Ingénierie des Systèmes d'Information
Vol. 28, No. 3, June, 2023, pp. 567-573
Journal homepage: http://iieta.org/journals/isi

Experimental Investigations to Fault Reduction System for Software Applications

Vemulapalli Rashmi[1], Chetla Chandra Mohan[1], Vasantha Bhavani[2], Yarlagadda Anuradha[3], Lella Kranthi Kumar[4], Battula Sowjanya[5], Kusuma Sundara Kumar[6], Kodepogu Koteswara Rao[7*], Anil Kumar Pallikonda[7]

[1] Department of IT, PVP Siddhartha Institute of Technology, Vijayawada 520007, India
[2] Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram 522302, India
[3] Department of CSE, Gayatri Vidya Parishad College of Engineering (A), Visakhapatnam 530048, India
[4] School of Computer Science and Engineering, VIT-AP University, Amravati 522237, India
[5] Computer Science and Engineering, Chebrolu Engineering College, Guntur 522212, India
[6] Civil Engineering, BVC Engineering College, Odalarevu 533210, India
[7] Department of CSE, PVP Siddhartha Institute of Technology, Vijayawada 520007, India

Corresponding Author Email: kkrao@pvpsiddhartha.ac.in

https://doi.org/10.18280/isi.280304

Received: 21 October 2022
Accepted: 20 January 2023

Keywords:
fault, AGA, test case, metrics, dependence model

ABSTRACT

A fault reduction system for software applications is put forward in direct random testing which includes two phases i.e., test case generation and false reduction. The system's primary components are a database, a false reduction mechanism, a test case creation mechanism, and an application selection mechanism. By using feature values from the input application, the test case generation method sets up an Object Behaviour Dependence Model (OBDM) to produce test cases. In addition to the indistinguishable inputs, the false reduction method configures an Adaptive Genetic Algorithm (AGA) to minimise the banned inputs. The AGA graciously accepts the exposure measurements of the experiment circumstances in order to considerably reduce the tendency to make mistakes.

1. INTRODUCTION

In recent years, continuous development of modern software resulted in increased need of software for businesses and individuals. The software coding involves faults which require software developers to fix the problems. Several testing techniques have evolved to repair the software faults. Black-box testing is an ad hoc testing method where programmes are examined by generating ad hoc and autonomous inputs. In the middle of several software testing procedures, random testing is the crucial strategy that is generally simple to apply. The software being tested can frequently be used in unexpected ways thanks to random testing, which is also effective in spotting errors. Current random testing is not very efficient in terms of time and money. Test cases produce failures but do not find errors immediately. The biggest drawback of random testing is the time it takes to generate test cases, which results in numerous inputs that are unlawful [1, 2].

Debugging is a time-consuming task in the development of software. Even though several mechanical techniques have been proposed, they are ineffective. Additionally, while physical debugging, breakpoint selection is difficult for developers. To address these issues and help developers locate errors quickly, interactive error localization techniques are used, which combine the benefits of mechanical approaches with physical debugging [3].

Before the error is discovered, the structure continuously suggests inspection locations based on the statements' uncertainties, which are planned in accordance with the implementation information of experiment conditions and the developer's response information at earlier checking points [4].

In general, the genetic algorithm represents an all-encompassing search technique that strengthens signal from the evolutionary data of inheritance. The iterations and the population in this process are represented by the production and the chromosomes, respectively. The meeting fee for the traditional GA is lower as compared to reality. As a result, it is important to create as many test cases as you can in a way that will enable you to find as many flaws and coverage targets as you can. An effective random testing test case is required. Through the best experiment condition in the direct random testing, interaction defects must be reduced. The production of test cases must be carried out using an effective procedural approach. An Adaptive Genetic Algorithm is needed to construct the finest product which pointedly moderates the prohibited inputs [5].

2. OBJECTIVES

The invention's primary objective is to provide software applications undergoing direct random testing with an effective fault reduction mechanism. Other goals include reducing interactive faults by employing the best experiment condition in direct random testing, generating as many test cases as possible that will help find as many faults as possible coverage targets, carrying out test case generation using an effective procedural model, and creating the best product to considerably minimise the forbidden inputs using an Adaptive Genetic Algorithm [6].

3. SUMMARY OF THE INVENTION IN THE PROPOSED RESEARCH

A failure reduction method for software applications is suggested by the innovation. In order to give a fundamental grasp of some features of the claimed subject matter, the following gives a condensed synopsis. This synopsis is not a thorough analysis. It's not meant to define the boundaries of the stated topic matter or to point out important or crucial components. Its main objective is to provide certain ideas just as a prologue to the eventual presentation of a more in-depth exposition [7].

According an aspect, the invention proposes a fault reduction system for software applications which comprises an application selection means, a test case generation means and a false reduction means. The application selection means is configured to select an input application from the database for software testing. The test case generation means is configured to generate test cases by means of feature value from the input application. The test case generation means utilizes Object Behavior Dependence Model (OBDM) to generate test cases [8].

The false reduction means is configured to reduce illegal inputs and equivalent inputs. The fault reduction system provides efficient software testing in direct random testing. The false reduction means utilizes Adaptive Genetic Algorithm (AGA) to reduce interactive faults. The Adaptive Genetic Algorithm utilizes Cauchy's mutation for adaptive behavior [9].

According to other aspect of the invention, test case generation phase using Object Behavior Dependence Model comprises assigning a function with a variable name, determining whether the function is previously called anywhere and storing the variable if called, checking for any if condition occurrence and assigning a value if occurred, assessing ratio for individual task and storing the destination variable, adding line coverage and loop coverage and finally terminating the test case generation if number of functions is less than maximum value otherwise repeating the procedure.

The proportion time one function spends calling another function to the overall amount of function time is what determines the ratio value for a single activity [10].

The percentage of lines that are exercised to all lines is known as line coverage. The task of determining and illuminating whether each loop body is executed either zero times, precisely once, or several times is allocated to the loop coverage [11].

According to another aspect of the invention, the false reduction phase using Adaptive Genetic Algorithm comprises creating populations of chromosomes, evaluating fitness value for each one limitation, selecting finest chromosome that contains utmost fitness value, implementing mutation and crossover operators, updating initial solution where innovative chromosome is generated and estimating the fitness value once again and finally electing the innovative chromosome as the finest chromosome if the fitness value of the innovative chromosome surpasses that of existing chromosome otherwise repeating the procedure. The chromosomes are the set of test cases generated [12].

4. DESCRIPTION OF DRAWINGS

The accompanying illustrations, which are integrated into and form a part of the specification, provide further explanation of the invention's concepts by illustrating a particular embodiment of the invention.

According to an illustrative embodiment of the invention, Figures 1 and 2 show a block diagram of the proposed fault reduction system for software applications. It indicates how the input application can be taken and phase 1 test case generation and phase 2 optimal test case generation.

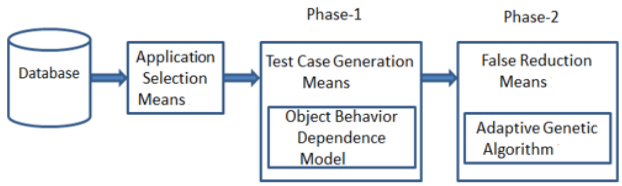

Figure 1. A method of modus operandi of the proposed fault reduction system

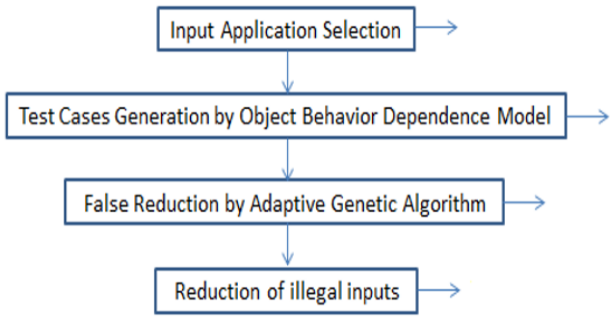

Figure 2. The flow diagram of the proposed method

The above figure explains how the phase 1 test cases can be Generated using OBDM.

The above figure explains about how optimal test cases can be generated using Adaptive Genetic Algorithm.

5. DETAILED DESCRIPTION OF DRAWINGS

With reference to the accompanying figures, one illustrative embodiment of the present invention will be discussed. In the illustrations and the description, the same or comparable reference numerals are used, if feasible, to refer to the same or similar parts or steps. Conferring to an archetypal embodiment, Figure 1 represents the block diagram of the fault reduction system for software applications. The system comprises a database, an application selection means, a test case generation means and a false reduction means. As depicted in the figure the fault reduction system is carried out in two phases i.e., through test case generation and false reduction. The phase 1 i.e., test case generation is carried out through test case generation means which utilizes Object Behavior Dependence Model (OBDM). The phase 2 i.e., false reduction is carried out through false reduction means 104 which utilizes Adaptive Genetic Algorithm (AGA) [13, 14].

Here, the number of functions utilised to create test cases is present for each input application. The proposed method determines the feature value based on the function value. The OBDM value is used to indicate this value. The OBDM technique fundamentally devotes its attention to the task and reporting metrics of the function that have been helpful for the experiment condition formulation, which departs in a significant way from intriguing the creation of the

reproduction and unsuitable test circumstances [15].

The fault reduction system's modus operandi is depicted in Figure 2. The input application is initially chosen from the database for software testing at phase. At phase, the Object Behavior Dependence Model (OBDM) is used to construct test cases after choosing the input application. Here, feature values from the input application are used to construct the test cases. The Adaptive Genetic Algorithm (AGA) performs false reduction at step after choosing the test cases. In order to decrease the unlawful and equivalent inputs at step, the optimum inputs are developed [16].

A flowchart for step 1, which involves creating test cases using an object behaviour dependency model, is shown in Figure 3 (OBDM).

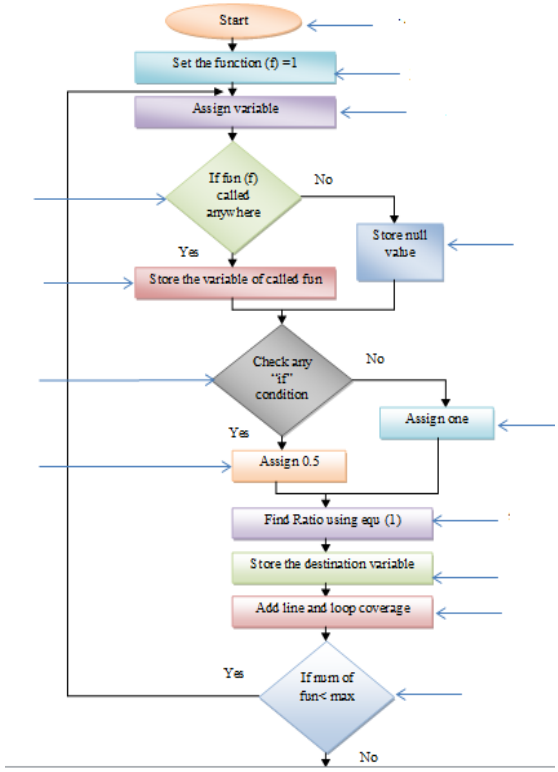

Figure 3. Flowchart for test case generation using OBDM according to an exemplary embodiment of the invention

Figure 3 pleasantly portrait the creation system of the experimentation condition and each one task is represented as the basis task. At step, the basis task of the test case generation starts. At step the function (f) is set as value 1. The function is then assigned with a variable name at step. Now, each function is experiential to determine whether it is previously fixed to definite supplementary mission i.e., whether the function called anywhere, at step If the function is called, the variable of the called function is stored at step otherwise a null value is stored at step depicting it as illogical. Further, a check is executed to determine whether any "if" condition is occurred at step. If yes, a value of 0.5 is assigned at step or else a value of one is distributed at step. At step, the ratio for the individual task is assessed as per Eq. (1) specified beneath and subsequently the target task is distinguished, trailed by the calculation of the whole experiment conditions to the coverage matrix.

$$Ratio = \frac{how\ much\ time\ call\ the\ other\ function}{Total\ function} \quad (1)$$

The ratio value is determined by how frequently a certain function calls another function relative to all functions combined. Eq. (1) specifies it. The destination variable is saved at step. Following that, at step, line coverage and loop coverage are added. The line coverage and loop coverage from the coverage metrics are used in the epoch-making procedure [17].

The line coverage is identified as a statement since it only includes the precise circumstances. Additionally, it estimates the code's prominence and ensures that various trails in the code in question flow smoothly. The ensuing Eq. (2) evaluates it.

$$Line\ coverage = \frac{number\ of\ lines\ exercised}{total\ number\ of\ lines} \quad (2)$$

The loop coverage is delegated by the mission of establishing and enlightening whether each one loop body is carried out either zero times, accurately on one occasion or numerous times. It also indicates if the loop body is correctly implemented only once or several times when 'do-while' loops are included. Additionally, the while-loops and for-loops generate a variety of presentations. The number of functions is then checked to see if it is less than the maximum value from step. If the answer to the condition is yes, the process starts over at the stage when the function is given a variable name; otherwise, the effort of creating the test ends at that point [18].

Examples include variable names supplied as E1, E2, E3, E4 and E5 for tasks A1, A2, B1, B2 and C1. The test case in the suggested technique includes the name of the source function, a probability value, a ratio value, the name of the destination function, line coverage, and loop coverage. Below is a technique for creating test cases along with an example.

Test case 1: [E1, -, 0.5, 2/5=0.4, E2] + LC + LPC
Test case 2: [E1, -, 0.5, 2/5=0.4, E4] + LC + LPC
Test case 3: [E2, -, 1, 1/5=0.2, E5] + LC + LPC

E1 stands for the starting function name in this example, and 0.5 denotes whether or not an if condition is present in the test case. If there is, 0.5 will be given. The value will be 1 if not. The following number, 2/5, represents the ratio value for the test case given in Eq. (1), and E2 stands for the name of the final function. For the purpose of creating test cases, these values are combined with those for line coverage and loop coverage. The same goes for all of the test scenarios.

Phase 2 is the next step, and this is where the false reduction is basically defined as the limitation of extra qualities that are not essential to the concept of dispensation. The Adaptive Genetic approach is tastefully used in the epoch-making process for the concept of false reduction, which is achieved in keeping by the optimization. The Adaptive Genetic Algorithm (AGA) explicitly relieves itself from its obligation to reduce erroneous inputs and inputs that are difficult to discriminate [19].

A meta-heuristic method for reducing the standard development system is distinguished by adaptive genetic algorithms. A flowchart for erroneous reduction using the suggested Adaptive Genetic Algorithm is shown in Figure 4. The populations of the chromosomes, (), are initially generated randomly. The population's dimension is indicated by the letter "N." The test scenarios generated arbitrarily are covered by the chromosome. The collection of test cases in this instance is the chromosome. The fitness value of each

limitation is then evaluated, as shown in the following Eq. (3).

$$fitness_i = \sum_{i=1}^{N} x^i_{if\ value} + x^i_{ratiovalue} + x^i_{linecoverage} + x^i_{loopcoverage} \quad (3)$$

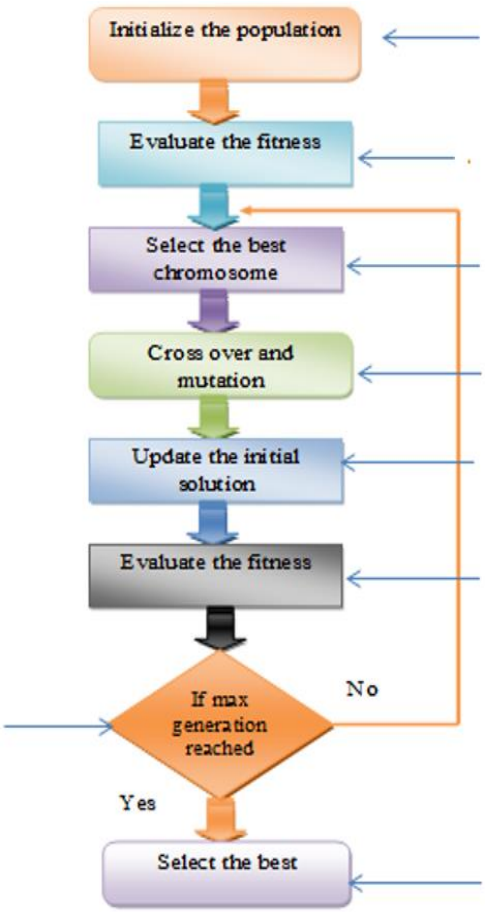

Figure 4. Flowchart for false reduction using Adaptive Genetic Algorithm according, to a representative embodiment of the invention

The metrics that are used to gauge fitness, such OBDM value, fault proneness ratio, line coverage, loop coverage, etc., have previously been computed for each test case. At step 403, the best chromosome is chosen as the one with the highest fitness value. In this case, maximising fitness means minimising interaction flaws. Crossover and mutation are two significant genetic operators introduced at step 404 that aid in solution convergence. In this case, the genetic algorithm uses Cauchy's mutation to increase adaptability. The entities are successfully changed using Eq. (4) by using the Cauchy transformation [20, 21].

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

The predetermined converting possibility implements the change. Illogical variable "x" represents a Cauchy allocation in the regions where the Cauchy change is realised. When the transformation work is completed, inventive chromosome is generated, and it then replaces the existing chromosome as the original solution is updated. The fitness value is once more estimated at step. If the fitness value of the novel chromosome exceeds that of the current chromosome, the novel chromosome is chosen as the best chromosome at step, indicating that the maximum generation has been attained. If not, the process starts over from the phase when the best chromosomes are chosen. As a result, using the best test cases, the Adaptive Genetic Algorithm may significantly reduce the interactive defects [22].

The chromosomal fitness values for the Adaptive Genetic Algorithm are calculated for dissimilar iterations, and the results are shown. The fitness value for the recommended approach, which has been shown to be superior to the technique using Genetic Algorithms, is shown in Table 1 below.

Table 1. Fitness value comparison

Iterations	Fitness values		
	AGA	GA	PSO
25	670	579	652
50	652	530	631
75	594	487	549
100	589	496	498

The graphical depiction of fitness value for various iterations utilising the adaptive and conventional genetic algorithms is shown in Figure 5. Plots of the fitness values produced from the Adaptive Genetic Algorithm and those from the Genetic Algorithm are shown below. It is clear from the graph that our suggested strategy, which uses an adaptive genetic algorithm, produces a higher fitness value than the genetic algorithm [23, 24].

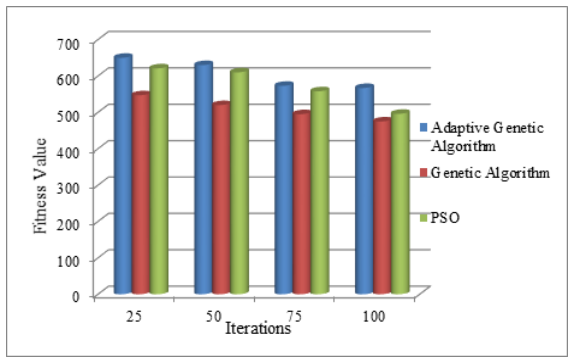

Figure 5. Graphical representation of fitness value using the adaptive and normal genetic algorithm

The test count value acquired before and after optimization is shown in Table 2 below. To choose the best test cases, the test case counts must be decreased. According to the discovered Test case count, the optimized result demonstrates that test cases that are irrelevant to the needed procedure are being disregarded.

Table 2. Test case count comparison

Iterations	Test case count	
	Without AGA	With AGA
25	696	454
50	696	445
75	696	459
100	696	458

The test count value acquired before and after optimization is represented graphically in Figure 6. Plots of the test cases produced before and after optimization may be seen here. The graph shows a clear reduction in the number of test cases when compared to results obtained prior to optimization.

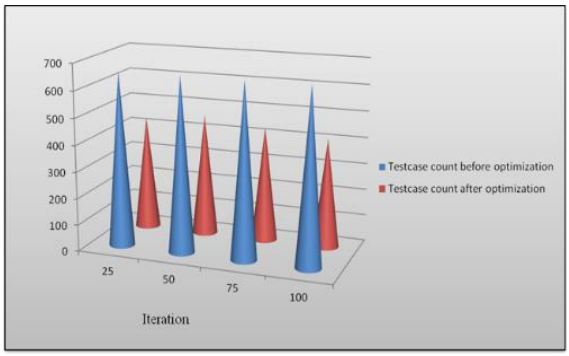

Figure 6. Graphical representation for test count value before and after optimization

The test case count acquired using various methods already in use and our suggested Adaptive Genetic Algorithm approach are both displayed in Table 3 below. Compared to previous methods, the test case count has increased.

Table 3. Test case count comparison

Iterations	Test case count		
	AGA	GA	PSO
25	454	496	505
50	445	491	509
75	459	484	512
100	458	473	501

The test case count for various iterations of the proposed and current methodologies is represented graphically in Figure 7.

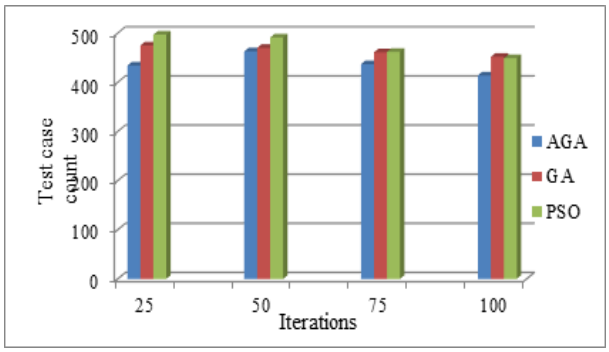

Figure 7. Graphical representation of test case count for proposed and existing methods

The suggested approach, Adaptive Genetic Algorithm, is shown in Table 4 below, along with its time and memory requirements for various iterations. The associated time and memory use for each iteration are computed, and the outcomes are tabulated. The number of interactive defects can be decreased while also decreasing memory use and execution time. The time and memory use automatically decrease with increasing iteration.

Table 4. Time and memory usage

Iteration	Time usage (sec)	Memory usage (Bytes)
25	5346	5432100
50	5288	5001399
75	5533	5876680
100	5503	5178817

The time utilisation derived from several currently used methodologies and our suggested Adaptive Genetic Algorithm method is shown in Table 5 below. When compared to other ways, the better time use was achieved. The difference between the process starts system time and the process end system time is the time utilisation that is computed.

Table 5. Time usage comparison

Iteration	Time usage (sec)		
	AGA	GA	PSO
25	5346	6985	7988
50	5288	6897	6964
75	5533	6835	6958
100	5503	6782	6795

The time utilisation for several iterations of the proposed and current approaches is represented graphically in Figure 8.

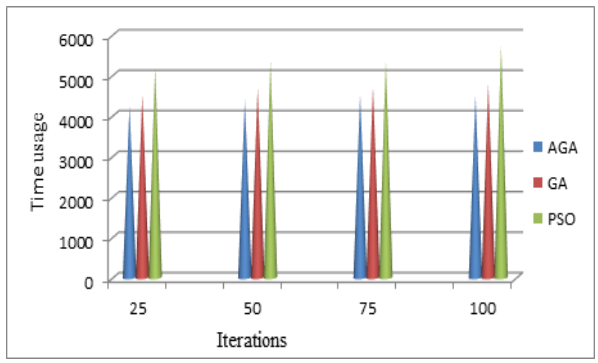

Figure 8. Graphical representation of time usage for proposed and existing methods for different iterations

The flaws found by the suggested method of random testing may be compared to those found by other testing methods, which are shown in Table 6 below.

Table 6. Comparison of regression and random testing

Testing methods	Faults detected (in %)
Random testing	91%
Regression testing	86%

Table 6 shows that interactive flaws are more easily found by random testing than through other types of testing, such as regression testing. Large test suites will produce better results for RT. Combinatorial Interaction Testing's applicability is examined in a novel way using the suggested method's real-time implementation (CIT). Thus, the invention proposes an efficient fault reduction system for software applications in direct random testing. By using the best experiment conditions for direct random testing, this approach lowers interactive software coding errors. This approach enables the creation of as many test cases as are necessary to identify as many coverage targets and errors as feasible. The system uses an effective Object Behaviour Dependence model to generate test cases. Using the suggested Adaptive Genetic Algorithm, a superior product is produced to considerably reduce the forbidden inputs. Numerous adjustments and variations can be made to the procedures outlined in the aforementioned examples without diverging from the invention's guiding principles, and this application is intended to cover all such modifications and alterations.

6. CONCLUSIONS

A fault reduction system for software applications comprising: An application selection means configured to select an input application from database for software testing; a test case generation means configured to generate test cases by means of feature value from said input application; a false reduction means configured to reduce illegal inputs and equivalent inputs; and whereby said fault reduction system provides an efficient software testing in direct random testing.

1. The fault reduction system for software applications as claimed as said test case generation means utilizes (OBDM) to generate test cases.
2. The fault reduction system for software applications as claimed as, said false reduction means utilizes Adaptive Genetic Algorithm (AGA) to reduce interactive faults.
3. (AGA) utilizes Cauchy's mutation for adaptive behavior.
4. A false reduction phase using Adaptive Genetic Algorithm comprises: creating populations of chromosomes; evaluating fitness value for each one limitation; Selecting finest chromosome that contains utmost fitness value; implementing mutation and crossover operators; updating initial solution where innovative chromosome is generated and estimating said fitness value once again; and electing said innovative chromosome as the finest chromosome if the fitness value of said innovative chromosome surpasses that of existing chromosome otherwise repeating the procedure.

A false reduction phase using Adaptive Genetic Algorithm as, said chromosomes are the set of test cases generated.

REFERENCES

[1] Zhou, Z.Q., Sinaga, A., Susilo, W. (2012). On the fault-detection capabilities of adaptive random test case prioritization: Case studies with large test suites. In 2012 45th Hawaii International Conference on System Sciences Maui, HI, USA, pp. 5584-5593. https://doi.org/10.1109/HICSS.2012.454

[2] Niu, X., Wang, Y., Wu, D. (2014). A method to generate random number for cryptographic application. In 2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing Kitakyushu, Japan, pp. 235-238. https://doi.org/10.1109/IIH-MSP.2014.65

[3] Dionísio, J., Mota, T., Pinto, I., Niehus, M. (2014). Real time random number generator testing. Procedia Technology, 17: 534-541. https://doi.org/10.1016/j.protcy.2014.10.207

[4] Malpani, P., Bassi, P. (2014). Analytical & empirical analysis of external sorting algorithms. In 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC) Delhi, India, pp. 1-6. https://doi.org/10.1109/ICDMIC.2014.6954224

[5] Tahbildar, 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. https://doi.org/10.1016/j.infsof.2012.03.010

[7] Fraser, G., Arcuri, A., McMinn, P. (2015). A memetic algorithm for whole test suite generation. Journal of Systems and Software, 103: 311-327. https://doi.org/10.1016/j.jss.2014.05.032

[8] Galeotti, J.P., Fraser, G., Arcuri, A. (2013). Improving search-based test suite generation with dynamic symbolic execution. In 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE) Pasadena, CA, USA, pp. 360-369. https://doi.org/10.1109/ISSRE.2013.6698889

[9] Darab, M.A.D., Chang, C.K. (2014). Black-box test data generation for Gui testing. In 2014 14th International Conference on Quality Software Allen, TX, USA, pp. 133-138. https://doi.org/10.1109/QSIC.2014.42

[10] Putra, I.P.E.S., Mursanto, P. (2013). Centroid based adaptive random testing for object oriented program. In 2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS) Sanur Bali, Indonesia, pp. 39-45. https://doi.org/10.1109/ICACSIS.2013.6761550

[11] Chow, C., Chen, T.Y., Tse, T.H. (2013). The ART of divide and conquer: An innovative approach to improving the efficiency of adaptive random testing. In 2013 13th International Conference on Quality Software Najing, China, pp. 268-275. https://doi.org/10.1109/QSIC.2013.19

[12] Khan, S.A., Nadeem, A. (2013). Automated test data generation for coupling based integration testing of object oriented programs using evolutionary approaches. In 2013 10th International Conference on Information Technology: New Generations Las Vegas, NV, USA, pp. 369-374. https://doi.org/10.1109/ITNG.2013.59

[13] Campos, J., Abreu, R., Fraser, G., d'Amorim, M. (2013). Entropy-based test generation for improved fault localization. In 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) Silicon Valley, CA, USA, pp. 257-267. https://doi.org/10.1109/ASE.2013.6693085

[14] Yu, B., Pang, Z. (2012). Generating test data based on improved uniform design strategy. Physics Procedia, 25: 1245-1252. https://doi.org/10.1016/j.phpro.2012.03.228

[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. https://doi.org/10.1109/TSE.2013.10

[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. https://doi.org/10.1016/j.jestch.2014.06.001

[17] Arcuri, A., Briand, L. (2011). Formal analysis of the probability of interaction fault detection using random testing. IEEE Transactions on Software Engineering, 38(5): 1088-1099. https://doi.org/10.1109/TSE.2011.85

[18] Minku, L.L., Sudholt, D., Yao, X. (2013). Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. IEEE Transactions on Software Engineering, 40(1): 83-102. https://doi.org/10.1109/TSE.2013.52

[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. https://doi.org/10.1109/TSMC.2014.2318019

[20] Arts, T., Gerdes, A., Kronqvist, M. (2013). Requirements on automatically generated random test cases. In 2013 Federated Conference on Computer Science and Information Systems Krakow, Poland, pp. 1347-1354. IEEE.

[21] 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. https://doi.org/10.1109/TC.2016.2547380

[22] McMinn, P., Harman, M., Lakhotia, K., Hassoun, Y., Wegener, J. (2011). 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. https://doi.org/10.1109/TSE.2011.18

[23] Arcuri, A. (2011). 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. https://doi.org/10.1109/TSE.2011.44

[24] Kempka, J., McMinn, P., Sudholt, D. (2013). 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, New York, NY, pp. 1445-1452. https://doi.org/10.1145/2463372.2463549