# Genetic-RTA: A Test Suite Automation for Concurrent Software Systems

S. Farhana
*PG Student, Department of CSE, Jawaharlal Nehru Technological University College of Engineering (Autonomous) Anantapuramu*

***Abstract*—It gets more difficult to evaluate the functionality of a software application as it grows because of addition of features, performance improvements, and quality enhancement, among other things. In testing, writing test cases and manually assessing those test cases by analysing the software application under test will cost a lot. Mostly, symbolic execution and genetic algorithms are often used to generate test cases. However, whatever test cases are made, are added to the original test suite, which may comprise a significant number of test cases, and used to use all those test cases in a sequential bid to achieve the coverage of targets in the program under test, which may cause a slight time delay. Moreover, if any further versions of that program version are patched in the future, and employ that newly produced test suite to compute on it, it may require a great deal of testing time and computational resources. By considering this problem, the paper proposed Genetic-RTA, to perform the good testing on the current program version as well as its upcoming modified versions in the future, it not only generates the new test cases but also collects the fittest test cases from it based on maximum coverage and minimum time consumption, from this, it forms the test suite that contains the ordered test case sequence as well.**

***Index Terms*—Regression testing, genetic algorithm, Genetic-RTA (Regression Test suite Automation).**

## I. INTRODUCTION

Software testing is used to verify the requirements that are employed on the software products. It detects the information about the level of quality by removing all the faults, errors, defects that are elevated in the software application. It is important to meet the customer requirements and expectations in terms of performance, design, dependability, and durability and the product's cost as well as testing should be done on every part and corner of the software application to get the good quality product be delivered.

As the software evolves, due to adding features, fixing bugs, improving design etc., so, it becomes difficult to revalidate the functionality of software applications. Normally regression testing is a dynamic testing used to evaluate the overall functionality whenever the code modification occurs. In regression testing, testers start the testing with executing test cases to which prioritization and selection of test case techniques are used to cut down on the expense of testing. Although, in some cases, previous test cases might not be sufficient but also need new test cases to examine the code or system behaviors that exist in the new versions of the software application.

Writing test cases and computing it manually by understanding the program application under test will cost much in testing. So, in order to generate the test cases automatically, there exists various techniques. In that, some test case generation techniques use the evolutionary approach such as Genetic Algorithms (GA) to produce new test cases [1] as well as in regression testing, test suite augmentation techniques are being used to discover code sections impacted by changes and create test cases to cover those areas [1,8]. Usually GA is the most widely used one of evolutionary algorithms, since, this shows the most accurate representation of natural evolution on computers [4] as well as it belongs to the family of optimization algorithms that is inspired from the biological techniques. The survival of the fittest concept is used in this community-based search technique. The new populations are generated by repeatedly applying genetic operators on the population's current members. Chromosome representation, selection, crossover, mutation, and fitness function computation are the core elements of GA [2].

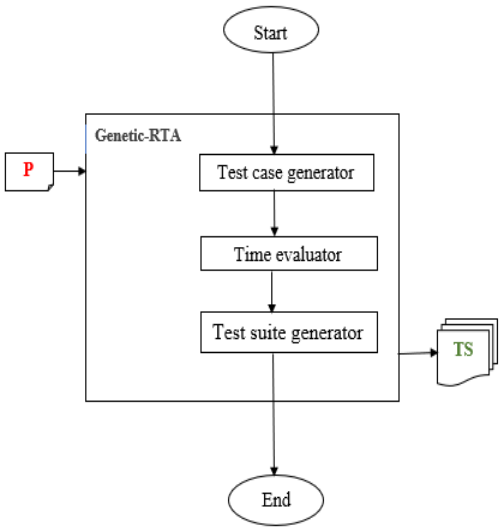


Fig. 1: Process flow of Genetic-RTA.

Prior techniques work in the form of, whatever the test cases that are generating, are added into the initial test suite which may form a bulk of test cases and used to pick those test cases in a sequential manner to meet the coverage of targets in the program under test, which may provide a bit delay in the form of time.

By considering the problem and to perform the good testing on the current program version as well as its upcoming modified versions in the future, it has to reduce the computation resources by collecting the finest test cases from the generated test cases for the better coverage of the program under test. Even though there is no extra time to compute all the test cases that exist in the test suite and to get the maximum coverage to that program version, the test cases must be arranged in the order for performing the test.

In this proposed work, Genetic-Regression Test suite Automation (Genetic-RTA) technique is used to reduce the test cost whenever the modified version of a program is under test, firstly, it uses the genetic algorithm in order to create new test cases, then it collects some of the feasible test cases from the newly formed test cases and places those test cases in an order on the basis of execution time and coverage by calculating the fitness to those generated test cases, such that a test suite be created.

## II. SOFTWARE TESTING

Software testing [5], the aim of software testing is to detect mistakes, and a good test is one that has a wide chance of finding an error. As a result, a software engineer should consider testability while designing and implementing a computer-based application program. Simultaneously, the tests must have a combination of qualities that enable it to identify the most faults with the least amount of work.

Each and every developing product can be assessed in two distinct ways: (1) realizing the every particular function that a product is programmed to deliver, tests can be performed to illustrate that function is operative while also looking for errors in every function, and (2) understanding the inner structure of a product, tests can be conducted to ensure that internal activities are performed as per specifications, and all system components are in working order.

```
1.  Let a, b, c, d, r1, r2 as double
2.  if a == 0 then
3.      r1 = r2 = 0
4.  else
5.      d = (b * b) - (4 * a * c)
6.      if d == 0 then
7.          r1 = r2 = -b / (2 *a)
8.      else
9.          if d > 0 then
10.             r1 = (- b + sqrt (d)) / (2 *a)
11.             r2 = (- b - sqrt (d)) / (2 *a)
12.         else
13.             t1 = -b / (2 *a)
14.             t2 = sqrt (-d) / (2 * a)
15.             r1 = t1 + t2
16.             r2 = t1 - t2
17.         end if
18.     end if
19. end if
```

Fig. 2: Quadratic equation program as motivating example

In this, functional testing is the first method, while structural testing is the second. The term "functional testing" refers to tests performed at the application interface. Black box testing is another name for this type of testing. It evaluates certain essential characteristics of a system while paying minimal attention to the software's core logical structure. Software is anticipated through structural testing, which involves a careful analysis of procedural detail. It's often referred to as "white box" testing. Test cases that exercise particular sets of criteria and/or loops are provided to exercise logical paths across the application and interactions between subsystems.

## III. GENETIC ALGORITHM

For machine learning, pattern classification, and optimization problems, genetic algorithms are frequently employed. Holland and his student introduced it in 1970, and it is mostly used for adaptive exploration and dynamic system design. To find a solution to an issue, GA employs reintegration and modification operators. The solution consists of numerical strings, such as bit-String, which represent the genes [6].

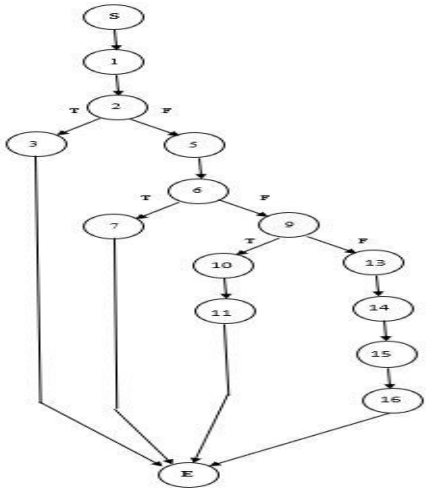


Fig. 3: Example CFG of quadratic equation.

Operators [3] that are widely used in the GA are encoding schemes, selection, crossover and mutation. These operators also include a variety of approaches and methodologies. The following is a quick rundown of the GA operators.

The first is encoding schemes, which play an essential role in most computational problems and demand that supplied information be encoded in a certain bit string, which is varied according to the domain issues. Binary, octal, hexadecimal, permutation, value-based, and tree are quite known encoding techniques.

Second is the selection, which is a crucial phase in genetic algorithms that decides whether or not a specific string will engage in the replication process. The reproduction operator is another term for the selection step, and the stability of GA is influenced by the evolutionary changes. Roulette wheel, rank, tournament, boltzmann, and stochastic universal sampling are some of the widely used selection strategies.

Table 1: Example paths for quadratic equation program.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Path1 | S | 1 | 2T | 3 | E | | | | | | |
| Path2 | S | 1 | 2F | 5 | 6T | 7 | E | | | | |
| Path3 | S | 1 | 2F | 5 | 6F | 9T | 10 | 11 | E | | |
| Path4 | S | 1 | 2F | 5 | 6F | 9F | 13 | 14 | 15 | 16 | E |

Third, crossover, which is a technique for integrating the specific genes of two or even more parents to produce offspring. Single point, two-point, k-point, uniform, partially matched, order, precedence preserving crossover, shuffle, reduced surrogate, and cycle are often used crossover operators.

Finally, mutation, it ensures that genetic diversification is maintained from one group towards the next. Displacement, simple inversion, and scramble mutation are three quite known mutation operators. Example of a basic genetic algorithm in activity is in the following.

1) Create a starting number of individual populations.
2) Using the fitness function, assess the fitness of every individual in the population.
3) Keep looping until the termination condition isn't met:
   a) Pick individuals out from population who look to be better in the form of fitness than the rest of the population.
   b) Reassemble the people that were chosen in the first place (a).
   c) Individuals be altered.
   d) Make a new population of people.

## IV. RELATED WORK

T. Yu et al. featured ConTesa, a concurrent software tool for enhancing regression tests that may reuse current test suites besides newly generated test cases. It handles the issues of test input creation and interleaving discovery in the same way, with fresh test inputs created through test reuse guiding investigation of the effected interleaving space which isn't yet covered by existing inputs. It may also use an active scheduler to replay regression concurrency problems. On a collection of multithreaded Linux apps, tested on ConTesa. Regarding execution speed, coverage of testing and fault detection capabilities, their findings demonstrate that it trumps state-of-the-art approaches.

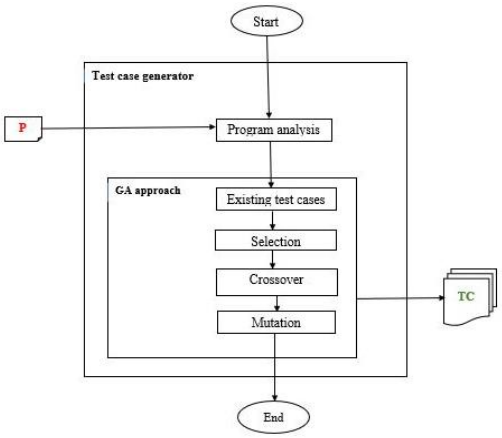


Fig. 4: Process flow of test case generator

Z. Xu et al. provided a new structured technique for augmentation integrating different test case generation algorithms and varied configurations of possibly important elements and presented the findings of the first controlled experiment on test suite augmentation, as well as the first experiment of its kind to assess alternative test case creation approaches in the context of augmentation (genetic and concolic). Their findings contribute to the growing body of evidence that guided test suite augmentation approaches can be successful. The findings indicate important considerations for investigators and experimentalists when developing and testing guided test suite augmentation approaches.

N. Chuaychoo et al. presented a genetic algorithm-based approach for generating test cases, in this the paths of the graph are determined by following path coverage criteria after a flow graph is constructed from the application source code. The genetic algorithm is then utilized to create test inputs. Finally, employed mutation testing to assess the test cases' efficiency. Experiments demonstrate that the tests provided by their technique are quite good at detecting flaws as well as based on the suggested technique, this research has built a test case generating tool.

Rijwan et al. proposed a novel approach for generating better automated test cases that combines mutation analysis with a genetic algorithm. GA operations mutation and Crossover are used to create these additional test cases. When calculating randomly produced test cases, the mutation score was very low, but when GA was applied to randomly generated test cases, the mutation score improved.

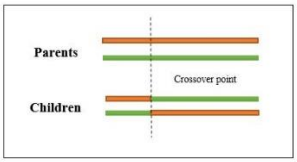


Fig. 5: Example for single point crossover technique.

X. Yao et al. constructed a mathematical methodology for creating data for multiple route coverage testing then, to solve the established model, a multi population individual sharing genetic algorithm is provided, and the performance of the suggested technique is not only theoretically examined, but also to a variety of applications that are being tested. The results of the experiments demonstrated that the suggested technique greatly improves the rapidity with which test data may be generated for a variety of paths.

Alsmadi developed and analysed a test case generation approach based on genetic algorithm principles to create test cases with high coverage of the paths tested or visited within the application and was able to test the overall sequence created by each test case by encoding the position of the controls (in contrast to the chromosomes) and displaying it in binary code. The aim the work has chosen is to generate a "new" test case each time.

## V. PROPOSED WORK

In this proposed work, the algorithm shows the procedure of Genetic-RTA, fig.1 illustrates the process flow of the proposed system. It consists of three main components, test case generator, for generating test cases, time evaluator, for evaluating time, and test suite generator, for generating test suite. In the proposed work, genetic operators used are: selection, to select test cases. Crossover, to produce offspring. Mutation, to mutate the generated test cases. Fitness function, to collect the fittest ones. Quadratic equation program, fig.2 is represented as the motivating example, initially, it identifies the paths that exists in the program by analyzing each instruction and generates the Control Flow Graph (CFG) that consists of nodes and edges (line 1), fig.3 depicts its CFG and Table 1 shows its example paths of a program.

**Algorithm:** Genetic-RTA

**Input:** Program PG, existing test cases ET

**Output:** A new test suite T

```
1.  Identify the path
2.  T_cr = ET // a set of existing test cases
3.  T = {ϕ} // a set of newly generated test cases
4.  P = T_cr // population P
5.  i = 0
6.  loop
7.      P = Roulette wheel selection(P)
8.      P = Single point crossover(P)
9.      P = Simple inversion mutation(P)
10.     for each t in P
11.         Concrete_execution(PG, t)
12.         if t covers paths
13.             t_s = Fitness(t) // collection of test
                cases for test suite t_s
14.             T = T + t_s
15.         end if
16.     end for
17.     i = i + 1
18. until i >= n
19. arrange the collected test cases in an
    ascending order
20. return T
```

Test case generator generates the test cases in two steps, as fig. 4 illustrates its process flow, firstly, it performs the program analysis [6,7]. In the second step, it uses the genetic algorithm approach to generate the test cases based on binary and value encoding scheme. In this, existing test cases are used as the initial population, then a selection process is performed to select the test cases from the initial test cases by using roulette wheel selection technique (line 7). After selecting, perform crossover on the selected test cases to generate new test cases using single point crossover technique (line 8), fig.5 shows the example of crossover that, at a random location on both parent's genes, is chosen as the 'crossover point.' Between both the parent genes, bits towards the right from that location are exchanged. As a result, two individuals are generated, with each some genetic variation from both parents. To that, mutation process is performed to mutating the genes of the generated test cases for maintaining the diversity between the test cases by flipping of bits (at random positions) by using simple inversion mutation technique (fig. 6). From this process, new test cases are generated.

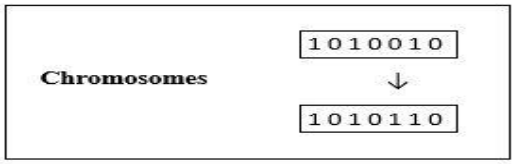


Fig. 6: Example for simple inversion mutation technique

Time evaluator evaluates the time by assessing test cases in the concrete execution of the program under test and updates whether the path is covered or not. From this, it extracts the real time entry and exit of the test case. Similarly, it evaluates time for all the newly generated test cases and updates the status as covered or uncovered based on the program coverage (line 11).

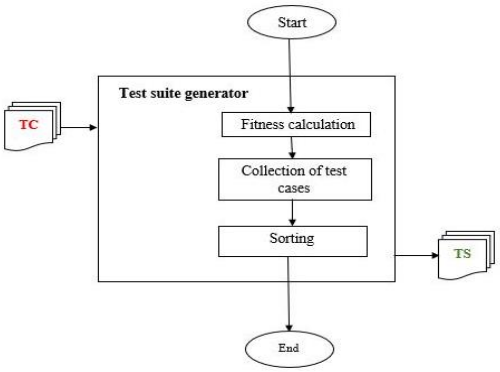


Fig. 7: Process flow of test suite generator.

The test suite generator, fig. 7 illustrates its process flow, that generates the test suite, it calculates the fitness to the generated test cases which are covered by using the fitness function (line 13) in terms of maximum program coverage and minimum time in the first step. In the second step, it collects the fittest test cases from the first step and then in the third step it arranges those collected test cases in an order on the basis of time to reduce the testing cost of the program. From this process, it yields the test suite that can reduce the testing cost whenever the updated versions are tested, which helps the testers to consume less time in testing the product.

## VI. EXPERIMENTAL SETUP

In this paper, the experimental setup was made on the 7 programs in which 6 are of the sequential programs and 1 is of the concurrent program to address our approach. Programs are listed in the following with its brief description.

Triangle classifier: the program determines whether or not the triangle's sides form a triangle. It identifies the triangle is isosceles, equilateral, or scalene assuming together make a triangle. Quadratic equation: the program takes three inputs and determines if the inputs may be combined to make a quadratic equation or not. It also discovers the roots of the problem if it constitutes a quadratic equation. Point circle: this program receives the positions of a point in the x-y Cartesian plane as well as the radius of a circle with the origin as its center. Then it determines if the point is within, outside, or on the circle's perimeter. Quadrant: the program intakes the points that coordinates and calculates the quadrant wherein the point lies. Even odd: the program consists of two threads that contain the logic of displaying even, odd numbers respectively, these are intercommunicated with one variable and provide the output accordingly.

Character classifier: the program that consists of one variable that can be determined whether the given character value is either integer or lower case or upper case. Anagram: the program contains two input variables which are of string data type, determines whether the given string is anagram or not.

Table 2: Identified input variables and paths by Genetic-RTA

| S.no | Programs | Predicted input variables | Identified input variables | Identified paths |
|---|---|---|---|---|
| 1 | Triangle classifier | 3 | 3 | 8 |
| 2 | Quadratic equation | 3 | 3 | 4 |
| 3 | Point circle | 3 | 3 | 3 |
| 4 | Quadrant | 2 | 2 | 4 |
| 5 | Even odd | 1 | 1 | 2 |
| 6 | Character Classifier | 1 | 1 | 4 |
| 7 | Anagram | 2 | 2 | 2 |

Table 3: Program coverage of total test cases vs selected test cases.

| S.no | Programs | Total test cases vs Selected test cases (Coverage) |
|---|---|---|
| 1 | Triangle classifier | = |
| 2 | Quadratic equation | = |
| 3 | Point circle | = |
| 4 | Quadrant | = |
| 5 | Even odd | = |
| 6 | Character Classifier | = |
| 7 | Anagram | = |

This approach generated the test cases for all the above programs, initially, the existing test cases should be given and then repeatedly after testing the program, the previously generated test suite can be considered to choose the parents for generating additional test cases, that gives the highest possibility of obtaining the optimized test suite.

As well as, when considering all these test cases that are generated, gave the maximum coverage of the program, besides, the fact of capturing point is the selected test cases that are considered fittest, also provide the same coverage with the least amount of time.

Table 2, illustrates the variables that are predicted from the program manually in column 2, in column 3,4 the identified variables and paths for the programs under test by Genetic-RTA respectively. Table 3 depicts the program coverage that is obtained from this approach, which gives the equal (=) coverage for both the total test cases and the selected test cases in column 3, table 4 shows the total time taken for concrete execution of programs under test that presents the difference between both the perceptions.

Table 4: Total execution time for both total test cases and selected test cases.

| S.no | Programs | Total test cases (Execution time in sec) | Selected test cases (Execution time in sec) |
|---|---|---|---|
| 1 | Triangle classifier | 8.5666E-3 | 8.724E-4 |
| 2 | Quadratic equation | 5.2116E-3 | 3.125E-4 |
| 3 | Point circle | 3.629E-3 | 2.064E-4 |
| 4 | Quadrant | 4.533E-3 | 2.463E-4 |
| 5 | Even odd | 0.4291 | 0.0313 |
| 6 | Character Classifier | 4.502E-3 | 5.532E-4 |
| 7 | Anagram | 0.01972 | 5.920E-5 |

Table 5: Comparison of coverage and execution time for existing vs new method.

| S.no | Programs | Existing vs New method (Coverage ↑) | Existing vs New method (Executing Time in sec ↓) |
|---|---|---|---|
| 1 | Triangle classifier | = | N (1.08E-3) |

| | | | |
|---|---|---|---|
| 2 | Quadratic equation | = | N (4.01E-3) |
| 3 | Point circle | = | N (3.04E-3) |
| 4 | Quadrant | = | N (2.77E-4) |
| 5 | Even odd | = | N (1.12E-3) |
| 6 | Character Classifier | = | N (1.41E-3) |
| 7 | Anagram | = | N (1.72E-4) |

such as total generated test cases and the selected test cases which consists of fittest test cases in column 3 and 4 respectively. Table 5 shows the coverage of existing vs new methods in which '=' describes the two provides equal coverage in column 3, where as in the column 4, 'N' shows that the new methods consumed less time than the existing with the seconds shown in the parenthesis.

## VII. CONCLUSIONS AND FUTURE WORK

This paper proposed a technique called Genetic-RTA, a test suite automation based on the Regression Test suite Augmentation (RTA) technique, to perform good testing on the current program version as well as future modified versions; it not only creates additional test cases by using the genetic algorithm but also collects the best test cases from it, regarding the maximum coverage and minimum time consumption using fitness function; through this, it forms the new test suite that produces the ordered test case sequence which helps the tester to consume less testing cost in the initial stages itself. Genetic-RTA generates the test cases in two steps, in the first, it analyses the program and generates the path and in the second step it generates the test cases based on the binary and value based encoding schemes using the genetic algorithm, and then it employs the newly generated test cases on the program under test in concrete execution and extracts the testing time, updates whether the program is covered or not for every test case. In the last, it collects the fittest test cases by using the fitness function on the metrics of time and coverage and then it provides those test cases in an ordered sequence on the basis of time to form a test suite that is capable of consuming less time with the less utility of test cases as well as with the high coverage. Genetic-RTA gave satisfied results in generating the test cases as well as defining the ordered sequence of test cases in less time which made the testing cost less expensive for the programs under test.

Other encoding schemes like octal and hexadecimal are also used in generating test inputs, as in the future, the current work can be extended to generate more testcases using those encoding schemes using genetic algorithm.

## REFERENCES

[1] Z. Xu, Y. Kim, M. Kim, G. Rothermel, and M. B. Cohen, "Directed test suite augmentation: Techniques and tradeoffs," in Proc. 18th ACM SIGSOFT Int. Symp. Found. Softw. Eng., 2010, pp.257-266.

[2] M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," in Computer, vol. 27, no. 6, pp. 17-26, June 1994, doi: 10.1109/2.294849.

[3] Katoch, S., Chauhan, S.S. & Kumar, V. A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80, 8091–8126 (2021). https://doi.org/10.1007/s11042-020-10139-6.

[4] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 2016, pp. 261-265, doi: 10.1109/ICGTSPICC.2016.7955308.

[5] R. S. Pressman, "Software Engineering: A Practitioner's Approach," 6th Edition, McGraw-Hill Publication, New York, 2005.

[6] Ruchika Malhotra, Chand Anand, Nikita Jain and Apoorva Mittal. Article: Comparison of Search based Techniques for Automated Test Data Generation. International Journal of Computer Applications 95(23): 4-8, June 2014.

[7] Chuaychoo, Nuntanee and Supaporn Kansomkeat. "Path Coverage Test Case Generation Using Genetic Algorithms." Journal of Telecommunication, Electronic and Computer Engineering 9 (2017): 115-119.

[8] T. Yu, Z. Huang and C. Wang, "ConTesa: Directed Test Suite Augmentation for Concurrent Software" in IEEE Transactions on Software Engineering, vol. 46, no. 04, pp. 405-419, 2020.