International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958 (Online), Volume-9 Issue-4, April 2020

Test Case Prioritization & Selection for an Object Oriented Software using Genetic Algorithm.

Prashant Vats, Manju Mandot, Saurabh Mukherjee, Neelam Sharma

Abstract: ***In this paper our aim is to propose a Test Case Selection and Prioritization technique for OOP for ordering the test cases as per in accordance with their priority for finding the faults in the OOS. We have used the heuristic Genetic Algorithm, in order to generating the order of these prioritized test cases for a given OOS. The motive is to put a test case first into the ordered sequence that may have the highest prospective of finding an error in the given OOS & then soon..***

Keywords: Test Case Selection, Test case prioritization, Genetic Algorithm, Fitness function, Object Oriented Software.

I. INTRODUCTION

Whenever we perform, testing onto the software during the development & maintenance phase, we want to make sure that after rerunning of all the existing test cases they should be able to detect the presence of errors if any. But it is quite tedious & a bit difficult to test the entire program under the peer pressure of meeting the deadlines for the delivery of a software project on time. So a better alternative for this to is that we should prioritize our test cases test cases for fault detection in such a manner that they should appear during execution in accordance to their relevant order of detection of probabilities of error in the given Object oriented software (OOS). It provides an ordered sequence of test cases that provides with a prioritized set of test cases, which are more likely to find errors in the given OOS. Now in order to propose a technique for ordering the test cases as per in accordance with their priority for finding the faults in the OOS, we would use the Genetic Algorithm, for generating the order of these prioritized test cases for a given OOS. The motive is to put a test case first into the ordered sequence which has the highest probability of finding an error in the given OOS & then soon.

Revised Manuscript Received on March 16, 2020.
* Correspondence Author
Prashant Vats*, AIMACT, Banasthali Vidyapith, Rajasthan, India. Email: prashantvats12345@gmail.com.
Manju Mandot, J.R.N. Rajasthan Vidyapith, Udaipur, Rajasthan, India. Email: Manju.mandot@gmail.com.
Saurabh Mukharjee, AIMACT, Banasthali Vidyapith, Rajasthan, India. Email: mukherjee.saurabh@rediffmail.com.
Neelam Sharma, AIMACT, Banasthali Vidyapith, Rajasthan, India. Email: sharmaneelam27@gmail.com

II. RELATED WORK FOR THE GA ON OOT

A Kumar, M., et al. [1] for evaluating the OOS has proposed a GA in tree representation by using class diagrams. In they have increased the efficiency of the OOS by solving the problem of optimization, thus facilitating effective code reusability with memory management.

For the Object-Oriented metrics Satish, et al. [2] has proposed software fault prediction models based on a GA based used in Fault based Testing. These predictive fault prone classes for a SUT are adaptable to OOS.

For aspect-oriented OOS based on GA, R. Delamare, et al. [3] has proposed a Fault based Testing approach for the class integration test order problem. By integrating the classes & aspects of the SUT based information with the information of class methods integration order is produced based on class aspects thus for the un-impacted classes in a SUT resulting in avoidance of the test case suite modifications. It can't be applied on large chunks of codes.

During integration testing, for measurement of inter- class coupling Jie F., et al. [4] has used GA by using minimal stubbing complexity.

Using a GA programming approach by implementing in Java for generation of test cases for classes in Evolutionary Testing at the unit level, Nirmal G., et al.[5] has given a method in OOS using statements in test cases in form of the tree representation.

Using the identification of path clusters by using GA Sabharwal S., et al. [6] in Rational Rose has done their work for the selection and prioritization of the test case scenarios. Considering the IF metrics & stack based memory allocation, using the State Dependency Graph & prioritization of the nodes of control flow graph they have addressed requirements change issues.

For model based cases in an OOS using GA for the automated generation of the test cases Chandran, K., et al. [7] has proposed their work onto prediction of internal and external stimuli based behavior of the objects which is dynamic in nature.

For symbolic execution & evolutionary testing of objects during Structural Testing of the OOS Inkumsah K., et al. [8] has proposed a framework at the integrated class level called Evacon uses GA algorithms to find method sequences with for a Software under Test (SUT) thus ensuring higher branch coverage.

For optimizing the test suites by using a coverage criterion, for GA, Fraser, G., et al. [9] has presented a search-based approach EVOSUITE for dynamic handling of test case dependencies among their predicates. During Integrated Class level testing at the early stages of design and analysis, for the Couple Based Testing, Alexneder C., et al. [10] uses the GA for attaining the optimization of single-objective for methods and attributes that are used in the classes of an OOS.

Using the Mutation based testing, to fit with test optimization by analyzing the application of GA, Franck F., et. al. [11] has attempted for the generation of test cases.

For Evolutionary Testing of the OOS, Kanmanani et al. [12] used Java to introduce a Class-Based Elitist involving GA resulting in achieving of faster results over the time.

III. KEY RESEARCH CONCEPT FOR GENERATION OF G.A.

Genetic algorithm (G.A.) is meta-heuristic search algorithms that are based on the ideas of selection of the fittest gene among a chromosome. In GA, the populations of chromosomes are denoted by various reassembling codes like as Binary, Permutation onto real world members by using genetic operators like Selection, Crossover or Mutation etc., which would be applied onto a participating chromosome in order to find the fitness function to decide that will decide the fittest chromosome which is nothing but just an objective function to decide that what number of prioritized test cases ensures hundred percent code coverage with maximum fault detection for a given OOS.

The GA provides a multidimensional search technique by using a combination of random iterative search methods that provides an optimized solution for a given problem. The GA method is indeed the most efficient featured algorithm that provides solution to search space based problem by considering a entire commutated population of a genetic chromosome.

The steps involved in the execution of GA are:

[1] Generate Population (Chromosomes).

[2] Find the fitness function of the proposed Genetic population.

[3] Apply the Selection, Crossover & Mutation process to determine the survival of the fittest one.

[4] Evaluate the Chromosome and reproduce it.

IV. GENERATING A CHROMOSOME POPULATION

Initially a GA Chromosomal function's population is randomly selected & encoded. Each Chromosome has denoted the possible answer to a given problem in order to arrange the test cases in a Chromosomal order & our motive is to optimize that genetic sequence. For e.g., we have got a following test sequence for a given set of N test cases where N=1 to n onwards.

Let us suppose N=10, so we may obtain the following genetic sequence:

T1->T3->T4->T6->T12->T5->T9->T17->T8->T13.

[1] Evaluation of the fitness of the Generated Prioritized test cases population. The fitness of a genetically populated chromosome can be defined by an objective fitness function. A fitness function will indicate the survival of a Chromosome into a good or bad. This objective fitness function will generate a sequence of number, consisting of all the prioritized test cases that will perform a comparative two or more chromosome.

[2] Apply Selection of test cases for individual Chromosome. In general, the selection of chromosomes will be dependent onto the fitness value of it. The possible chromosome with a higher or lower value would be chosen as a base for our problem definition.

[3] Applying the Crossover & mutation over a chosen gene. The parents of a gene will be chosen & combined in a random manner. This process of generation of genetic chromosome into a random order is called as crossover. There would be two types of crossover in genes:

1. Single point crossover
2. Multiple point based crossover.

For e.g. Given two sequences of test cases that has a high probability of detection of faults in an OOS. We have got two parents:

P1:

T1	T2	T3	T4	T5	T6	T7	T8	T9

P2:

T4	T2	T5	T3	T8	T1	T6	T9	T12

After using the one point genetic crossover, the resultant genetic offspring would be as follows:

CH1:

T1	T2	T3	T4	T8	T1	T6	T9	T12

CH2:

T4	T2	T5	T3	T5	T6	T7	T8	T9

For CH1, we will write the first portion of the P1 as it is in original form and after putting a constraint that for the second part of P2 we will not include a test case into CH1.

For the mutation onto any two genes, we have to select them randomly along with their Chromosome & then swap them randomly along with their Chromosome & then swap them with each other.

Ch1: T1->T2->T3->T4->T8->T7->T5->T9.

Ch2: T4->T2->T6->T7->T8->T1->T5->T9.

For Ch1, we will write the first portion of P1 as its in original form and after putting a constraint that for the second part of P2 we will not include test case into Ch1.

For the mutation onto any two genes we, have to select them randomly along with their chromosome & then swap them with each other.

For e.g. when T3 & T5 get selected randomly during mutation performed onto a chromosome.

Ch1: T1->T2->T3->T4->T8>T6->T9->T5->T7.

Ch2: T1->T2->T9->T4->T8->T6->T3->T5->T7.

V. TERMINATION CRITERIA.

The termination criteria could be selected in as many forms a may be defined like as:

1. Attainment of predefined value.
2. Number of generations of Chromosome.

In our approach we used a fixed generation number as a termination criteria. Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows:

1. Start with a randomly generated population of n l−bit chromosomes (candidate solutions to a problem).
2. Calculate the fitness $f(x)$ of each chromosome x in the population.

3. Repeat the following steps until n offspring have been created:

a. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.

b. With probability pc (the "crossover probability" or "crossover rate"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents.

c. Mutate the two offspring at each locus with probability pm (the mutation probability or mutation rate), and place the resulting chromosomes in the new population.

If n is odd, one new population member can be discarded at random.

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

VI. PROPOSED TECHNIQUE FOR TEST CASE PRIORITIZATION

For the test case prioritization & selection we have given the following technique:

1.Selection: The selection can be implemented as follows:

1 Evaluate the fitness, fi , of each individual in the population.

2 Compute the probability (slot size), pi , of selecting each member of the population:

$$pi = fi / \sum_{i=0}^{n} fj, where\ n\ is\ the\ pouplation\ size$$

3 Calculate the cumulative probability, qi , for each individual:

$$qi = \sum_{l=1}^{n} Pj .$$

4 Generate a uniform random number, r ∈ (0, 1].

5 If r < q1 then select the first chromosome, x1, else select the individual xi such that qi−1 < r ≤ qi .

6 Repeat steps 4–5 n times to create n candidates in the mating pool.

To illustrate, consider a population with five individuals (n = 5), with the fitness values as shown in the table below.

The total fitness,

$\sum_{i=1}^{n} fj$ f j= 28+18+14+9+26 = 95.

The probability of selecting an individual and the corresponding cumulative probabilities are also shown in the table 1.1 below.

2.Crossover : In this proposed paper we will use one point cross over with crossover probability Pc=0.33.

3.Mutation : In this paper we will use mutation probability Pm=0.2. it means that 20% of the genes will be muted within a chromosome.

Example -Test cases with execution history [1].

Test case ID	Principal	Rate	Time	Results	Execution History
T1	4814110.0	11.0	28.0	8944568.73	8, 9, 10, 11, 12, 13
T2	758794.0	2.0	4.0	821343.2	8, 9, 10, 11, 12, 13, 14, 15,16, 20, 21, 22
T3	359575.0	23.0	6.0	1214545.02	10, 11, 12, 13
T4	593972.0	6.0	30.0	3414472.90	10, 11, 12, 13, 14, 15, 16, 20, 21, 22

T5	160253.0	14.0	26.0	4234567.90	12, 13, 14, 15, 16, 20, 21,22
T6	971281.0	9.0	15.0	1431244.56	22, 23, 24, 25, 28
T7	141261.0	9.0	16.0	23567.34	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 21, 15, 16, 20, 21, 35
T8	8888800.0	10.0	5.0	12456.67	15, 16, 20, 21
T9	414831.0	24.0	4.0	1691530.98	5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 18, 19, 20, 21.

Table 2: are lines of code covered that covers by each test

Now we apply genetic algorithm, on this data.

The Table. 3 for displaying Results after applying GA for testing of OOP according to their Normalized value is given below.

	A	B	C	D
1	principle	rate	time	result
2	395421	14	19	4767074
3	816067	14	5	1571267
4	547008	13	24	10277236
5	841753	17	16	10379070
6	989990	14	10	3670112
7	638982	7	7	1026065
8	995530	7	1	1065217
9	623037	4	20	1365151
10	945517	6	20	3032401
11	635208	5	17	1455908
12	475902	16	12	2824967
13	916436	22	23	88792901
14	631151	2	25	1035470
15	102925	1	11	114830.2
16	915983	9	5	1409353
17	635918	11	5	1071559
18	800795	5	25	2711776
19	930607	9	14	3109835
20	230233	7	7	369703.9
21	224319	16	6	546530
22	932403	18	2	1298278
23	789096	25	20	68443168
24	218143	14	3	323188.5
25	884901	11	19	6427340

VII. EXPERIMENTAL RESULTS & SETUP.

The use of GA for the testing of OOP has provided improvised results as compared to the other techniques in terms of test case prioritization. The results are given in Table 4. & are shown in Fig. 1.

Test Cases	T1	T2	T3	T4	T5	T6	T7	T8
Running Test Case using without GA	4.3	4.8	5.9	6.7	8.3	8.5	7.3	6.8
Running Test Case using GA	5.6	7.5	6.8	5.3	7.1	6.4	5.3	4.9

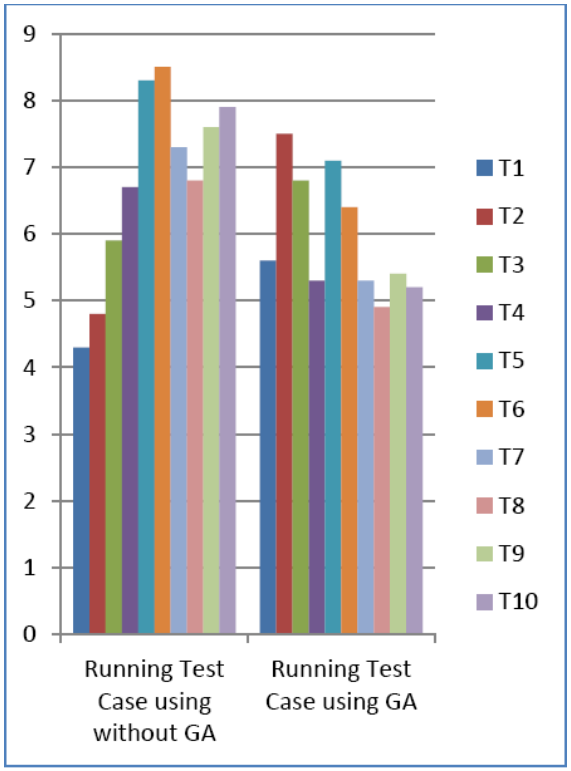

Fig. 1. Graph to show the execution of Test cases using GA & without GA.

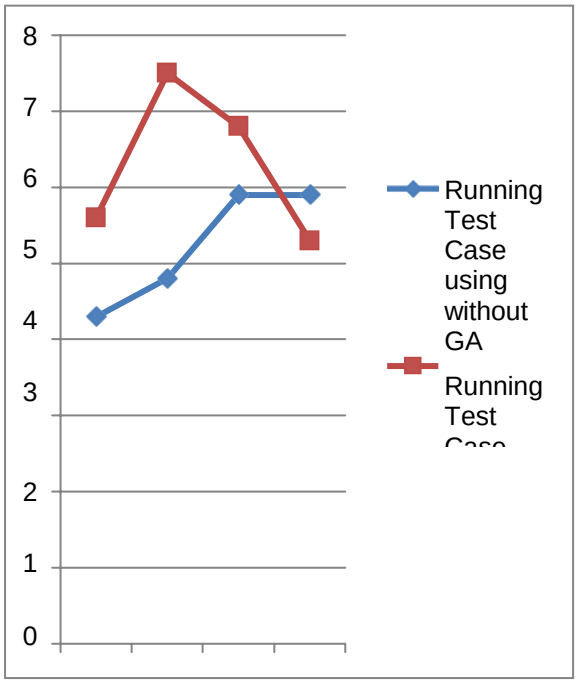

Fig. 2. Graph to show the code coverage during execution of Test cases using GA & without GA.

VII. CONCLUSION.

In this research paper we have provided a GA for test case prioritization using multidimensional search technique by using a combination of random iterative search methods that for provides an optimized solution for a given test case selection & their test case prioritization for a given Object oriented program. The GA method is indeed has proven it's worth as the most efficient featured algorithm that provides an optimized solution to search space based problem in the area of testing of OOP by considering a entire commutated population of a genetic chromosome.

REFERENCES:

1. Kumar, Manoj and Husain, Mh. (2011), "An Efficient Algorithm for Evaluation of Object-Oriented Models", Pub. in International Journal of Computer Applications, Vol. 24, No. 8, pp. 11–15, June 2011.
2. Sandhu, Parvinder S. and Dhiman, Satish Kumar (2009), "A Genetic Algorithm Based Classification Approach for Finding Fault Prone Classes", Pub. in Proceedings of World Academy of Science, Engineering and Technology, Vol. 60, pp. 485–488.
3. Delamare, Romain, Kraft, Nicholas A. (2012), "A Genetic Algorithm for Computing Class Integration Test Orders for Aspect-Oriented Systems", Pub. in IEEE Fifth International Conference on Software Testing, Verification and Validation (lCST), pp. 804–813.
4. Briand, Lionel C. and Feng, Jie (2002), "Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders", Pub. in Carleton University, Technical Report SCE-02-03, Version 3, October 2002.
5. Nirmal, K.G. and Mukesh, K.R. (2009), "Using Genetic Algorithm for Unit Testing of Object Oriented Software", Pub. in IJSSST, Vol. 10, No. 3, pp. 99–104.
6. Sabharwal, Sangeeta and Sibal, Ritu (2011), "Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams", Pub. in IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2, pp. 433–444, May 2011.
7. Prasanna, M. and Chandran, K.R. (2009), "Automatic Test Case Generation for UML Object Diagrams using Genetic Algorithm", Pub. in Int. J. Advance. Soft Comput. Appl., Vol. I, No. I, pp. 19–32, July 2009.
8. Inkumsah, Kobi and Xie, Tao (2008), "Improving Structural Testing of Object-Oriented Programs via Integrating Evolutionary Testing and Symbolic Execution", Pub. in Proceedings of 23rd IEEE! ACM International Conference on Automated Software Engineering, pp. 297– 306.
9. Fraser, Gordon and Arcuri, Andrea (2011), "Evolutionary Generation of Whole Test Suites", Pub. in Proceedings of 11th IEEE International Conference on Quality Software (QSIC), pp. 31–40.
10. Margaritis, B. and Alexander, C. (2010), "Placement of Entities in Objectoriented Systems by Means of a Single-objective Genetic Algorithm", Pub. in Proceedings of Fifth IEEE International Conference on Software Engineering Advances (ICSEA), pp. 70–75.
11. Baudry, Benoit and Fleurey, Franck (2005), "From Genetic to Bacteriological Algorithms for Mutation-based Testing", Pub. in Journal of Software Testing, Verification and Reliability, Vol. 15, pp. 73–96.
12. Maragathavalli, P. and Kanmani, S. (2012), "Multi- objective Genetic Algorithm using Class-Based Elitist Approach", Pub. in Computer Science & Engineering: An International Journal (CSElJ), Vol. 2, No. 5, pp. 3I–41 , October 2012.

AUTHORS PROFILE

Mr. Prashant Vats, is working in the field of CSE & IT as an Assistant Professor from past 11 years. He is pursuing Ph.D. in CSE from Banasthali Vidyapith, Rajasthan. He is a member of IEEE. He has contributed more than 35 publications in various National and International Journals, Conferences of International repute.

Prof. Manju Mandot, is a Professor and Director of Directorate of Jan Shikshan and Extension, J.R.N. Rajasthan Vidyapeeth (D) University. She completed her Ph.D (Computer Science) and has 27 years of teaching experience. Her research interest includes image processing, E- governance, women empowerment with technology. She is esteemed member of Computer Society of India.

Prof. Saurabh Mukherjee, currently works at the Department of Computer Science, Banasthali University. Saurabh does research in Human-computer Interaction and Computing in Mathematics, Natural Science, Engineering and Medicine.

Dr. Neelam Sharma, is an Assistant Professor at the Department of Computer Science, Banasthali University. She has completed her PhD in Computer Science and has 13 years of teaching experience and her research interest includes machine learning, pattern recognition.

Chromosome	Fitness value	Normalized value	Cumulative probability	Selection of random number	Recommendation
T1->T2->T3->T4->->T6->T7->T5->T8->T9->T10->T11->T12	196	196/5	0.342	0.3	Chromosome 1
T2->T4->T6->T8->T10->T12->T1->T3->T5-> T7->T9->T	189	189/5	0.671	0.4	Chromosome 2
T5->T6->T8->T9-> T12->T1->T7-> T3->T4->T10	188	188/5	1	0.2	Chromosome 1

Table. 3. Results after applying GA for testing of OOP according to their Normalized value