

DIFFERENTIAL GENETIC ALGORITHM (DGA) BASED OPTIMAL DIRECTED RANDOM TESTING FOR REDUCING INTERACTIVE FAULTS

¹T BALAJI, ²K PRASUNA, ³SRINIVAS PADALA, ⁴ANJANEYULU NAIK R
⁵VENKATA NARAYANA T, ⁶G. N. SOWJANYA,

¹Sr.Asst.Prof., Department of ECE, PVP Siddhartha Institute of Technology, Vijayawada, A.P, India,

²Associate Prof., Department of ECE, Vijaya Institute of Technology for Women, Vijayawada, A.P, India,

³Associate Prof., Dept.of ECE, Sasi Institute of Technology and Engineering, Tadepalligudem, A.P, India

⁴Associate Prof., Dept. of EEE, Lakireddy Bali Reddy College of Engineering, Mylavaram, A.P, India

⁵Assistant Prof., Dept. of ECE, S.R.K.R. Engineering College, Bhimavaram, A.P, India

⁶Assistant Prof. Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, A.P, India

Email: balu170882@gmail.com

Abstract

The goal of software testing is to identify software flaws. Software testing is the process of confirming that a program works as intended. Test inputs are generated at random from the software's input space during random testing. to consistently provide test instances that are random and have some similarities. We will provide a method for minimizing the errors based on the best test cases produced by directed random testing in order to get around these problems. Using the object behavior dependence model as a basis, we will create an effective random testing test case in the suggested approach. The Differential Genetic Algorithm (DGA) will be used in this study to generate the best inputs, minimizing both equivalent and illicit inputs. AGA makes advantage of the test case coverage metrics to lessen fault proneness. By merging the old input with the present one, our suggested approach will reduce the input space and improve scalability and efficacy in the software testing age.

Keywords: *Testing, Test Case, Object Behavior Dependence Model, DGA, Coverage Metrics.*

1. INTRODUCTION

In the unsystematic testing method known as "black-box" testing, programs are tested by producing random, autonomous inputs. Random testing (RT) is a crucial tactic used in the midst of several software testing processes. The idea is straightforward. It has demonstrated efficacy in identifying problems, is frequently easy to use, and frequently reveals the program being tested in novel ways [1]. In cryptography, random integers are commonly employed as keys. because only the key's randomness and quantity can provide protection. Generating random numbers is essential for cryptography applications [2]

Random number initiators that are significantly involved in corporate operations do not take these needs seriously[3]. The largest obstacle to these systems' mechanical training, however, is the unpredictability, two-dimensionality, and combinatorial large input gap [4]. According to Tabular and Kalita, the fusion method is employed to generate time test data in order to compensate for mutually static and dynamic processes [5]. Because

software testing is one of the costliest processes in the software engineering lifecycle, its computerization continues to be a source of great concern ([6]. A method that identifies progression may need to be developed in order to locate the entity into a specific position, and the group must be launched in order to generate trials that wrap each and every branch in the group [7]

A crucial contribution to improving software eminence is the mechanical production of unit test sets. Techniques such as search-oriented software testing and active representative implementation can effectively produce experiment sets that achieve increased code exposure[8]. Compared to its traditional documentation, model-oriented scheme testing of functions through a GUI front-end is more efficient and proficient [9]. Adaptive Random Testing (ART), a kind of RT that is used to increase its efficacy, is used to analyze mathematical programs. It is based on break-down models, which include three types: block models, strip models, and point models [10]

However, because of the additional task of ensuring that experiment conditions are dispersed, ART is less successful than random testing. The

effectiveness of ART is determined by the amount of time needed to create an experiment condition [11]. Random test data generation generates experiment data by allocating unpredictable inputs in a variety of ways. The program's organized flow diagram is used by path-oriented and structural approaches to produce experiment data. They select a path and use a procedure, such representational implementation, to do so. Regardless of the approach being used, goal-oriented experiment data generation methods select inputs like reports, condition exposure, and decision exposure to achieve the chosen target [12]. A test collection serves as the source for identification, and often the available experiment collection is not optimal for producing high-quality analytical results. Therefore, creating tests to develop identification is important. There are numerous Search-Based Software Testing (SBST) experiment production techniques now in use. Genetic algorithms are used for global searches [13].

Debugging is a time-consuming task in software development. Numerous mechanical techniques have been proposed, however they are insufficiently effective. Additionally, developers are unsure about how to determine breakpoints in physical debugging. In order to address these issues and help developers identify errors effectively, an interactive mistake localization framework that combines the benefits of mechanical techniques and physical debugging is used. The structure continuously recommends inspection locations based on the uncertainties in the assertions before the error is established. These points are planned and compatible with the implementation knowledge of the experiment conditions and the developer's response information at earlier checking points.

2. RELATED WORKS

The improved standardized intend approach to intend experiment data for altering experiment class was proposed by Bo Yu and Zeliang Pang [14]. According to research, the improved standardized intention approach was consistent and could immediately resume the amalgamation experiment procedure for the limited experiment period resulting from the assumption testing.

A representation-oriented oracle production approach has been proposed by Padgham et al. [15] for unit testing conviction aspiration target mediators. By experimenting with 14 mediator schemes, they expand a mistake representation that is derived from the center units' attributes in order

to contain the types of mistakes that may be encountered and determine how to mechanically generate a limited, inactive oracle from the mediator intend representation Line. When more than 400 issues were raised, they were examined to determine if they were genuine errors or fake positives.

They prove that over 70% of the issues raised were related to issues with the code or the intention. Out of the 19 tests that their oracle conducted, all but five of them included errors. Additionally, they demonstrate that at least one error was found in eight of the eleven mistake categories included in their mistake representation.

A policy has been projected to be used for GUI practical testing [16]. By using combinatorial design, the policy creates the required experiment conditions for the GUI under study and then removes the ones that are not needed. The experiment circumstances that allowed for the combinatorial intent perception were optimized by utilizing the Simplified Swarm Optimization theory. By creating the settings for the experiment, it can be used to test the effectiveness of the policy in a real-world scenario.

Numerous theorems have been projected by Andrea Arcuri and Lionel Briand) to show how arbitrary testing can be used to identify relations errors and compare the results to combinatorial testing in situations where there are no restrictions on the attributes that can be a part of a product [17]. As the number of qualities increases and converges toward equal efficiency through combinatorial testing, random testing proves to be even more effective. Given that combinatorial testing necessitates significant processing overhead when hundreds or thousands of attributes are involved, the results suggest that there are plausible scenarios where random testing may outperform combinatorial testing in large schemes.

Minku, Leandro L. et al. an improved EA design after projecting a theoretical research with logical recommendations resulting from it. These include a fitness task that wants less pre-defined limitations and provides a clear gradient towards workable solutions; an improved depiction and alteration operator; and standardizing employees' dedication for various tasks to ensure they are not operational eventually [18].

A fusion approach that irregularly combines AT and random partition testing (RPT) has been proposed by Junpeng Lv et al. [19]. The method's idea came from the way policies were combined to

reduce the basic computational complexity of AT by including RPT into the experiment process without affecting the efficiency of deficiency recognition. Seven real-life theme programs were available for a condition examination. In terms of the number of experiment cases used to identify and eliminate a given amount of deficiency, the research findings show that the new policy considerably reduces the computational overhead of the innovative AT policy while still outperforming the uncontaminated RT policy and PT policy.

According to a fixed reliance study indicated by Phil McMinn et al. [20], the program segment can be used to sustain search space reduction. The role of this approach to open source and manufacturing production code is examined both hypothetically and experimentally in that document. The results provide evidence to support the assertion that, while a purely random search remains unchanged, input field reduction has a significant impact on the presentation of limited, global, and fusion searches. The role that extent plays in software testing, especially branch exposure, has been projected and investigated by Andrea Arcur [21]. We demonstrate that longer experiment series develop their testing inconsequentially on "hard" software testing standards. As a result, they contest that the alternative of the experiment series' scope was crucial for software testing. To support our claims, we conducted both experimental and hypothetical research on widely used standard and commercial software.

Another similarity metric has been proposed by to support multiclass level testing using ART. They calculated the remove between two article arrangements and two successions of strategy summons while developing test contributions using the similitude metric. They combine that metric with ART and apply it to a set of open-source apps. The experimental results show that this strategy performs better in OOS testing than other RT and ART approaches [22].

By using GSA, Syed Abdul Moeed and Niranjan Polala, investigated the many sources of input that were either legal or illegal, reducing both the illicit and identical sources of input. In order to avoid ambiguity, experiments were designed at random [23]. Additionally, testing was conducted based on GSA health, which could lessen systemic flaws. The yield result that shows the fault rate reduction should be a legal input. [26] [27]

A direct request ART technique for programming with non-numeric input sources was demonstrated by Chen, Jinfu, et al. [24]. A good "remove" measure was one of the main prerequisites for using ART with non-numeric information sources. [28] [29] To describe such a metric, they used the concepts of classes and the findings from classification segment testing. They conduct an observational evaluation program and use two standard measurements, the F-measure and the P-measure, to assess the adequacy of our system in recognizing disappointment. [30] [31] On three of the four remaining applications, our ART algorithm shows execution similar to RT and outperforms RT substantially in basic ways. Our ART calculation's determination overhead was close to R's. [32][33]

3. PROBLEM DEFINITION

In this section briefly discussed the problem definition,

- Scalability and effectiveness is an important problem that needs to be considered while testing and it is a critical issue in the software industry.
- Test cases trigger failures and do not directly uncover faults. Existing random testing will not be that much productive in terms of time consuming and cost.
- Existing ART needs higher computation overhead to achieve the even spread of test cases.
- The existing software fault prediction models require software metrics collected with automated tools and fault data belonging to previous software version or similar software project.
- The main disadvantage of random testing is 1) lengthy test case generation 2) it gives equivalent inputs for test cases 3) it creates many illegal inputs.
- Generating minimal test suites that guarantee t-wise coverage is a highly difficult problem which has been the subject of a great deal of research.

In this paper, our primary goal is to create as many test cases as we can that will assist identify as many flaws as possible for as many coverage targets. Every time it creates, these test cases have to be legitimate. In the age of software testing, another goal is to provide equal failure rates while increasing scalability and efficacy.

-detection effectiveness.

4. PROPOSED METHOD

The primary goal of the proposed technique is to provide a competent process that reduces interactive faults while maintaining the best possible experiment conditions in direct random testing. T-wise interaction flaws are taken into consideration in order to address the efficacy and scalability of random testing. The new process effectively makes use of the Object Behavior Dependence Model (OBDM) for the idea of creating the experiment circumstances.

When the experiment condition establishment reaches the final step, the resulting inputs are being delivered to the Differential Genetic Algorithm (DGA). The best inputs are produced by the Adaptive Genetic Algorithm, which significantly reduces the banned and indistinguishable inputs, hence reducing the likelihood of errors. Figure 1 elegantly displays the anticipated procedure's method of operation. In the following section, the block illustration of the novel approach is thrillingly etched out.

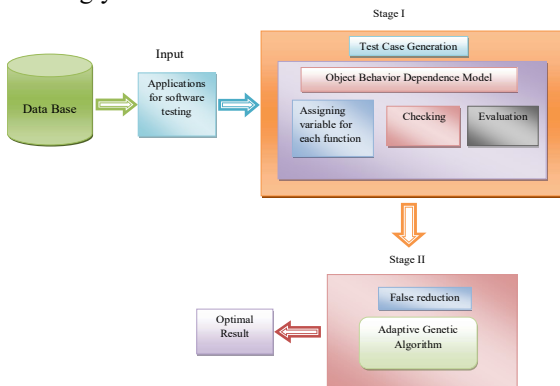


Fig.1 Block diagram of proposed method

For software testing, the input application is first chosen from the database. The Object Behavior Dependence Model (OBDM) is used to generate test cases when the input application has been chosen. Here, the input application's feature values are used to generate the test cases. The adaptive genetic algorithm does the false reduction after the test cases have been chosen. The adaptive genetic algorithm reduces the amount of equivalent and illegal inputs by generating optimal inputs. The suggested method does the false reduction based on the procedure. The two stages of the unique technique are depicted below.

1. Test case generation

❖ Object Behavior Dependence Model (OBDM)

2. False reduction

- ❖ Genetic Algorithm
- ❖ Differential Genetic Algorithm (DGA)
- ❖ Particle Swarm Optimization

Stage 1:

1.1 Test Case Generation

The predictable procedure efficiently engenders the experiment conditions in keeping through the Object Behavior Dependence Model (OBDM). The function tackled to authentication, is considered as an input for the object activities reliance representation in the software testing. The detail explanation of the Object Behavior Dependence Model is illustrated in further section,

❖ Object Behavior Dependence Model (OBDM):

In the progression diagram, the group of nodes distinguishes the objects (O_b) and the group of edges exposes the function (F), where, $F \in S_f$ distinguish the synchronous task, which hold the six characteristic comprehensive beneath and include a direct reliance among the basis and end objects.

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

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

F_{name} - characterized the name of the function

$F_{BW} \in S_f$ - corresponds to the backward navigable function and where, $F_{BW} \neq F$ and it is indicated as “-”.

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

F_{EER} - corresponds to the expected execution rate of a function in a Sequence Diagram and where, $0 \leq F_{EER} \leq 1$ and the default value is one.

We examine the state of a basis code's division control structure, where it is anticipated that the implementation charge of a job may be impacted. Assume that a job is in the mutual alteration portion and that it will only be completed once the requirement in the portion has been met. The task's implementation charge probability is 0.5 if it is carried out with the corresponding specification section around it. If not, one is the default value. The potential implementation charge of a progression diagram is represented by the expected implementation charge of a task.

As a result, it conveys the potential implementation time for the full number of tasks in

a specific class to the implementation time for the full number of tasks in the entire input function. Only when prompted does a progression diagram's utility function. Additionally, F_{EER}'s default value is one

Here, the amount of functions required to generate test cases varies by application. The proposed method determines the feature value based on the function value. The OBDM value is represented by these numbers. The task and reporting metrics of the function that have been helpful for the experiment condition development are essentially given consideration in the unique OBDM technique, which goes a long way in revealing the formation of the reproduction and unsuitable test settings. The functionality and coverage metrics of the application that we utilize to generate test cases are the primary emphasis of the suggested approach. The function names are represented as variables in our implemented method.

. In Fig.2 demonstrates the general system of experiment condition formation is gracefully revealed and is efficiently endorsement by definite occurrence comprehensive beneath,

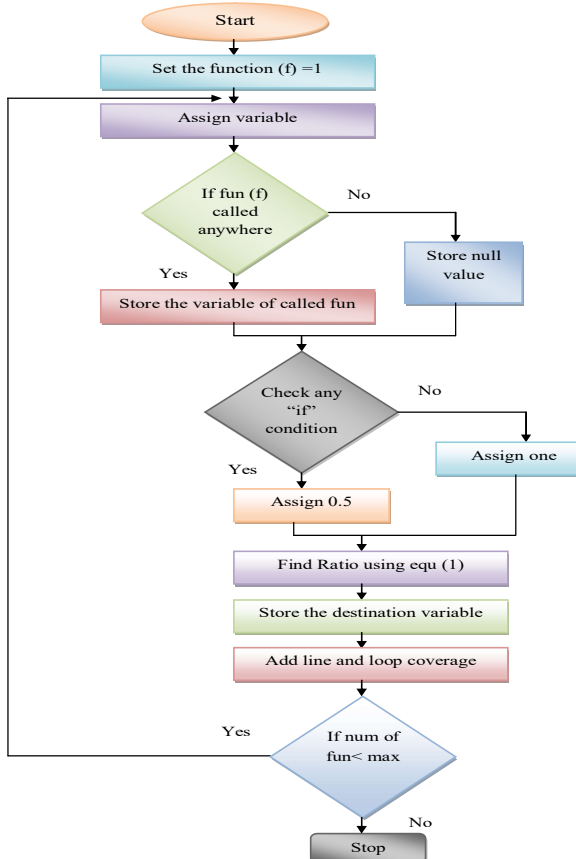


Fig.2 Flowchart for test case generation using OBDM

Figure 2 charmingly depict the experiment condition's creation structure, with each task serving as the foundational task. As a result, a basic task is used to represent a variable name. Each work is then inspected to see if it has already been assigned to a certain supplementary mission. The associated variable name must then be distributed in the experiment condition if it has already been allocated; otherwise, it is dispensed as "illogical." A value of 0.5 is dispensed in the experiment condition in the event that any job receives the "if" specification; if not, a value of one must be distributed. As a result, the representation of the target task is calculated after the specific task proportion. At the moment, the coverage is linked to the complete experiment circumstances.

A possibility value is dispensed in this condition, and each task that is distributed using the "if" specification is given a value of 0.5, while a task that is not selected in any such scenario is given a value of 1. Equation 1 below is used to evaluate the ratio for each individual task. The target task is then identified, and the coverage matrix is calculated when the entire experiment circumstances have been determined. The ratio of the amount of time that a certain function calls another function to the overall function is known as the ratio value. Equation specifies it.

(1).

$$Ratio = \frac{\text{how much time call the other function}}{\text{Total function}}$$

The epoch-making technique makes use of the line coverage and loop coverage from the coverage metrics.

Line coverage:

Additionally, line coverage is indicated as a statement that only includes the precise conditions. Additionally, it predicts the code's prominence and ensures that a variety of trails in the code in question flow. The following Equation 2 evaluates it.

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

Loop coverage:

The task of determining and illuminating whether each loop body is executed zero times, precisely once, or several times is assigned to the loop coverage tool. In the case of "do-while" loops, it also indicates whether the loop body is executed exactly once or multiple times. In addition, the for-loops and while-loops generate a lot of

presentations. Additional coverage utensils do not indicate the comparable statistics. We examine the state of a single function for a situation that has two programs and four tasks, each of which is represented by a variable. The particular process illustrated in Table

Table I: Sample code

Class A	Class B
A1	B1
If	{
{	B2
A2	C1
B1	}
}	}
A2	B2
{	If
C1	{
}	A1
	}
	}

Examples for line coverage and branch coverage

Sample code:

```
if(cond)
{
line1();
line2();
line3();
line4();
}
else
{
line5();
}
```

Four out of five lines and one out of two branches are covered if your test simply tests the condition being true and never executes the else branch. Overall, the percentage of lines that this test run executed is called Line Coverage, and the percentage of branches that this test run executed is called Branch Coverage.

Examples for test case generation

Variable names for the tasks A1, A2, B1, B2 and C1 are delivered as F1, F2, F3, F4 and F5. . In our 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

Here, F1 represents the initial function name, 0.5 indicates that if a test case contains any if condition or not. If contains means 0.5 will be assigned. If not means the value will be 1. Then the next value 2/5 indicates the ratio value of the test case mentioned in equation 1 and F2 represents the destination function name. Along with these values, line coverage and loop coverage values are used for test case generation.

Likewise, all the test cases are generated.

The banking function is now indicated by the input task. It has about 108 tasks and 58 classes. 641 is the refrain for the entire experiment condition manufacturing in phase 1. The full experiment conditions are generated in an indistinguishable manner and are thereafter sent to the adaptive genetic algorithm. Given the fact that the experiment conditions are created every time, it is reasonable to anticipate that each instance will have some degree of similarity. The adaptive genetic method is effectively used in the original procedure to lessen the likelihood of errors occurring in the experiment settings.

Stage 2:

1.2 False Reduction

False reduction is essentially defined as the limitation of extra elements that are not essential to the dispensation concept. A variety of experimental conditions are created during the highlight process, some of which are unnecessary for the dispensing mission. In addition, there is a chance that identical experiment circumstances will be created at every time point. Therefore, using the services of a certain effective system to select the corresponding experiment settings is all that is required.

. The adaptive genetic approach is skillfully used in the epoch-making process for the false reduction principle, which is achieved in accordance with the optimization. With the strong assistance of the Deferential Genetic Algorithm (DGA), it is accompanied by the false reduction process. The Deferential Genetic Algorithm (DGA), which accurately releases its obligation of reducing the illegitimate inputs and indistinguishable inputs, generates the best inputs in the current investigation. In the following section, the Adaptive Genetic Algorithm's inclusive process roadmap is fascinatingly graphed.

❖ **Differential Genetic Algorithm (AGA)**

Generally, the Genetic Algorithm symbolizes an inventive adaptive universal search method

enchanting signal from the evolutionary data of the inheritance. In this process, the Iterations and the populace are symbolized as the production and the chromosomes correspondingly. In observation of the reality that the meeting charge of the conventional GA is lesser, the DGA is efficiently engaged for significantly rushing the meeting charge, through the powerful aid of the Cauchy alteration as the transformation machinist, which construct its impressive exterior as the supreme nominee in the genetic algorithm for calculating force to the GA mission and also to modify the GA recital.

DGA distinguishes a meta-heuristic procedure devoted to the reduction of the ordinary development system. It is consistently exploited to instigate the elucidation to a number of optimization and investigate problem, that methods activated by ordinary development, like the inheritance, adaptive mutation, selection, and crossover. Supplementary, the input of AGA composes the consequence of experiment condition invention.

Initial Phase

At the outset, the populations of the chromosomes x_i , ($i = 1, 2, \dots, N$) are created arbitrarily. N represents the dimension of the population. The chromosome (x_i) encompasses the test cases produced in an arbitrary manner. Here the chromosome x_i is the set of test cases.

Fitness Evaluation

The fitness value of each one limitation is assessed as demonstrated in the subsequent Equation 3 and the chromosome containing the utmost fitness value is pick out as the finest chromosome. We have already calculated some factors for each test case such as OBDM value, fault proneness ratio, line coverage, loop coverage etc. These metrics are used to estimate fitness which is given below.

$$\text{Fitness} = (\text{If value} + \text{RV} + \text{LC} = \text{LPC}) \dots (3)$$

Here our problem is maximization of fitness to reduce the interactive faults. The OBDM value will presents the object behavioral dependency value.

Selection of Chromosomes

Single or several parent chromosomes are pick out in keeping by the ' $N/2$ ', finest chromosomes possessing the maximum fitness value and the innovative explanation is engendered.

Crossover

The solitary peak crossover is implemented at the crossover charge of (C_r) and consequently ($N/2$

) issue are accomplished. In the condition of each and every crossover task, (NC_r) genes are exchanged among the comparative parents.

Mutation

Mutation and Crossover are the two important genetic operation which will help for solution convergence. Here the adaptive behavior of GA will be presented in terms of Adaptive mutation.

Here we are using Cauchy's mutation for adaptiveness in GA. The entities are concerned probabilistically to convey a radical modify in themselves. Whereas utilizing the alteration machinist, there is a possibility that definite original qualities emerge on description of the alteration in the chromosome. The Cauchy transformation is successfully exploited to alter the entities as per Equation 4 exposed. The alteration is executed in keeping by the fixed transforming possibility. Where the Cauchy alteration is achieved, illogical variable 'x' symbolize a Cauchy allocation. The Cauchy allocation task is distinct as per the subsequent Equation 4,

$$F(x) = \frac{1}{2} + \frac{1}{\pi} \arctan(x) \quad (4)$$

Here x represents the test cases.

Updating of solution

When the transformation task is ended, innovative chromosome is engendered and subsequently the current chromosome is surrogated by innovative chromosome. This system is identified as the modernizing of explanation. If the fitness value of innovative chromosome surpasses that of the existing chromosome, the innovative chromosome is elected as the finest chromosome.

Termination Criteria

The procedure is carried on till it satisfies the termination criteria.

The flowchart for the novel technique is elegantly exhibited in Figure 3.

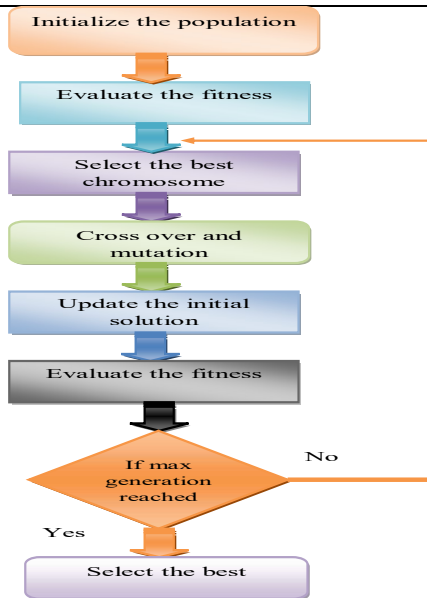


Fig.3 Flowchart For The Proposed Differential Genetic Algorithm

The most favorable product is contrasted by the complete task subsequent to accomplishing the conclusion from the adaptive genetic algorithm. Some task which does not give up the finest consequence has to be instantaneously detached from the function. Hence the adaptive genetic algorithm is capable to accomplish substantial lessening in the mistake charge rely on the best experiment condition.

In the inventive procedure, the task of the DGA is the selection of the finest ideal experiment conditions which are proper for the input function. Consequently, the counterfeit lessening is accomplished by means of the optimization algorithm and the optimized conclusion is attained rely on the fitness value selected the DGA process. The experiment conditions provided to the optimization algorithm are practiced in keeping by means of the fitness values. The experiment conditions encompassing better fitness values signify the bug free function. Therefore, through the aid of the DGA procedure, the essential experiment conditions are attained for the inventive procedure.

5 RESULT AND DISCUSSION

We employed the adaptive genetic algorithm for reduction in our experiment. Interactive Errors in Directed Random Testing Based on Ideal Test Cases. The following are the outcomes of the implementation, which was carried out on the Java platform.

The entire procedure in a genetic algorithm is based on the fitness value of the selected parameter. The parameter with the highest fitness value is selected for further processing. The chromosomes' fitness values for distinct iterations are calculated for the Adaptive Genetic Algorithm, and the outcomes are plotted. Figure 4 below illustrates the fitness value graphically using both normal and differential GA. This plots the fitness values that we acquire from the genetic algorithm and those that come from the adaptive genetic algorithm. The graph makes it clear that, in comparison to the GA, our suggested approach, which uses adaptive GA, yields a higher fitness value.

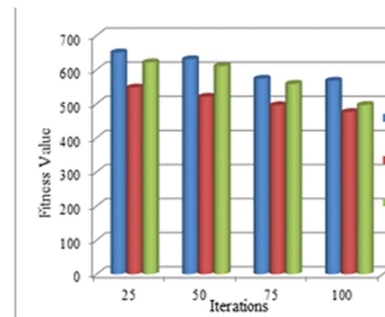


Fig 4: Graphical Representation Of Fitness Value For Different Iterations Using DGA And GA

The fig 5 shows the graphical representation of Test count value which is obtained before and after optimization. Here the test cases that are generated before and after optimization are plotted. From the graph it is evident that the Test case count has been reduced to a greater extend when compared to those obtained before optimization.

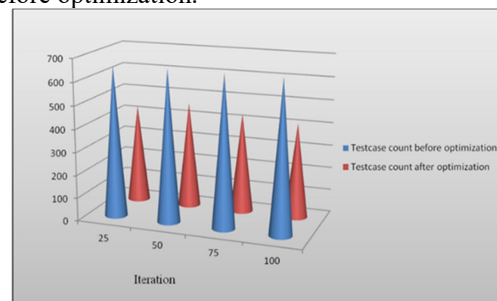


Fig 5: Graphical Representation For Test Count Value Obtained Before And After Optimization.

The fig 6 given below shows the graphical representation of Test case count for proposed and existing methods.

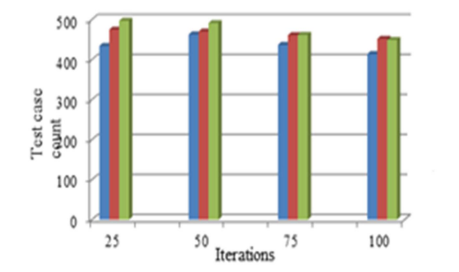


Fig 6: Graphical Representation For Test Case Count For Proposed And Existing Methods

The table 5 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 are reducing automatically.

The fig 7 given below shows the graphical representation of time usage for proposed and existing methods.

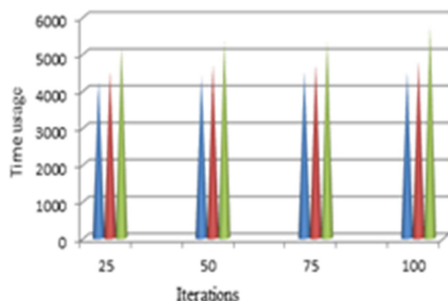


Fig 7: Graphical Representation Of Time Usage For Different Iterations

6. CONCLUSION

The paper proposes a creative method for minimizing interactive error that depends on the best experimental settings in the unsystematic experiment. According to the adaptive genetic algorithm, the original technique is used to minimize interactive errors. The best product is created using the differential genetic algorithm, which drastically reduces the forbidden inputs. The original strategy makes effective use of the coverage measures while keeping an eye out for problems.

The positive outcome demonstrates that the innovative process successfully overcomes the ambiguity of subjectively created experiment conditions and yields the best possible result. It is likely that the experiment condition allocation

metrics will be recognized in the coming days as a concept for choosing experiment conditions that will enable the faster achievement of higher coverage. To create error-free experiment circumstances, other, unquestionably more creative meta-heuristic research tools will be used.

8. FUTURE WORK:

The current study demonstrates the potential of Differential Genetic Algorithm (DGA)-based optimal directed random testing (ODRT) in reducing interactive faults by effectively exploring complex input spaces. However, there are several avenues for extending and improving this research.

First, the approach can be further optimized by integrating hybrid metaheuristics, combining DGA with other evolutionary or swarm-based algorithms such as Particle Swarm Optimization (PSO) or Ant Colony Optimization (ACO) to enhance convergence speed and diversity. This could lead to better fault detection in high-dimensional and non-linear input domains.

Second, the framework could be expanded to support automated test case generation for real-time and embedded systems, where constraints such as timing and resource utilization need to be considered. This requires extending the fitness function to account for temporal and performance-based criteria.

Third, scalability of the approach can be improved by parallelizing the DGA process on multi-core or distributed systems. Leveraging parallel computing could significantly reduce execution time, making the approach viable for large-scale industrial software systems.

Moreover, integrating machine learning techniques, particularly reinforcement learning or deep learning, could help predict fault-prone areas and guide the search process more intelligently. This synergy could increase the precision of test case selection and reduce the number of redundant or non-contributing test cases.

Finally, further empirical validation on diverse benchmark and real-world applications is essential. Comparing the performance of the DGA-based ODRT against other state-of-the-art testing strategies would help establish its effectiveness and generalizability. Additionally, developing a user-friendly tool or plugin for popular software testing environments would facilitate its adoption in practice.

Overall, these enhancements aim to make DGA-based ODRT more efficient, adaptive, and

applicable across a wider range of software domains and fault types.

REFERENCES

- [1] Zhou, Z. Q., Sinaga, A., & Susilo, W. (2012, January). On the fault-detection capabilities of adaptive random test case prioritization: Case studies with large test suites. In System Science (HICSS), 2012 45th Hawaii International Conference on (pp. 5584-5593). IEEE.
- [2] Niu, X., Wang, Y., & Wu, D. (2014, August). A Method to Generate Random Number for Cryptographic Application. In Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2014 Tenth International Conference on (pp. 235-238). IEEE.
- [3] Dionísio, J., Mota, T., Pinto, I., & Niehus, M. (2014). Real Time Random Number Generator Testing. *Procedia Technology*, 17, 534-541.
- [4] Malpani, P., & Bassi, P. (2014, September). Analytical & empirical analysis of external sorting algorithms. In Data Mining and Intelligent Computing (ICDMIC), 2014 International Conference on (pp. 1-6). IEEE.
- [5] Tahbaldar, H., & Kalita, B. (2011). Automated software test data generation: direction of research. *International Journal of Computer Science and Engineering Survey*, 2(1), 99-120.
- [6] McMinn, P. (2013). 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*, 55(1), 153-172.
- [7] Fraser, G., Arcuri, A., & McMinn, P. (2015). A memetic algorithm for whole test suite generation. *Journal of Systems and Software*, 103, 311-327.
- [8] Galeotti, J. P., Fraser, G., & Arcuri, A. (2013, November). Improving search-based test suite generation with dynamic symbolic execution. In Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on (pp. 360-369). IEEE.
- [9] Darab, M. A. D., & Chang, C. K. (2014, October). Black-box test data generation for gui testing. In Quality Software (QSIC), 2014 14th International Conference on (pp. 133-138). IEEE.
- [10] Putra, I. P. E. S., & Mursanto, P. (2013, September). Centroid Based Adaptive Random Testing for object oriented program. In Advanced Computer Science and Information Systems (ICACISIS), 2013 International Conference on (pp. 39-45). IEEE.
- [11] Chow, C., Chen, T. Y., & Tse, T. H. (2013, July). The art of divide and conquer: An innovative approach to improving the efficiency of adaptive random testing. In Quality Software (QSIC), 2013 13th International Conference on (pp. 268-275). IEEE.
- [12] Khan, S. A., & Nadeem, A. (2013, April). Automated test data generation for coupling based integration testing of object oriented programs using evolutionary approaches. In Information Technology: New Generations (ITNG), 2013 Tenth International Conference on (pp. 369-374). IEEE.
- [13] Campos, J., Abreu, R., Fraser, G., & d'Amorim, M. (2013, November). Entropy-based test generation for improved fault localization. In Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (pp. 257-267). IEEE Press.
- [14] Yu, B., & Pang, Z. (2012). Generating Test Data Based on Improved Uniform Design Strategy. *Physics Procedia*, 25, 1245-1252.
- [15] Padgham, L., Zhang, Z., Thangarajah, J., & Miller, T. (2013). Model-based test oracle generation for automated unit testing of agent systems. *IEEE Transactions on Software Engineering*, 39(9), 1230-1244.
- [16] Ahmed, B. S., Sahib, M. A., & Potrus, M. Y. (2014). Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing. *Engineering Science and Technology, an International Journal*, 17(4), 218-226.
- [17] Arcuri, A., & Briand, L. (2012). Formal analysis of the probability of interaction fault detection using random testing. *IEEE Transactions on Software Engineering*, 38(5), 1088-1099.
- [18] Minku, L. L., Sudholt, D., & Yao, X. (2014). Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. *IEEE Transactions on Software Engineering*, 40(1), 83-102.
- [19] Lv, J., Hu, H., Cai, K. Y., & Chen, T. Y. (2014). Adaptive and random partition software testing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(12), 1649-1664.

- [20] McMinn, P., Harman, M., Lakhota, K., Hassoun, Y., & Wegener, J. (2012). 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*, 38(2), 453-477.
- [21] Arcuri, A. (2012). A theoretical and empirical analysis of the role of test sequence length in software testing for structural coverage. *IEEE Transactions on Software Engineering*, 38(3), 497-519.
- [22] Kempka, J., McMinn, P., & Sudholt, D. (2013, July). A theoretical runtime and empirical analysis of different alternating variable searches for search-based testing. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation* (pp. 1445-1452). ACM.
- [23] Arts, T., Gerdes, A., & Kronqvist, M. (2013, September). Requirements on automatically generated random test cases. In *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on* (pp. 1347-1354). IEEE.
- [24] Barus, A. C., Chen, T. Y., Kuo, F. C., Liu, H., Merkel, R., & Rothermel, G. (2016). A cost-effective random testing method for programs with non-numeric inputs. *IEEE Transactions on Computers*, 65(12), 3509-3523.
- [25] Syed Abdul Moeed and Niranjan Polala, (2017) "Interactive Faults Detection (Ifd) In Acceptance Testing Based On Optimal Test Cases Using Gravitational Search Algorithm (Gsa)," *International Journal of Pure and Applied Mathematics Volume 115*(8), 559-564.
- [26] Chen, J., Kuo, F. C., Chen, T. Y., Towey, D., Su, C., & Huang, R. (2017). A Similarity Metric for the Inputs of OO Programs and Its Application in Adaptive Random Testing. *IEEE Transactions on Reliability*, 66(2), 373-402.
- [27] Scientific, Little Lion. "GENERATING OPTIMAL TEST CASES USING ELITIST GENETIC ALGORITHM." *Journal of Theoretical and Applied Information Technology* 103.8 (2025).
- [28] Rao, Kodepogu Koteswara, et al. "Prioritization of Test Cases in Software Testing Using M 2 H 2 Optimization." *International Journal of Modern Education and Computer Science* 13.5 (2022): 56.
- [29] Rao, K. K., M. Y. Saroja, and M. Babu. "Adaptive genetic algorithm (AGA) based optimal directed random testing for reducing interactive faults." *Indian Journal of Computer Science and Engineering* 12.2 (2021): 485-498.
- [30] Koteswara Rao, K., and G. S. V. P. Raju. "Reducing interactive fault proneness in software application using genetic algorithm based optimal directed random testing." *International Journal of Computers and Applications* 41.4 (2019): 296-305.
- [31] Rao, K. Koteswara, and G. S. V. P. Raju. "Random testing: the best coverage technique-an empirical proof." *International Journal of Software Engineering and Its Applications* 9.12 (2015): 115-122.
- [32] 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.
- [33] Rao, K. Koteswara, and G. S. V. P. Raju. "Theoretical investigations to random testing variants and its implications." *International Journal of Software Engineering and Its Applications* 9.5 (2015): 165-172.