ISSN: 2321-7782 (Online)
Volume 1, Issue 6, November 2013
International Journal of Advance Research in Computer Science and Management Studies
Research Paper
Available online at: www.ijarcsms.com

Reducing the size of test suite using a variant of Non-dominated Sorting Genetic Algorithm II

Gurucharan Mandi[1]
Department of Computer Science of Engineering
MNNIT
Allahabad - India

Pranav Kumar[2]
Department of Computer Science of Engineering
ISM
Dhanbad - India

***Abstract:** For software development life cycle (SDLC) software testing is one of the essential part. The basic objective for software testing is to find the error that is not discovered yet. It also increases the software reliability. In the last few years, there is tremendous growth in the field of software. The software is getting large and complex day by day. It will take a life long time if one has to check the source code of a program or a huge man force. It is a kind of complex, labor-intensive, and time consuming work; it accounts for approximately 50% of the cost of a software system development. But, both of the above mentioned solutions do not seem feasible in view of todays world. Software testing takes a considerable amount of time and resources spent on producing software. Therefore, it would be useful to have ways to reduce the cost of software testing. One way to reduce amount of the time is automation of the whole testing process. Automation of software testing has various advantages in the software industry. The automation process for software testing reduces the time as well as the man force required in manual testing of the software. The one of the automation process of software testing is a selection of the test case form the large set of the test cases. As for large software there are a large number of test cases. Now, if we modified or upgrade the software then we have to test all the possible test cases for the software that is very time consuming for the large software. By using the automation of selection of the test case, our task is to make testing more efficient and test cases are one of the key factors for determining the quality of any software. Today, many selection techniques are available, but emerging trends in programming and the rising complexity in the source code and involvement of the object oriented program has increased the research work for this area particularly for researchers, practitioners and students. For selecting the test cases, we are using genetic algorithm. The NSGA II is one of the algorithms for optimization problems. For selecting the test case we proposed an approach. In the proposed approach, we randomly initialize the chromosome and calculate the fitness of each chromosome. By using NSGA-M we find the test cases that have minimum methods. When the number of iteration completed then we have got the test cases that have minimum fitness.*

Additionally the thesis also introduces the brief description of the software testing, NSGA II. At the last the experimental results of the proposed approach shown and then reduce the test suite, and also gives an idea about the future work related to this area for researchers and students.

***Keywords:** Test Case, Regression Testing, Multi-objective optimization, NSGA II.*

I. Introduction

Software testing is the process of experimenting a program with well designed input data to observe failures. Testing identifies faults, by removing faults it can increase the software quality. Testing also measures the software quality in terms of its capability for achieving correctness, reliability, usability, maintainability, reusability and testability. The various Objectives of testing are as follows:

- A good software testing is one which finds out a undiscovered error, which is not found yet.
- A good testing always aims at modification or some changes if required, thus by adding or deleting a value.
- Software reliability and quality are always based on the data collected during a testing process.

The various advantages of testing are as follows:

- Cost reduction
- Time reduction
- Defect reduction
- Increasing productivity of the Software developers

Given here an abstract view of flow of testing from the below give fig 1:

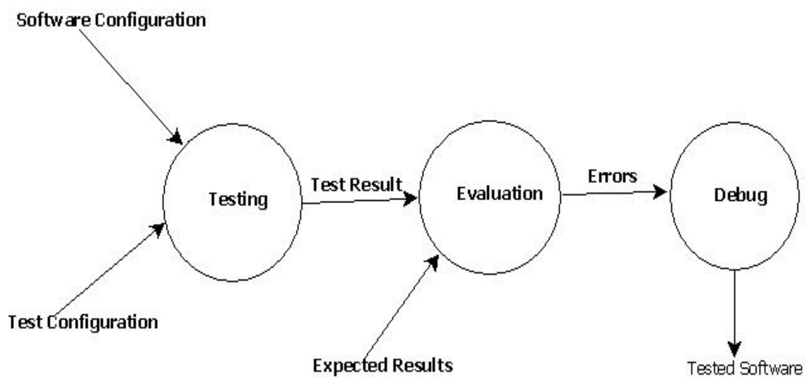

Fig 1: Testing Flow Information

Testing flow information graph is a technique of testing software which specifies the strategies to selects input test cases and analyzes test results [1]. There are mainly two types of testing techniques, they are:

- Functional testing
- Structural testing

Now a day's software is changing frequently, due to updating of technology regularly. Therefore, improvement of software is needed to cop up with the technologies. To improve software means to add some new feature or improve the performance of the software or fixed some defect in the software or modify the code to meet some requirement. When this software is modified then it require to test it fully or partially to check the errors. But if the software is large then to retest it fully, it is time consuming. So instead of running the entire test cases, it is better to run the selected part of the test case. To select the test case from the whole test cases is called selection problem. In this paper, we are giving the solution for selecting the test cases from the set of large test cases. We are using the NSGA-M for selecting the test cases for object oriented program and reducing the test suite.

II. Some Key Definition

Test Case

A test case is a combination of condition by which a tester can verify whether the software is correctly working or not, like running a particular with some specific argument [4].

Regression Testing

This is a type of software testing which find out new software bugs, from the existing functional or non-functional are of a system after the changes made on the software. By using this regression testing we can ensure that the above changes which we made into the software are not giving any new errors [7]. The main reasons for using regression testing are to see whether the changes made in one part of the software, is not affecting other parts of the software [8]. The common methods of using regression testing include retesting the previously completed tested software and check whether the changes made in the software are not giving any errors which are giving in the previously unchanged software. This testing can be done on any part of the software efficiently.

Multi-objective optimization

As the name suggests it deals with multiple objective for finding out an optimal solution. In these optimization problem their are involves multiple objectives, therefore a user never satisfied by finding one solution using single criteria. The presence of the conflicting objectives give a set of optimal solution called pareto optimal solution.

Evolutionary Algorithm (EA) is our natural choice for finding multi-objective optimization problems as it uses a population approach. Here non-dominated solution in a population has been used, for these numbers of multi-objective evolutionary algorithms have been suggested such as Pareto Archived Evolution Strategy (PAES), Non-dominated Sorting Genetic Algorithm (NSGA) etc.

Non Dominated Sorting Genetic Algorithm (NSGA II)

NSGA [9] is a popular non-domination based genetic algorithm for multi-objective optimization, and NSGA II is the modifies function of NSGA which give better sorting algorithm.

At first the population is initialized and after that the population is sorted based on non-domination into each front. These individual fronts are assigned rank or fitness value in which they belong. With this fitness value another new parameter called crowding distance is calculated for each individual. The crowding distance is a measure by how much it is close to the neighbors from the individual. Large the average crowding distance, the better will the result in diversity in the population.

Parents are selected from the population based on the rank and crowding distance by using binary tournament selection. An individual is selected if the rank is lesser than others or if crowding distance is greater than others. The crossover and mutation operators generate offsprings from the selected population.

Then the current population and current offspring'sare sortedd again based on non-domination, and only the best N (population size)is selectedd based on rank and crowding distance on the last front.

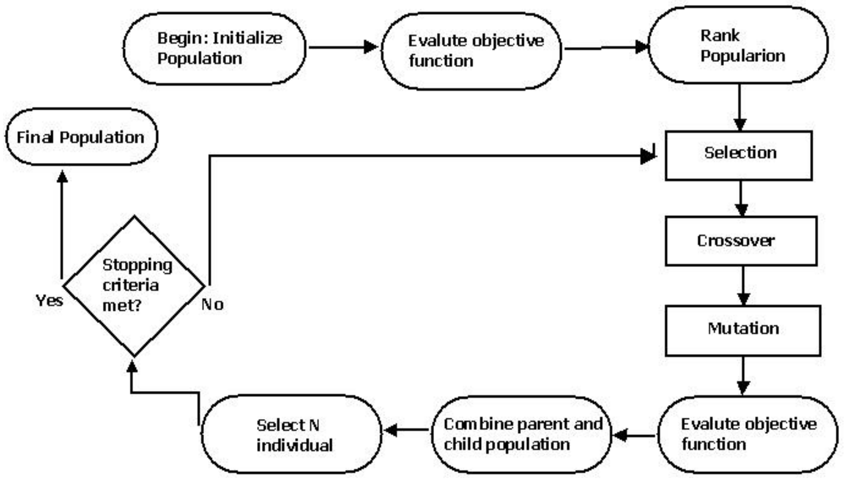
Begin: Initialize Population
Evalute objective function
Rank Popularion
Selection
Crossover
Mutation
Evalute objective function
Combine parent and child population
Select N individual
Stopping criteria met?
Yes
No
Final Population

Fig 2: Flow chart for NSGA II algorithm

Tools Used

The brief description of the tools used for executing the test cases and finding the coverage are given.

1. ***JUnit:*** JUnit is a unit testing framework for the Java Programming Language. It is developed by Kent Beck, Erich Gamma, David Saff, Mike Clark (University of Calgary). The stable release version is 4.11. The tool is written in Java language.

2. ***EMMA:*** EMMA is a tool for measuring coverage of Java software. Such a tool is essential for detecting dead code and verifying which parts of your application are actually exercised by your test suite and interactive use.

III. Previous Work

Software testing is the process of exercising a program with well designed input data with the intent of observing failures [10, 11, 12]. In other words, Software testing addresses the problem of effectively finding the difference between expected behavior specified by the system models and the observed behavior of the implemented system [13].

At present, software testing on the average makes up as much as 40% to 60% of the total development cost and would increase even further with rapidly increase size and complexity of software [10, 12, 14]. As systems are getting larger and more complex, the time and effort required for testing are expected to increase even further. Therefore, automatic software testing has become an urgent practical necessity to reduce testing cost and time. In this purview, we will start our discussion on theory and related survey on automatic test case generation and optimization of object-oriented software.

Chen et al. [15] discuss a specification-based (black box) method for regression test selection. Code-based regression test election is interesting, but as the size of the system grows it will be more difficult to do this test case selection by using code coverage.

Xu et al. [18] propose a causal model to the test case selection problem. According to them, if the company implements the known test practice recommended by independent verification and validation process the unavailability of source code at system level is true. In addition, different test engineers may have different ideas to perform a test case selection.

Yoo et al. [16] introduce the concept of Pareto efficiency to the test case selection problem. The Pareto approaches take multiple objective functions and construct a group of optimal solutions. In this case, each solution is the optimal test cases subsets. The paper instantiates the test case selection problem with two versions: the first one combines code coverage and cost as objective functions, and the second one combines code coverage, cost and fault history as objective functions.

Mansour et al. [17] compare five regression test selection algorithms, which include: Simulated Annealing, Reduction, Slicing, Dataflow and Firewall Algorithms. The objective function of all of them is based on code coverage.

IV. Proposed Work

In this dissertation NSGA-M algorithm is proposed. In the proposed algorithm, fitness has been calculated by giving weightage to the objective function and after that we are applying non-dominated sorting.

NSGA-M introduces the following modification:

- ***Change in the weightage algorithm:***
  1. Take two objective function fmax and fmin. Then take some values for both of them.
  2. Now, fixed the weightage for fmax but change for fmin.
  3. Change the lowest value in fmin with the summation of the total value and rest should be calculated as summation of total value divided by the original value. Then we are applying non-dominated sorting algorithm on it.
- ***Done some change on random number generator:***
  1. In old algorithm random number has been divided into two parts from the set of number.
  2. In new algorithm random number has been divided into five parts of the set of number. It helps in making the set simpler.

Algorithm of NSGA-M

Input: Size of the population (pop), generation, cross over probability, mutation probability, sigma-share value and objective function

Output: Optimum fitness value of function f(x).

1. NSGA-II classifies a population which is called fronts

2. The number of classes varies from generation to generation and the members in each class are equivalent

3. That is, it cannot be stated which individual is better.

4. Run the weightage function.

5. This classification which is called non-dominated sorting is implemented as follows.

6. All non-dominated individuals are classified into one category and assigned a dummy fitness value or rank.

7. This process continues until all members are classified.

8. Individuals of the first layer have the highest fitness while members of the last layer are assigned the smallest fitness.

9. An estimation of the density of solutions surrounding each member is calculated using the crowding distance

10. The population is sorted in ascending order.

11. All other solutions are assigned a distance value equal to the absolute difference in the function values of two adjacent solutions.

12. A non-dominated sorting is applied and a new population is formed.

13. A population of children Qt + 1fromPt + 1 is formed using a binary crowded tournament selection, crossover and mutation.

Experimental Result

Benchmarks Functions:

To compare the performance of the proposed NSGA-M algorithm with the previous NSGAII, four well known benchmark functions are used. Here, we have chosen ZDT problems as for the reasons of space; the four problems are ZDT1, ZDT2, ZDT3 and ZDT4. In this problem there are two objectives which is to be minimized:

$$f_1(\vec{x}) = x_1 \qquad ---------------(1)$$

$$f_2(\vec{x}) = g(\vec{x})h(f_1(\vec{x}), g(\vec{x})) \qquad ------------ (2)$$

ZDT1: In this problem there are 30 variable (n = 30) having a Pareto-optimal set, $x_1 \varepsilon[0,1]$:

$$g(\vec{x}) = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \qquad ----------------- (3)$$

$$h(f_1(\vec{x}), g(\vec{x})) = 1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}} \qquad --------------(4)$$

In this ZDT1 function the population is taken 100 and evaluation is 150. From this Pareto-optimal front appears when g = 1.0. From this the below figure is drawn and it is clearly show us that the modified NSGA-M is giving good pareto front than NSGA II.

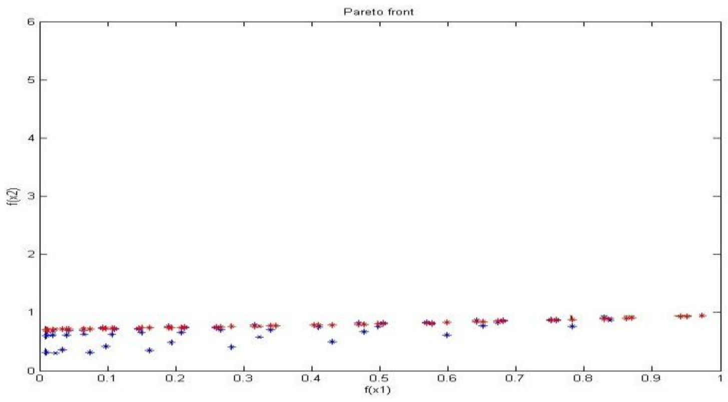

Fig 3: Comparison of NSGA II and Modified NSGA-M using function ZDT1, where blue one is NSGA-M and red one is NSGA II.

ZDT2: In this problem there are 30 variable (n = 30) having a Pareto-optimal set, $x_1 \epsilon [0,1]$:

$$g(\vec{x}) = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \qquad -------------------(5)$$

$$h(f_1(\vec{x}), g(\vec{x})) = 1 - \left(\frac{f_1(\vec{x})}{g(\vec{x})}\right)^2 \qquad ---------------(6)$$

In this ZDT2 function the population is taken 100 and evalution is 150. From this Pareto-optimal front appears when g = 1.0. From this the below figure is drawn and it is clearly show us that the modified NSGA-M is giving good pareto front than NSGA II.

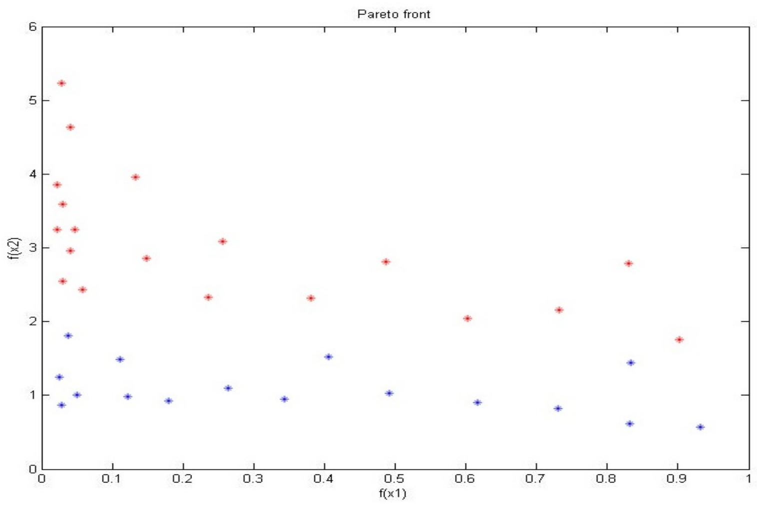

Fig 4: Comparison of NSGA II and Modified NSGA-M using function ZDT2, where blue one is NSGA-M and red one is NSGA II.

ZDT3: In this problem there are 30 variable (n = 30) having a Pareto-optimal set, $x_1 \varepsilon [0,1]$:

$$g(\vec{x}) = 1 + \frac{9\sum_{i=2}^{n} x_i}{n-1} \qquad -------------------(7)$$

$$h(f_1(\vec{x}), g(\vec{x})) = 1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}} - \left(\frac{f_1(\vec{x})}{g(\vec{x})}\right)\sin(10\pi f_1) \qquad ---------- (8)$$

In this ZDT3 function the population is taken 200 and evalution is 300. From this Pareto-optimal front appears when g = 1.0. From this the below figure is drawn and it is clearly show us that the modified NSGA-M is giving good pareto front than NSGA II.

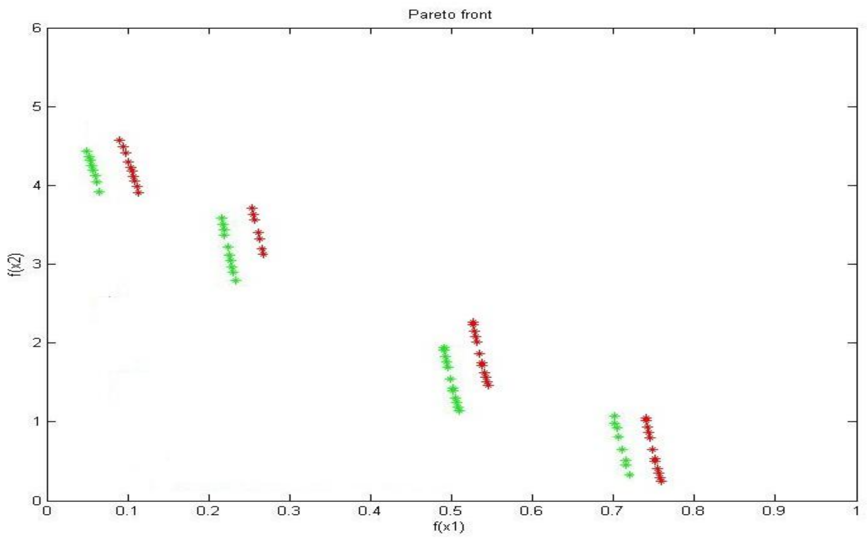

Fig 5: Comparison of NSGA II and Modified NSGA-M using function ZDT3, where green one is NSGA-M and red one is NSGA II.

ZDT4: In this problem there are 10 variable (n = 10) having a Pareto-optimal set, $x_1 \varepsilon[0,1]$ and x_2; $x_n \epsilon[-5,5]$:

$$g(\vec{x}) = 1 + 10(n-1) + \sum_{i=2}^{n} \left(x_i^2 - 10\cos(4\pi x_i)\right) \quad ------------(9)$$

$$h(f_1(\vec{x}), g(\vec{x})) = 1 - \sqrt{\frac{f_1(\vec{x})}{g(\vec{x})}} \quad ------------------(10)$$

In this ZDT4 function the population is taken 500 and evalution is 600. From this Pareto-optimal front appears when g = 1.0. From this the below figure is drawn and it is clearly show us that the modified NSGA-M is giving good pareto front than NSGA II.

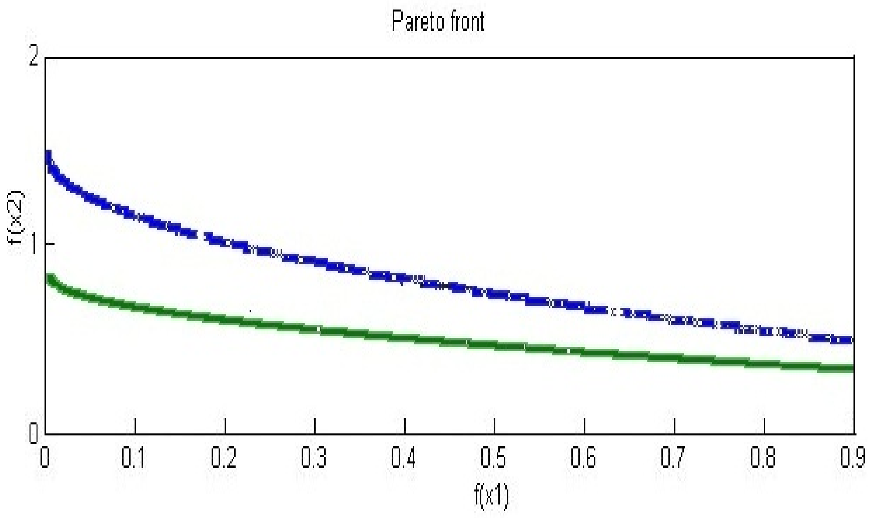

Fig 6: Comparison of NSGA II and Modified NSGA-M using function ZDT4, where green one is NSGA-M and blue one is NSGA II.

From above result, get from comparing proposed approach NSGA-M with previous algorithm NSGA II, by the benchmark functions ZDT, it is hear conclude that the proposed approach is giving good result. Therefore, we will used this algorithm for minimizing the test suit.

V. REDUCING OF TEST SUITE

Generation of Test Cases

The steps for generating test cases are given below:

1. The java source code is given as the input to the Branch Instrumentor module.
2. The Branch instrumentor module produces the instrumented version of the original class and performs these following tasks on the classes which are under test:
   a) Parsing the source code: Reads the source code of the classes which are under test and classifies each statement.
   b) Determines whole information about the constructor, method signature.
   c) Now, all the method of the given source file stored in an array.
3. Initially, the population is initialized randomly by binary value (either 0 or 1) with the random generator.
4. In the population or chromosome, binary value(0 or 1) are shown which means 1 tell us that the particular method is consider and 0 tell us that particular method is not consider while testing.
5. The fitness value for each chromosome is calculated as:
   a) The test case generator takes each chromosome and decoded chromosome and converted into constructor and method invocations.

b) In the test case the assertions are added. These assertion has two fields first one is expected value and the second one is value generates by the method.

c) The test case is executed using JUnit tool. The JUnit execute the test case and find whether the test case passed or failed.

d) If the test case executed by the JUnit is passed successfully then the coverage of the program is calculated by EMMA tool. The coverage value assign as the fitness of the chromosome.

6. Now, we select the chromosome by using one of the selection methods that is tournament selection technique.
7. If the number of iteration completed then print the population and we stop the algorithm. Else the next generation is calculated by using selection, mutation and crossover and again go to step 5.
8. The final population shows the test cases. Now, combine all the test cases to make a test suite and calculate the fitness of the test suite.

Minimizing the size of Test Suite

The steps for minimizing the size of the test suite:

1. Initially, the population or chromosomes are initializing randomly by binary value (either 0 or 1) with the random generator.
2. In the chromosome, 1 shows that particular test case considers while testing and 0 shows that particular test case not considering while testing.
3. The fitness value for each test suite is calculated as:

   a) The test case generator takes each chromosome and decoded test case. All the corresponding method of the test case called.

   b) The fitness of the whole test suite is calculated as discussed in the previous stage.
4. Now, we sort all the population and select the chromosome that has high fitness value i.e. coverage.
5. If we get a test suite having a fitness value greater or equal to the fitness value of the whole test suite then we store it.
6. If the number of iterations completed then we check the stored test suite. Else the next generation is calculated by using selection, mutation and crossover and again go to step 3.
7. In the stored test suite, print the test suite having the least number of test cases as the result.

Below table 1 shows the result for test case generation with population size = 10 and iteration = 10:

Table I: The generated test cases with fitness value.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**F**	**Time(sec)**
T0	1	0	0	1	0	0	1	1	1	0	0	1	1	0	0	70	0.29
T1	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	50	0.19
T2	0	0	0	1	1	0	1	1	1	0	0	1	0	0	1	55	0.20
T3	0	0	1	0	1	1	1	1	1	0	1	0	0	1	1	70	0.30
T4	0	1	0	0	1	1	0	1	0	0	0	1	0	0	0	65	0.25
T5	0	0	1	1	1	0	0	0	0	1	1	0	0	1	0	65	0.24
T6	1	0	1	1	1	1	1	1	0	1	1	0	0	1	0	60	0.29
T7	1	0	1	1	0	1	1	1	0	1	1	0	0	0	1	70	0.30
T8	0	0	1	1	1	1	0	1	0	1	1	1	0	1	0	60	0.19
T9	1	0	1	0	1	0	0	1	0	1	1	1	1	0	0	65	0.23

Now, we find the test suite by combining all the test cases generated above.

Table II: The test suite.

	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0	91	0.16

Below table 3 shows test suite that are minimized by NSGA-M:

Table III: The minimized test suite.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Fitness	Times(sec)
TS1	0	1	1	1	1	0	0	1	1	1	80	0.15
TS2	0	1	1	1	1	0	0	0	1	0	80	0.14
TS3	1	1	0	0	1	0	1	0	0	1	85	0.16
TS4	0	1	0	1	0	1	0	0	1	1	85	0.15
TS5	0	1	0	1	0	0	1	0	0	0	91	0.15

The minimized test suite found as **TS5** with size = 3 contains test cases **T2, T4, T7**

VI. Conclusion

In this thesis, we have performed an experimental comparison between two algorithms for multi-objective optimization to find out the minimize test suite. We have also explored the technique and application of evolutionary algorithm like NSGA to the automate the selection of test case for testing. Here in this thesis we mainly work on the reduction of test cases.

A lot of future scope has been observed during working in this area of study. In near future we compare this algorithm with restricted test function to get some better result. We can also improve our algorithm much more so that it gives better results in the near future.

Acknowledgement

I like to thanks all my friends and family member for their support. I also like to thank my friend Vikas Anand for helping me.

References

1. L. Luo, "Software testing techniques technology maturation and research strategy", Institute for Software Research International, Carnegie Mellon University, Pittsburgh, PA15232, USA, Tech. Rep. 17939, 2001.
2. E. F. Miller, "Introduction to software testing technology", Software Testing Validation Techniques, pp. 4 –16, 1981.
3. M. Fewster and D. Graham, "Software Test Automation Effective use of test execution tools", 2nd ed. New York: ACM Press, 1994.
4. R. V. Binder, "Testing Object-Oriented System Models, Patterns, and Tools", NY: Addison-Wesley, 1999.
5. J. B. Goodenough and S. L. Gerhart, "Toward a theory of test data selection", in Proceedings of the international conference on Reliable software, 1975, pp. 493 –510.
6. A. Andrews, R. France, S. Ghosh, and G. Craig, "Test adequacy criteria for uml design models ", Software Testing, Verification and Reliability, vol. 13, pp. 95 –127,2003.
7. Myers, Glenford , "The Art of Software Testing", Wiley. ISBN 978-0-471-46912-4,2004.
8. Savenkov, Roman , "How to Become a Software Tester", Roman Savenkov Consulting. p. 386. ISBN 978-0-615-23372-7,2008.
9. N. Srinivas and Kalyanmoy Deb., "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms", Evolutionary Computation, 2(3):221 –248, 1994.
10. R. Mall, "Fundamentals of software engineering", 2nd ed. New Delhi: Prentice- Hall of India Ltd, 2008.
11. P. Jalote, "An Integrated Approach to Software Engineering ", 3rd ed. Springer/Narosa, 2006.
12. G. Myers, "The art of software testing ", 2nd ed. Hoboken, New Jersey: JohnWiley Son, 2004.
13. R. V. Binder, "Testing Object-Oriented System Models, Patterns, and Tools ", NY: Addison-Wesley, 1999.

14. L. Owterweil, "Strategic directions in software quality ", ACM Computing Surveys (CSUR), vol. 28, no. 4, December 1996.
15. Y. Wu, M. Chen, and J. Offut, "Uml-based integration testing for component-based software ", in Proceedings of the Second International Conference on COTS-Based Software Systems. London, UK: Springer-Verlag, 2003, pp. 251 –260.
16. S. Yoo and M. Harman., "Pareto efficient multi-objective test case selection ", in Proceedings of the 2007 international symposium on Software testing and analysis, 2007, pp. 140 –150.
17. R. M. Hierons, S. Sadeghipour, and H. Singh, "Testing a system specified using statecharts and z ", Information and Software Technology, vol. 43, no. 2, pp. 137–149, 2001.
18. C. Mingsong, Q. Xiaokang, and L. Xuandong, "Automatic test case generation for uml activity diagrams ", in Proceedings of the 2006 international workshop on Automation of software test, Shanghai, China, 2006, pp. 2 –8.

AUTHOR(S) PROFILE

Gurucharan Mandi, received the B. Tech degree in Information Technology from Govt. College of Engg. And Text. Tech., Serampore and M.Tech. degree in Information Security from MNNIT Allahabad in 2011 and 2013, respectively. Now working as Porterm Lecture in FET, SSTC- SSGI, Bhilai.

Pranav Kumar, received the B. Tech degree in Information Technology from Govt. College of Engg. And Text. Tech., Serampore and M.Tech. degree (Persuing) in Computer Application from ISM Dhanbad in 2011 and 2014, respectively.