele Communication Engineering in 2022 from Assam Engineering College, India. She is pursuing her M.Tech. degree in Electronics and Communication Engineering since 2022 in Indian Institute of Technology (Indian School of Mines) Dhanbad, India. Her research interests include Testing of VLSI Circuits using Evolutionary Algorithms.