Journal of Theoretical and Applied Information Technology
30th April 2025. Vol.103. No.8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

GENERATING OPTIMAL TEST CASES USING ELITIST GENETIC ALGORITHM

[1]ASHOK KUMAR BANDLA, [2]C S PAVAN KUMAR, [3]DR K KOTESWARA RAO, [4]DR O. RAMA DEVI, [5]DR KALAIVANI K, [6]KOLACHANA SWETHA, [7]DR. CHANDANAPALLI SURESH BABU

[1]Assoc Prof, Dept. of CSE(AI&ML), Ramachandra College of Engineering, Eluru, Andhra Pradesh.
[2]Asst Professor, Dept.of IT, Siddhartha Academy of Higher Education (Deemed to be Univ.), Vijayawada.
[3]Assoc Professor, Dept.of CSE, PVP Siddhartha Institute of Technology Vijayawada, Andhra Pradesh
[4]Prof & HOD, Dept. of AI&DS, Lakireddy Balireddy College of Engineering, Mylavaram, Andhra Pradesh
[5]Assoc.Professor& HOD, Dept. of IT, Vignana Bharathi Institute of Technology, Ghatkesar, Hyderabad,
[6]Asst Professor Dept. of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur AP.
[7] Professor, Dept. of IT, Seshadri Rao Gudlavalleru Engineering College, Gudlavalleru. Andhra Pradesh
Email: drkoteswararo83@gmail.com

ABSTRACT

In order to reduce the number of test cases that do not significantly improve the mean of test coverage or where the test cases are unable to isolate mistakes, this research study examines the use and efficacy of the Elitist genetic algorithm. In this work, a genetic algorithm is used to help minimize or optimize the test cases. The algorithm creates the initial population at random, determines the fitness value using coverage metrics, and then uses genetic operations such as selection, cross-over, and mutation to select the offspring in successive generations. Specific genetic modeling processes may differ from standard genetic algorithms depending on the task. This generation process is performed until the fitness values remain unchanged for two successive generations. Convergence or a reduced test case is reached when the data generation remains unchanged for two iterations. The findings of study reveal that, genetic algorithms can greatly reduce the amount of the test cases

Keywords: *Elisist GA, Test Case, Optimal, NP Complete, Minimize*

1. INTRODUCTION

Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Among the various software testing techniques, Random Testing (RT) is the most fundamental strategy. It is simple in concept, often easy to apply, can exercise the software under test in unexpected ways, and has demonstrated effectiveness in detecting failures. Random number is widely used in cryptographic applications, which is mainly used as key. Because the security of key totally depends on the amount and randomness of itself, and very important to produce random numbers for cryptographic applications. Random number generators widely employed in commercial applications do not strictly guarantee these requirements. However, a major challenge for these approaches is the undetermined, two-dimensional, and combinatorial huge input space that they have to explore and exercise automatically.

The time test data generation using hybrid method that takes the advantages of both static and dynamic method was done. Software testing remains an extremely costly activity in the software engineering lifecycle, and as such, its automation continues to be of high concern. To generate tests that cover all of the branches in a class, the class must be instantiated, and a method call sequence may need to be generated to put the object into a certain state. Generating unit test suites automatically is an important contribution towards improving software quality, and techniques like search-based software testing dynamic symbolic execution can efficiently produce test suites achieving high code coverage Model-based system testing of applications with a GUI front-end to be more cost-effective and efficient compared to their traditional record-then-replay counter parts. A modification of RT exists to improve its efficiency, such as Adaptive Random Testing (ART) ART used to test numerical programs, based on failure patterns which consists of three categories: block pattern, strip pattern, and point pattern. However, ART is less efficient than random testing because of the extra task of ensuring even spreading of test cases, where the efficiency is measured in terms of

the time to generate a test case. Random generation of test data is based on selective inputs of some distribution. Path- oriented and structural approaches use the program's control flow graph for test data generation; they select a path, and use a technique such as symbolic execution for generation of test data. Goal-oriented test-data generation approaches select inputs to execute the selected goal, such as statement, condition coverage, decision coverage, irrespective of the path taken. The basis for a diagnosis is a test suite, and often the existing test suite is not optimized for producing high-quality diagnostic reports. Hence it is important to generate tests to improve diagnosis. There are many available test generation techniques Search-Based Software Testing (SBST). Global search algorithms are Genetic Algorithms.[28]
Modern systems are highly configurable to satisfy various needs of users For example, software applications running on mobile phones can configured with many features type of phone, operating system and installed application} Each configuration represent a different product and it may exhibit different failures. In industrial systems, there are typically millions of possible configurations where possibly a small sub set of configurations can trigger failures, the question is how to maximize failure detection when it is not possible to test all configurations [29]

Debugging is a time-consuming task in software development. Although various automated approaches have been proposed, they are not effective enough. On the other hand, in manual debugging, developers have difficulty in choosing breakpoints. To address these problems and help developers locate faults effectively, interactive fault-localization framework is used which combines the benefits of automated approaches and manual debugging. Before the fault is found, the framework continuously recommends checking points based on statements' suspicions, which are calculated according to the execution information of test cases and the feedback information from the developer at earlier checking points. This process of interactive fault detection makes qualities of software to improve to a greater extends. The interactive fault analysis makes the process more or less effective in software testing scenario. The major objective of our proposed method is to reduce the interactive faults that exist while designing software [30]. The interactive fault basically reduces the quality of particular software. Hence we have designed an efficient technique where soft computing technique like adaptive genetic algorithm for optimizing the test cases that are being generated

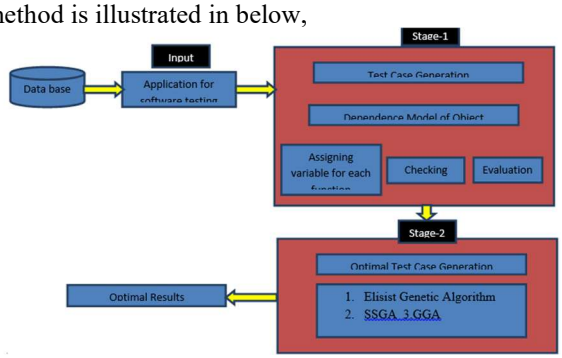
.

2. PROBLEM STATEMENT: Scalability and effectiveness is an important problem that needs to be considered while testing and it is a critical issue in the software industry. Many studies on real-world software are not so common and this is in part due to the huge computational time that is required to carry them out. The general purpose of random testing is to generate as many test cases as possible in such a way that they help uncover as many faults as many coverage targets as possible. Test cases trigger failures and do not directly uncover faults; from a mathematical standpoint we cannot consider faults as targets. Test cases are chosen with the constraint that at least one test case is chosen from each sub-domain. For example, each functionality of the software can be considered as a different sub-domain to test. During the generation of test cases, depending on the specifics of the partition strategy, had to generate and run several test cases to verify whether they belong to partition or not. An observation is showing that many program faults result in failures in contiguous areas of the input domain. ART systematically guides, or filters, randomly generated candidates, to take advantage of the likely presence of such inputs, which attempt to improve the failure-detection effectiveness of random testing. Regions of the input domain where the software produces outputs according to specification will also be contiguous. Therefore, given a set of previously executed test cases that have not revealed any failures, new test cases located away from these old ones are more likely to reveal failures.

3. PROPOSED METHOD OF IMPLEMETATION AND SOLUTION:

Software testing is huge and different field, replicating the different requirements that software artifacts must satisfy the different activities involved in testing and the different levels at which software can be checked. Arbitrary testing generates test inputs arbitrarily from the input space of the software under test. The basic feature of a random test generation technique is that it produces test inputs at arbitrary from a grammar or some other formal artifact explaining the input space. The most important purpose of the suggested method is to increase an effective technique for decreasing interactive faults based on optimal test case in direct random testing. For the generation of test cases the suggested method is employing Object

Behavior Dependence Model (OBDM). The resulting inputs are fed to the GA, SGA, EGA after the test case generation. In Elisist Genetic Algorithm the optimal inputs are produced which decrease the illegal inputs and equivalent inputs, and there by it decreases the fault proneness. The specified process of executed method is illustrated in Figure 31. The block diagram of the suggested method is illustrated in below,



.Fig 3.1 Block diagram

Dependency Model: In sequence diagram set of nodes representing objects (O_b) and set of edges that indicate the function (F) where, $F \in S_f$ represents the synchronous function. Function has the following six attributes and has a direct dependency between the source and destination objects.

$F_{source} \in O_b$ - Source of the function

$F_{dest} \in O_b$ - Destination of the function and where $F_{source} \neq F_{dest}$

F_{name} - Name of the function

$F_{BW} \in S_f$- Backward navigable function and where, $F_{BW} \neq F$ it is denoted as "-".

F_{ER} - Probabilistic execution rate of a function in a Sequence Diagram and where, $0 \leq F_{ER} \leq 1$ and the default value is 1.

F_{EER}- Expected execution rate of a function in a Sequence Diagram and where, $0 \leq F_{EER} \leq 1$ and the default value is 1.We consider a branch control structure of a source code, in which the execution rate of a function may be affected. Consider a function is in an alt combined fragment and only when the condition in the fragment is satisfied. If the function is executed within this condition fragment, then the probability of execution rate of a function is 0.5. Otherwise, the default value is 1. The expected execution rate of a function is a probability of the execution rate of a sequence diagram. In other words, it is the probability of the execution time for the total number of functions in a particular class to the execution time for the total number of functions in the whole input application. The function in a sequence diagram is executed only when it is activated. The default value of F_{EER} is also 1.Our proposed method has two stages namely,

1. Test case generation
   - ❖ Dependence Model of Object Behavior
2. Optimal Test Case Generation
   - ❖ Genetic Algorithm (AGA)
   - ❖ Steady state Genetic Algorithm (SGA)
   - ❖ Elisist Genetic Algorithm (EGA)

Stage 1:

4.1 Test Case Generation

The suggested method produces the test cases based on the Dependence Model of Object Behavior. The application which we are checking, takes as an input for object behavior dependence model in software testing. Each application has the number of function that is employed for the generation of test case. The suggested DM method principally focuses on the functions and coverage metrics of the application that we are applied for the test case generation. This will avoid generating duplicate and insignificant test cases. The function name is symbolized as a variable in our executed method. In flowchart (Figure 4.1) the overall process of test case generation is illustrated and specifically made cleared below with some examples,

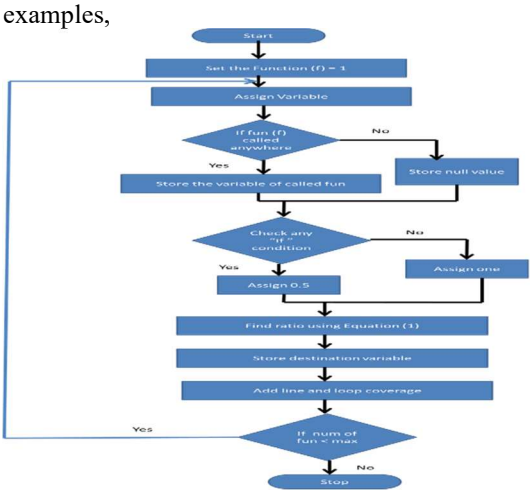


Fig 4.1 Test Case Generation

Figure.4.1 illustrates the generation process of test case. Reflect on each function as the source function. For each source function, a variable name is specified, and then each function is verified to find whether it is previously allocated for some other task. If it has previously allocated, next related variable name should be assigned in the test case, or else, it is allocating as illogical. If any

function contains "if" condition, then it allocates 0.5 values in the test case or else it assigns one. For the given function, the ratio will be worked out and at last it symbolizes the destination function. All the test cases are attached to the coverage metrics. Here we assume the probability value, any function consists "if" condition we assign 0.5 value in other cases we assign one. Ratio value will be calculated for the given function and finally represent the destination function. Then add all the test case to the coverage metrics.

$$Ratio\ value = \frac{The\ number\ of\ times\ a\ function\ called\ by\ other\ functions}{Total\ number\ of\ functions}$$

The suggested method employs only the line coverage and loop coverage from the coverage metrics.

Line coverage:

Line coverage is as well identified as the statement coverage or segment coverage. Only correct conditions are wrapped by line coverage. It as well measures the quality of the code and makes sure the flow of different path in that code.

$$line\ coverage = \frac{number\ of\ lines\ exercised}{total\ number\ of\ lines}$$

Loop coverage:

This coverage metrics reports whether each loop body is implemented zero times, precisely once and more than once. This metrics reports whether loop body is implemented precisely once and more than once for do-while loops. And as well, while-loops and for-loops perform more than once. This data is not accounted by other coverage metrics. For example, reflect on one application; it has two classes with four functions here the function names are symbolized as a variable one. The specified process illustrated in table.

Class A	Class B

A1
If
{
A2
B1
}
A2
{
C1
}
	B1
	{
	B2
	C1
	}
	B2
	If
	{
	A1
	}

Variable name for the functions A1, A2, B1, B2 and C1 are F1, F2, F3, F4 and F5. In the proposed method, the test case contains source function name, probability value, ratio value, destination function name, line coverage and loop coverage. Test case generation process with the corresponding example is given below,

Test case 1: [F1, -, 0.5, 2/5=0.4, F2] + line coverage + loop coverage

Test case 2: [F1, -, 0.5, 2/5=0.4, F1] + line coverage + loop coverage

Test case 3: [F2, -, 1, 1/5=0.2, F5] + line coverage + loop coverage

We take the input function is banking application. It contains nearly 47 classes and 108 functions. Total number of test case generation in stage 1 is around 661. In next stage resulting test cases are fed to the GA, adaptive genetic algorithm, PSO. Since we will produce test cases in each time, it encloses some resemblance on each time. The suggested method employs the adaptive genetic algorithm in order to decrease the fault occurrence of the test cases.

Stage 2:

4.2 Optimal Test Case Generation: False reduction is basically defined as the reduction of added parameters which are not required for processing. In the proposed method, we generate enormous test cases in which some of those test cases are not required for processing. Also there may be similar test cases that are being generated at each time interval. In order to select the concerned test cases, we require some efficient techniques. The next stage of the suggested method is false reduction by means of the GA, Steady State Genetic Algorithm (SGA) and EGA. Here in the proposed method Messy genetic algorithm for obtain optimized results. In this research, the optimal inputs will be produced based on Elisist Genetic Algorithm (E) which will decrease the illegal inputs and equivalent inputs.

4.2.1Genetic Algorithm: It is a stochastic algorithm: Randomness has to be essential role in GAs. Both selection and reproduction need random procedures GA always considers a population of solutions keeping in memory more than a single solution at each iteration offers a lot of advantages

The simple form of GA

1. Start with randomly generated population
2. Calculate the fitness of each chromosome in the population
3. Repeat the following steps until n off springs have been created
    - Select a pair of parent chromosomes from current population

- With probability P_c crossover the pair at randomly chosen point to form two offspring's
- Mutate the two offspring's at each locus with probability P_m

4. Replace the current population with the new population
5. Go to step 2

Generally, the initial population is generated randomly then the generation is loops over an iteration process to make the population evolve each iteration consists of the following

1. Selection: The first step consists in selecting individuals for reproduction. This selection is done randomly with probability depending on the relative fitness of individuals
2. Reproduction: This step consists of both recombination and mutation
3. Evaluation: Then the fitness of new chromosome is evaluated
4. Replacement: Individuals from the old population are killed and replaced by new ones

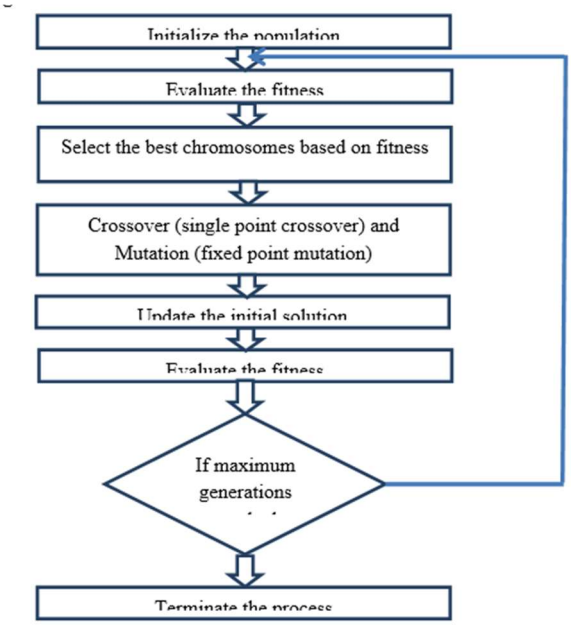


Fig 4.3 GA procedure

4.2.2 Steady State Genetic Algorithm (SGA):

A **Steady State Genetic Algorithm (SGA)** is a type of genetic algorithm that continuously updates a small portion of the population instead of replacing the entire generation at once. This approach ensures **a steady evolution** of solutions, maintaining genetic diversity while converging efficiently.

Step 1: Initialize Population

A **population** of individuals (chromosomes) is randomly generated.

Each chromosome represents a potential solution to the problem, encoded as a binary string, real numbers, or other representations.

The **population size (N)** is fixed but remains unchanged throughout the process.

Step 2: Evaluate Fitness

Each chromosome is assessed using a **fitness function**, which quantifies how well it solves the problem.

The fitness score determines the probability of selection for reproduction.

Step 3: Selection of Parents

Two individuals (parents) are selected based on their fitness. Common selection methods include:

Roulette Wheel Selection – Probability-based selection favoring fitter individuals.

Tournament Selection – A subset of the population competes, and the best individual is chosen.

Rank-Based Selection – Individuals are ranked, and selection is based on rank.

Step 4: Crossover (Recombination)

The selected parents undergo **crossover** to create offspring (one or two children).

Common crossover techniques:

Single-Point Crossover: A random point is chosen, and genetic material is swapped.

Two-Point Crossover: Two crossover points are used.

Uniform Crossover: Genes are randomly inherited from each parent.

Step 5: Mutation

A small random modification is applied to the offspring to introduce genetic diversity.

Mutation types:

Bit Flip Mutation (for binary chromosomes).

Gaussian Mutation (for real-valued encoding).

Step 6: Replacement Strategy

Instead of replacing the entire population, **SGA replaces only a few individuals** (usually the worst-performing ones).

The **worst individuals are removed**, and the newly created offspring take their place.

This ensures **steady evolution** without sudden changes in the population.

Step 7: Repeat the Process

Steps **2 to 6** are repeated continuously.

Since only a small portion of the population is replaced at a time, the algorithm maintains stability and **gradual convergence**.

Step 8: Termination Criteria

The algorithm stops when:

- A predefined number of iterations is reached.
- A solution reaches an acceptable fitness level.
- No improvement is observed for several generations.

Steady State Genetic Algorithm (SGA) ensures continuous, **gradual evolution**, avoiding drastic population changes. It is effective for **dynamic environments** where maintaining genetic diversity is essential. Its **steady replacement strategy** helps in maintaining stability while progressively improving solutions.

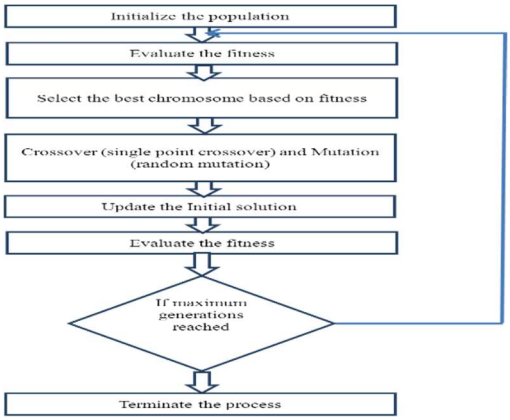


Figure 4.4 Flowchart for the proposed steady State genetic algorithm

The optimal result is contrasted to all the function after getting result from the adaptive genetic algorithm. If any function of the application not in the optimal result then the consequent function can be eliminated from the application. Hence that adaptive genetic algorithm attains decrease of faults rate based on the optimal test case. This method, the process of AGA is selecting the optimal flawless test cases which are suitable for the input application. The false reduction is thus obtained using the optimization algorithm. We obtain the optimized result based on the fitness value that we assign for AGA. The test cases that we give to the optimization algorithm are processed based on the fitness values. Test cases with high fitness value indicate the bug free application. Hence by using the AGA we obtain required test cases for our proposed method

4.2.3Step-by-Step Process of Elitist GA (Genetic Algorithm)

Elisist GA is an advanced genetic algorithm designed for optimization problems. It follows an evolutionary approach to find the best solutions by mimicking natural selection. Below is a step-by-step breakdown of how Elisist GA works:

1. Initialization

- A **population** of potential solutions (chromosomes) is randomly generated.
- Each chromosome represents a possible solution, encoded as a string (binary, real-valued, or other formats).
- The **population size** is chosen based on problem complexity.

2. Fitness Evaluation

- Each chromosome is evaluated using a **fitness function**, which determines how well it solves the given problem.
- The fitness score influences the probability of a chromosome being selected for reproduction.

3. Selection (Survival of the Fittest)

- The best-performing chromosomes are selected for reproduction using techniques like:
  - **Roulette Wheel Selection** (probability-based selection).
  - **Tournament Selection** (selecting the best from a subset).
  - **Rank-Based Selection** (assigning ranks to solutions).
- The goal is to ensure that good solutions survive and pass their traits to the next generation.

4. Crossover (Recombination)

- Two selected parent chromosomes exchange genetic material to create offspring.
- Common crossover techniques include:

Single-Point Crossover: One point is selected, and genes are swapped.

Multi-Point Crossover: Multiple crossover points are chosen.

Uniform Crossover: Each gene is inherited randomly from either parent.

This step introduces **diversity** and combines good traits from different parents.

5. Mutation

Small, random changes are introduced in some chromosomes to maintain diversity.

Mutation prevents **premature convergence** and helps escape local optima.

Mutation types include:

Bit Flip Mutation (for binary encoding).

Gaussian Mutation (for real-valued encoding).

6. Adaptive Optimization (Unique to Elisist GA)

Unlike traditional GAs, Elisist GA **dynamically adjusts** mutation rates and selection pressure based on the search progress.

This **self-adaptation** helps balance exploration and exploitation.

7. New Generation Formation

The new generation of chromosomes replaces the old population.

- The process repeats from Step 2 (Fitness Evaluation).

8. Termination Criteria

The algorithm stops when:

- A predefined number of generations is reached.
- A satisfactory fitness level is achieved.
- No significant improvement occurs over several generations.

Elitist GA enhances traditional genetic algorithms by **adapting evolutionary parameters** dynamically, making it highly effective for complex optimization problems. Its **self-learning** capabilities and **efficient selection mechanisms** enable faster convergence to optimal solutions.

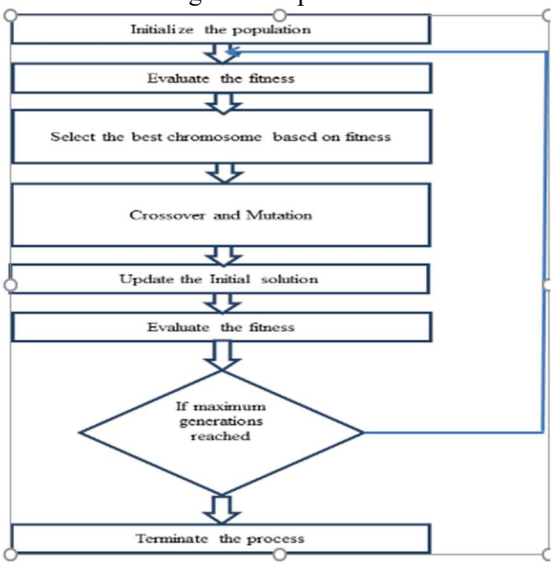


Figure 4.5 Flowchart For The Proposed Elisist Genetic Algorithm

5. RESULTS AND DISCUSSION

In the experiment, the researchers have utilized the GA, Steady State Genetic Algorithm and EGA for Optimal Test Cases in Directed Random Testing**.** By observing the Table 5.1 the fitness value for the suggested technique is verified to be superior to the technique where EGA,SGA,GA is applied.

Table 5.1: Fitness comparison for different iterations using EGA, SGA and GA.

Iterations	Fitness values		
	EGA	SGA	GA
25	680	567	631
50	640	540	620
75	593	485	568
100	587	465	486

Table 5.2: Test Count Value After Optimization.

Iterations	Test case count		
	EGA	SGA	GA
25	454	465	487
50	443	450	481
75	427	451	472
100	404	442	4549

The table 5.3, 5.4 given below shows the time and memory usage of our proposed methodology. For each iteration the corresponding time and memory usage are calculated and the results are tabulated. By reducing the interactive faults here we reduce the execution time and memory usage. When the iteration increases, the time usage and memory usage reduce automatically in EGA only

Table 5.3: Time Usage For Different Iterations.

Iteration	Time usage (milliseconds)		
	EGA	SGA	GA
25	4366	4631	5324
50	4489	4788	5475
75	4635	4834	5578
100	4612	4892	5851

Table 5.4: Memory Usage For Different Iterations.

Iteration	Memory usage (bits)		
	EGA	SGA	GA
25	3597136	3687104	3788436
50	3112288	3528428	3646828
75	2984680	3152221	3398761
100	2299412	2885821	3257892

6. CONCLUSION AND FUTURE WORK

In this work, the authors have proposed a method for reducing interactive faults based on optimal test cases in directed random testing. The implemented method is used to optimal test cases using Elitist Genetic Algorithm. Here, EGA

generates the optimal result which reduces the prohibited inputs. In order to reduce the faults, the proposed method uses the coverage metrics. The result shows that our proposed method removes the ambiguity of randomly generated test cases and produces the optimal results than SGA and GA. In future, researchers can adopt test case distribution metrics as test case selection criteria for obtaining high coverage test cases quickly and can use some other metaheuristic search tools to produce flawless test cases like Hybrid and Parallel Genetic Algorithm can be compared with variants of Particle Swarm Optimization.

REFERENCES

[1]. Vahid Garousi, "A Genetic Algorithm-Based Stress TestRequirements Generator Tooland Its Empirical Evaluation", IEEE transactions on software engineering, vol. 36, no. 6, December 2010

[2]. Reza MeimandiParizi, Abdul Azim Abdul Ghani, Rusli Abdullah, and RodziahAtan, "On the Applicability of Random Testing for Aspect-Oriented Programs", International Journal of Software Engineering and its Applications Vol. 3, No. 4, October, 2009.

[3]. Bo Yu, Zeliang Pang, "Generating Test Data Based on Improved Uniform Design Strategy", International Conference on Solid State Devices and Materials Science, vol.25,pp.1245-1252, 2012.

[4]. Lin Padgham, Zhiyong Zhang, John Thangarajah, and Tim Miller, "Model-Based Test Oracle Generation for Automated Unit Testing of Agent Systems", IEEE Transactions On Software Engineering,vol.39,no.9,1230-1244,2013.

[5]. Bestoun S. Ahmed, Mouayad A. Sahib, and Moayad Y. Potrus, "Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing", International Journal an Engineering Science and Technology,vol.17, pp.218-226,2014.

[6]. Leandro L. Minku, Dirk Sudholt, and XinYa, "Improved Evolutionary Algorithm Design for the Project Scheduling Problem Based on Runtime Analysis", IEEE Transactions On Software Engineering, vol.40,no.1,pp.83-102, 2014.

[7]. JunpengLv, Hai Hu, Kai-Yuan Cai, and TsongYueh Chen, "Adaptive and Random Partition Software testing", IEEE Transactions On Systems Man And Cybernetics: Systems, vol.44, no.12, pp.1649-1664, 2014.

[8]. Phil McMinn, Mark Harman, KiranLakhotia, Youssef Hassoun, and Joachim Wegener, "Input Domain Reduction through Irrelevant Variable Removal and Its Effect on Local, Global, and Hybrid Search-Based Structural Test Data Generation, IEEE Transactions On Software Engineering, vol.38,no.2,pp.453-477, 2012.

[9]. Tao Yuan, Xi Liu and Way Kuo, "Planning Simple Step-Stress Accelerated Life Tests Using Bayesian Methods", IEEE Transactions on Reliability, Volume: 61,Pp: 254 - 263, March 2012.

[10]. Wu, J."Stress testing software to determine fault tolerance for hardware failure and anomalies", AUTOTESTCON, 2012 IEEE, Sept. 2012.

[11]. Jianhui Jiang and Jipeng Huang, "System Modules Interaction Based Stress Testing model",IEEE transaction on application software,March 2011

[12]. Daning Hu, J.Leon Zhao and Zhimin Hua, "Banking Event Modeling and Simulation in Scenario-Oriented Stress Testing",springer,Volume 108,pp 379-389, 2012

[13]. EduardasBareisa, Vacius Jusas, Kestutis Motiejunas and Rimantas Seinauskas "the non-scan delay test enrichment based on random generated long test sequences", issn 1392 – 124x information technology and control, Vol. 39, No. 4, 2010.

[14]. Bo Zhou, Hiroyuki Okamura and Tadashi Dohi, "Enhancing Performance of Random Testing Through Markov Chain Monte Carlo Methods", IEEE transactions on computers, journal of latex class files, vol. 6, no. 1, January 2011.

[15]. Ah-Rim-Hom "Measuring Behavioral Dependency for Improving Change Proneness Prediction in UML Based Model" The Journal of Systems and Software 83 (2010) 222–234

[16]. ZhiQuan Zhou, ArnaldoSinaga and Willy Susilo "On the Fault-Detection Capabilities of Adaptive Random Test Case Prioritization: Case Studies with Large Test Suites", Hawaii International Conference on System Sciences, pp.5584-5593, 2012.

[17]. Kempka, Joseph, Phil McMinn, and Dirk Sudholt. "A theoretical runtime and empirical analysis of different alternating variable searches for search-based testing." Proceeding of conference on Genetic and evolutionary computation conference. ACM, 2013.

[18]. Juan Pablo Galeotti, Gordon Fraser and

Andrea Arcuri, "Improving Search-based Test Suite Generation with Dynamic Symbolic Execution", In proceeding of IEEE International Symposium on Software Reliability Engineering (ISSRE) , pp.360-369, 2013.

[19]. McMinn, Phil. "An identification of program factors that impact crossover performance in evolutionary test input generation for the branch coverage of C programs." Information and Software Technology, pp.153-172, 2013.

[20]. Fraser, Gordon, Andrea Arcuri, and Phil McMinn. "A Memetic Algorithm for Whole Test Suite Generation." Journal of Systems and Software, pp.1-36, 2014.

[21]. Joao Dionisio, Tiago Mota, Iola Pinto, Manfred Niehus, "Real Time Random Number Generator Testing", Conference on Electronics, Telecommunications and Computers.vol.17, pp.534-541, 2014.

[22]. XiamuNiu,Yongting Wang, Di Wu "A Method to Generate Random Number for Cryptographic Application" In proceeding of International Conference on Intelligent Information Hiding and Multimedia Signal Processing, pp.235-238. 2014.

[23]. Ali Darvish Carl K. Chang "Black-box Test Data Generation for GUI Testing" In proceeding of IEEE International Conference on Quality Software, PP.133-138, 2014.

[24]. PoonamMalpani and ParasBassi "Analytical & Empirical Analysis of External Sorting Algorithms", International Conference on Data Mining and Intelligent Computing,pp.1-6, 2014.

[25]. I PutuEdySuardiyana Putra and PetrusMursanto, "Centroids Based Adaptive Random Testing for Object Oriented Program", in proceeding of IEEE International Conference on Advanced Computer Science and Information Systems,pp.39-45,2013.

[26]. Thomas Arts, Alex Gerdes and Magnus Kronqvist, "Requirements on automatically generated random test cases", In Proceedings of IEEE Federated Conference on Computer Science and Information Systems, PP.1347-1354,2013.

[27]. S. Rajasekaran, G. A. Vijayslakshmi Pai "Neural Networks, Fuzzy Logic and Genetic Algorithms "PHI 2003 ISBN 971-81-203-2186-1

[28]. Rao, K. Koteswara, and G. S. V. P. Raju. "Developing optimal directed random testing technique to reduce interactive faults-systematic literature and design methodology." *Indian Journal of Science and Technology* 8.8 (2015): 715.

[29]. Praveen, S., et al. "The efficient way to detect and stall fake articles in public media using the block chain technique: Proof of trustworthiness." *International Journal on Emerging Technologies* 11.3 (2020): 158-163.

[30]. Kumar, J. Ratna, K. Koteswara Rao, and D. Ganesh. "Empirical investigations to find illegal and its equivalent test cases using RANDOM-DELPHI." *International Journal of Software Engineering and Its Applications* 9.11 (2015): 107-116.