IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 5, May 2016.
www.ijiset.com
ISSN 2348 – 7968

# Software Test Case Reduction using Genetic Algorithm: A Modified Approach

**Bhawna [#1], Gaurav Kumar[*2], Pradeep Kumar Bhatia[#3]**

[#]Deptt.of CSE, G.J. University of Science & Technology, Haryana, India
[1]bhawnakataria14792@gmail.com
[3]pkbhatia.gju@gmail.com
[*]Deptt. of IT, Panipat Institute of Engineering & Technology, Haryana, India
[2]er.gkgupta@gmail.com

***Abstract* –Software test cases play an important role in Software Testing. A lot of test cases are developed while designing test cases that are of no use or redundant in nature. These types of test cases increase the effort and cost involved in testing and results in decrease of software testing efficiency. In this paper, a Genetic Algorithm based methodology has been proposed which can significantly reduce the test suite and leads to optimization of the efficiency of software testing. The testing efficiency can be increased by identifying the most critical paths. The proposed Genetic Algorithm based approach select the software path clusters that are weighted in accordance with the criticality of the path. Proposed algorithm has been evaluated by applying simple Genetic Algorithm and Fitness Scaling on a case study. GA with Fitness scaling helps in improving the efficiency of software testing by effectively reducing software test cases.**

***Index Terms*–Fitness Scaling, Genetic Algorithm, Software Testing, Test Cases**.

## I. INTRODUCTION

Software testing is the process of finding errors in the computer software. According to ANSI/IEEE 1059 standard, testing can be defined as "A process of analysing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item". The software testing is a lengthy and time consuming process which takes almost 50% of software development process time and resources. A very large number of test cases can be generated for a small piece of code. This is because the input space of the Software under Test (SUT) might be very large. The testing under all combinations of inputs and pre conditions is not feasible. So, the main goal of testing must be to reduce or minimize the number of test cases to save the time and resources. The representative subset of test cases must be chosen such that it has the highest likelihood of uncovering errors in the software [1]. Exhaustive testing is impractical due to resourcelimitations and requiresprogram execution with all possible combinations of valuesfor program variables. Redundant test cases or the test cases that are ofno use, simply increases the testing effort and henceincreases the cost [19].In this paper, we have used the application of Genetic Algorithm search along with fitness scaling to identify the most critical paths of program and hence to minimize the test cases.

Genetic Algorithms solves optimization problems for a large search space with given constraints.Genetic algorithm is a random and directed search algorithm based on the mechanics of biological evolution to search large and complex search spaces. Salvatore Mangano defined Genetic Algorithm as "Genetic Algorithms are good at taking large, potentiallyhuge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime" [6].

GA is an iterative process in which each iteration is known as 'Generation'. Each generation is responsible for selecting better and fit individuals from a large population of individuals (known as 'chromosomes') and throwing away individuals which are less fit. The selection of individuals is done by the process of mutation and crossover. On each generation, some individuals are recombined, crossing over elements of their chromosomes. Some of the combined individuals are mutated and then a selection process is used to determine the new population obtained from the offspring and the original population. Crucially, recombination and selection are guided by the fitness function; fitter chromosomes will have a greater chance to be selected and recombined. The probability for mutation must be kept lower and that of crossover must be kept higher to achieve right balance of exploration and exploitation in the search space [7]. The basic Genetic operators used are [2,4]:

**Crossover operator:**The crossover operator exchanges genes from two chromosomes (parents) and creates two new chromosomes (offspring). It happens according to a crossover probability $p_c$, which is set by the user. A random number r is generated in the range [0, 1] and crossover is applied on parents only if $p_c$>r. It actually increases the exploitation of the search space.

**Mutation operator:** Mutation alters one or more genes of a chromosome and creates one new chromosome. Mutation occurs according to a mutation probability $p_m$ which is also an adjustable parameter. A random number r is generated in the range [0, 1] for each bit within the chromosome and mutation is applied on the bit only if $p_m$>r. It increases the exploration in the search space.

ISSN 2348 – 7968

## II. PROPOSED APPROACH

To minimize the test cases using Genetic Algorithm, Weighted Control Flow Graph (CFG) has been used to make the fitness function or to calculate the fitness values in GA. CFG is a graphical representation of all the paths that might be traversed during the execution of a program. CFG is made for the program for which test cases are to be minimized.

Path testing tries to search each and every path in the program i.e. all the suitable test cases are searched that can cover maximum paths in CFG. Computational cost of covering all the test cases is very high. This is mainly because of the reasons: A program with loops contains infinite number of paths; the number of test cases can be very large. So, our approach selects a subset of paths to find the test cases[8].

Inputs for the proposed approach are Population size (initial number of test cases to be generated randomly), Number of generations, CFG of the program to be tested.

### *A. Evaluation of population*

The chromosomes will be evaluated according to the fitness function F used in GA. Our approach uses weighted CFG to make the fitness function and calculate the fitness values. The main problem in CFG is to assign weights to its edges. So, our first step is to assign weights to the edges of CFG. To minimize the test cases and to find more error prone test cases, more weight is assigned to edges that are more critical or error prone and less weight is assigned to less critical edges. The edges with looping and branching paths are considered to be more critical than the sequential path edges. So, the edges with loops and branches are given more weight.

At each node of CFG, its' in degree and out degree is to be considered. Here, in degree means the total sum of weights of incoming edges to that node and out degree means the weight which is distributed to outgoing edges by that node. In degree of a node is always equal to its out degree. The out degree is distributed among the outgoing edges using 80-20 rule i.e. 80% out degree is given to the critical edges and remaining 20% is given to the sequential path (less critical) edges. If there is only one outgoing edge from a node then it's in degree is equal to its out degree.

### *B. Finding fitness values*

Selection of individuals for reproduction will be done on the basis of probability of selection of individuals. For finding the probability, firstly fitness values of individuals are evaluated. Fitness value of each chromosome (test case) will be calculated according to the weight of edges included in the path of test case in CFG. As discussed in above section, more weight is assigned to path which is more critical i.e. the path containing more loops and branches. So, our fitness function becomes:

$$F = \sum_{i=1}^{n} w_i$$

Where $w_i$ = weight of the $i^{th}$edge in the particular path of test case,n = population size.

### *C. Fitness scaling*

The fitness values are needed to be scaled because of the following reasons [9]:

A local convergence can be caused by a high selection probability for few "super individuals" (pre mature convergence). It reduces the exploration of search space by GA.

At the end of GA run, all the individuals have almost same fitness values. So, proportionate selection is not effective.

Proposed approach uses linear scaling and the fitness values are scaled as follows:

$$F' = aF + b$$

where F' = Scaled fitness, F = Raw fitness,

anda, b = coefficients of linear scaling.

The values of a and b are calculated on the basis of whether we are at the beginning of GA run or at the ending of GA run. The condition for beginning or ending can be checked as:

$$F_{min} > \frac{C_{mul} * F_{avg} - F_{max}}{C_{mul} - 1.0}$$

Where$C_{mul}$= number of copies of super individuals to be sent to next generation. Usually, $C_{mul}$ = [1.2, 2].If the above condition holds true, then we are at the beginning of GA run otherwise at the ending.

**At the beginning of running GA:**

$$a = \frac{F_{avg}}{\delta}$$

$$b = \frac{F_{avg}}{\delta}(F_{max} - 2 * F_{avg})$$

where$\delta$ = $F_{max}$ - $F_{avg}$

**At the ending of running GA:**

$$a = \frac{F_{avg}}{\delta}$$

$$b = -F_{min} * \frac{F_{avg}}{\delta}$$

where$\delta$ = $F_{avg}$ - $F_{min}$

After the calculation of scaled fitness values for all chromosomes (test cases), probability of selection $P_k$ for each individual is calculated as follows:

$$P_k = \frac{F'_k}{\sum F'_k}$$

where k = 1 to n, and n = population size.

Above formula predicts that the individuals with high fitness values are selected for next generation and the individuals with low fitness values are dropped in this generation only. Finally, the cumulative probability $C_j$ is calculated as:

ISSN 2348 – 7968

$$C_j = \sum_{k=1}^{j} P_k$$

*D. Proposed Algorithm*

Following steps are followed to implement the proposed approach:

1. Initialize the population;
2. Evaluate the population (Using the given fitness function); /* Find value of F*/
3. ind scaled fitness values;/*$F' = aF + b$ */

   3.1. if ($F_{min}$> (($F_{avg}$*$C_{mul}$ - $F_{max}$)/($C_{mul}$-1.0))
   {
   $\delta = F_{max} - F_{avg}$;
   if($\delta$<=0.00001)
   a=1; b=0;
   else
   {
   a= (($C_{mul}$-1.0)*$F_{avg}$)/$\delta$;
   b= $F_{avg}$*($F_{max}$ - $C_{mul}$*$F_{avg}$)/$\delta$;
   }
   }

   3.2. else
   {
   $\delta = F_{avg} - F_{min}$;
   if($\delta$<=0.00001)
   a=1; b=0;
   else
   {
   a= $F_{avg}$/$\delta$;
   b= -$F_{min}$ * $F_{avg}$/$\delta$;
   }
   }

4. while Termination_Criteria_Not_Satisfied
   {
   select parents for reproduction;
   perform crossover and mutation;
   evaluate population;
   find scaled fitness values;
   }

*E. Flowchart of Proposed Algorithm*

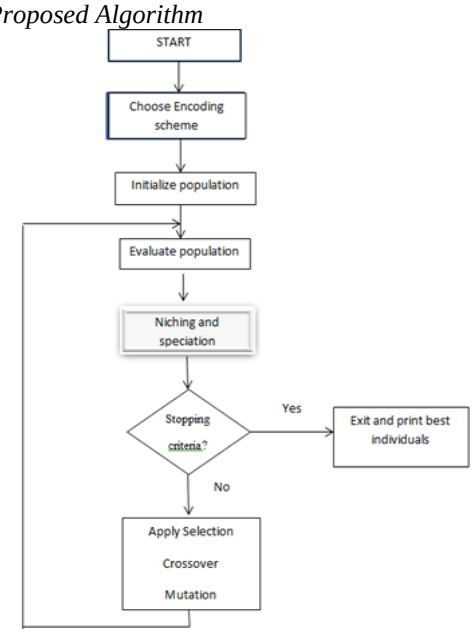


Figure 1: Flowchart of proposed algorithm

## III. Implementation and Result Analysis

Consider the problem of finding HCF (Highest Common Factor) of two numbers and results will becompared when using simple GA and GA with fitness scaling. The program and its CFG for which test cases are to be minimized are shown in figure 2.

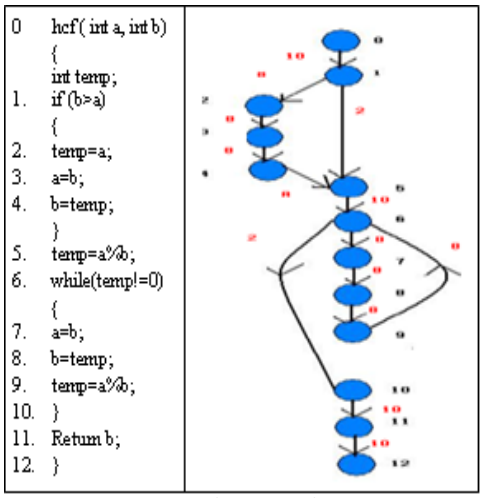


Figure2: CFG to find HCF of two numbers

*A. Solving Case study using simple GA*

Application of GA for various generations has been shown in Table I-IX

ISSN 2348 – 7968

TABLE I:FITNESS FINDING FOR GENERATION 1

| | Test cases | F | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|
| 1 | (11,12) | 108 | 0.168 | 0.169 | 0.438 | 3 | 0 |
| 2 | (6,10) | 140 | 0.220 | 0.389 | 0.381 | 2 | 2 |
| 3 | (14,3) | 172 | 0.270 | 0.660 | 0.765 | 4 | 1 |
| 4 | (3,12) | 76 | 0.119 | 0.779 | 0.795 | 5 | 1 |
| 5 | (15,12) | 140 | 0.220 | 1.000 | 0.186 | 2 | 1 |

TABLE II: CROSSOVER AND MUTATION FOR GENERATION 1

| Sn | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 3 | (14,3) | (15,2) | (14 , 3) |
| 2 | 2 | (6,10) | (10,6) | (6 , 10) |
| 3 | 4 | (3,12) | (12,3) | (3 , 12) |
| 4 | 5 | (15,12) | (15,12) | (15 , 12) |
| 5 | 2 | (6,10) | (6,10) | (6 , 10) |

TABLE III:FITNESS FINDING FOR GENERATION 2

| Sn | Test cases | F | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|
| 1 | (14,3) | 172 | 0.257 | 0.257 | 0.263 | 2 | 0 |
| 2 | (6,10) | 140 | 0.209 | 0.467 | 0.654 | 4 | 2 |
| 3 | (3,12) | 76 | 0.113 | 0.580 | 0.689 | 4 | 0 |
| 4 | (15,12) | 140 | 0.209 | 0.790 | 0.748 | 4 | 3 |
| 5 | (6,10) | 140 | 0.209 | 1.000 | 0.450 | 2 | 0 |

TABLE IV: CROSSOVER AND MUTATION FOR GENERATION 2

| Sn | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 2 | (6,10) | (14,2) | (10,6) |
| 2 | 4 | (15,12) | (12,15) | (15,12) |
| 3 | 4 | (15,12) | (15,12) | (15,12) |
| 4 | 4 | (15,12) | (12,15) | (15,12) |
| 5 | 2 | (6,10) | (10,6) | (10,6) |

TABLE V:FITNESS FINDING FOR GENERATION 3

| Sn | Test cases | F | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|
| 1 | (10,6) | 172 | 0.225 | 0.225 | 0.821 | 5 | 3 |
| 2 | (15,12) | 140 | 0.183 | 0.408 | 0.015 | 1 | 0 |
| 3 | (15,12) | 140 | 0.183 | 0.591 | 0.043 | 1 | 0 |
| 4 | (15,12) | 140 | 0.183 | 0.774 | 0.169 | 1 | 1 |
| 5 | (10,6) | 172 | 0.225 | 1.000 | 0.649 | 4 | 1 |

TABLE VI: CROSSOVER AND MUTATION FOR GENERATION 3

| Sn | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 5 | (10,6) | (10,6) | (10,6) |
| 2 | 1 | (10,6) | (14,2) | (14,2) |
| 3 | 1 | (10,6) | (14,2) | (14,2) |
| 4 | 1 | (10,6) | (14,2) | (14,2) |
| 5 | 4 | (15,12) | (15,12) | (12,15) |

TABLE VII:FITNESS FINDING FOR GENERATION 4

| Sn | Test cases | F | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|
| 1 | (10,6) | 172 | 0.284 | 0.284 | 0.594 | 3 | 2 |
| 2 | (14,2) | 108 | 0.178 | 0.463 | 0.262 | 1 | 0 |
| 3 | (14,2) | 108 | 0.178 | 0.642 | 0.602 | 3 | 2 |
| 4 | (14,2) | 108 | 0.178 | 0.821 | 0.711 | 4 | 1 |
| 5 | (12,15) | 108 | 0.178 | 1.000 | 0.221 | 1 | 0 |

TABLE VIII: CROSSOVER AND MUTATION FOR GENERATION 4

| S n | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 3 | (14,2) | (14,2) | (14,2) |
| 2 | 1 | (10,6) | (10,6) | (10,6) |
| 3 | 3 | (14,2) | (14,2) | (14,2) |
| 4 | 4 | (14,2) | (14,2) | (14,2) |
| 5 | 1 | (10,6) | (10,6) | (10,6) |

TABLE IX:FITNESS FINDING FOR GENERATION 5

| Sn | Test cases | F | $P_i$ | $C_i$ | Random no. | N |
|---|---|---|---|---|---|---|
| 1 | **(14,2)** | **108** | 0.161 | 0.161 | 0.978 | 5 |
| 2 | (10,6) | 172 | 0.257 | 0.419 | 0.712 | 4 |
| 3 | (14,2) | 108 | 0.161 | 0.580 | 0.500 | 3 |
| 4 | (14,2) | 108 | 0.161 | 0.742 | 0.471 | 3 |
| 5 | (10,6) | 172 | 0.257 | 1.000 | 0.059 | 1 |

Finally, the best test case out of five randomly generated test cases is (14, 2) with fitness value = 108 but it can easily be seen that this is not the best test case. So, it is clear that simple GA sometimes leads to pre mature convergence of individuals which is the main drawback of simple GA.

*B. Solving Case study using GA with fitness scaling*

Application of GA with fitness scaling is shown in table x-xix for various generations.

TABLE X:FITNESS FINDING FOR GENERATION 1

| S. No | Test cases | F | F' | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|---|
| 1 | (9,0) | 22 | 76 | 0.200 | 0.200 | 0.987 | 5 | 2 |
| 2 | (0,12) | 72 | 0 | 0 | 0.200 | 0.170 | 1 | 0 |
| 3 | (5,5) | 44 | 0 | 0 | 0.200 | 0.257 | 4 | 0 |
| 4 | (9,12) | 108 | 152 | 0.400 | 0.600 | 0.396 | 4 | 2 |
| 5 | (2,3) | 108 | 152 | 0.400 | 1.000 | 0.074 | 1 | 1 |

TABLE XI: CROSSOVER AND MUTATION FOR GENERATION 1

| Sn | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 5 | (2,3) | (2,3) | (2,3) |
| 2 | 1 | (9,0) | (0,9) | (9,0) |
| 3 | 4 | (9,12) | (8,13) | (9,12) |
| 4 | 4 | (9,12) | (9,12) | (9,12) |
| 5 | 1 | (9,0) | (9,1) | (9,0) |

ISSN 2348 – 7968

TABLE XII:FITNESS FINDING FOR GENERATION 2

| Sn | Test cases | F | F' | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|---|
| 1 | (2,3) | 108 | 158.6 | 0.333 | 0.333 | 0.120 | 1 | 2 |
| 2 | (9,0) | 22 | 0 | 0 | 0.333 | 0.525 | 3 | 0 |
| 3 | (9,12) | 108 | 158.6 | 0.333 | 0.666 | 0.325 | 1 | 3 |
| 4 | (9,12) | 108 | 158.6 | 0.333 | 1.000 | 0.546 | 3 | 0 |
| 5 | (9,0) | 22 | 0 | 0 | 1.000 | 0.398 | 3 | 0 |

TABLE XIII: CROSSOVER AND MUTATION FOR GENERATION 2

| Sn | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 1 | (2,3) | (3,2) | (2,3) |
| 2 | 3 | (9,12) | (8,13) | (8,13) |
| 3 | 1 | (2,3) | (2,3) | (2,3) |
| 4 | 3 | (9,12) | (13,8) | (8,13) |
| 5 | 3 | (9,12) | (9,12) | (8,13) |

TABLE XIV:FITNESS FINDING FOR GENERATION 3

| Sn | Test cases | F | F' | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|---|
| 1 | (2,3) | 108 | 0 | 0 | 0 | 0.180 | 2 | 0 |
| 2 | (8,13) | 204 | 276.0 | 0.333 | 0.333 | 0.045 | 2 | 2 |
| 3 | (2,3) | 108 | 0 | 0 | 0.333 | 0.723 | 5 | 0 |
| 4 | (8,13) | 204 | 276.0 | 0.333 | 0.666 | 0.347 | 4 | 2 |
| 5 | (8,12) | 204 | 276.0 | 0.333 | 1.000 | 0.660 | 4 | 1 |

TABLE XV: CROSSOVER AND MUTATION FOR GENERATION 3

| Sn | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 2 | (8,13) | (9,12) | (8,13) |
| 2 | 2 | (8,13) | (8,13) | (8,13) |
| 3 | 5 | (8,12) | (9,12) | (9,12) |
| 4 | 4 | (8,13) | (8,13) | (13,8) |
| 5 | 4 | (8,13) | (12,9) | (13,8) |

TABLE XVI:FITNESS FINDING FOR GENERATION 4

| Sn | Test cases | F | F' | $P_i$ | $C_i$ | Random no. | N | M |
|---|---|---|---|---|---|---|---|---|
| 1 | (8,13) | 204 | 211.7 | 0.214 | 0.214 | 0.511 | 4 | 1 |
| 2 | (8,13) | 204 | 211.7 | 0.214 | 0.428 | 0.082 | 1 | 1 |
| 3 | (9,12) | 108 | 0 | 0 | 0.428 | 0.719 | 5 | 0 |
| 4 | (13,8) | 172 | 282.2 | 0285 | 0.714 | 0.996 | 5 | 1 |
| 5 | (13,8) | 172 | 282.2 | 0.285 | 1.000 | 0.354 | 2 | 2 |

TABLE XVII: CROSSOVER AND MUTATION FOR GENERATION 4

| Sn | N | M | Crossover | Mutation |
|---|---|---|---|---|
| 1 | 4 | (13,8) | (8,13) | (8,13) |
| 2 | 1 | (8,13) | (12,9) | (9,12) |
| 3 | 5 | (13,8) | (13,8) | (12,9) |
| 4 | 5 | (13,8) | (9,12) | (12,9) |
| 5 | 2 | (8,13) | (13,8) | (8,13) |

TABLE XVIII:FITNESS FINDING FOR GENERATION 5

| Sn | Test cases | F | F' | $P_i$ | $C_i$ | Random no. | N |
|---|---|---|---|---|---|---|---|
| 1 | **(8,13)** | 204 | **298.5** | 0.375 | 0.375 | 0.343 | 1 |
| 2 | (9,12) | 108 | 0 | 0 | 0.375 | 0.936 | 5 |
| 3 | (12,9) | 76 | 99.5 | 0.125 | 0.500 | 0.124 | 1 |
| 4 | (12,9) | 76 | 99.5 | 0.125 | 0.625 | 0.730 | 5 |
| 5 | (8,13) | 204 | 298.5 | 0.375 | 1.000 | 0.646 | 5 |

***F*** in the above tables represent raw fitness values and ***F'*** represents scaled fitness values. The best test case comes out to be (8, 13) with fitness value = 204 which is in actual the best test case out of five randomly generated test cases. So, fitness scaling improves the efficiency of simple GA and gives the optimized results.

## IV. CONCLUSION

A more effective way to increase efficiency of testing has been introduced which saves a lot of time and resources. GA can be used for any optimization problem independently and hence this ever emerging technique is of great importance for researchers in every field.In this paper, GA has been used to minimize the test cases. It is shown by using an example that Genetic search can be applied in Software Testing field to minimize the number of test cases by finding most error prone test cases based on the criticality of path. Fitness scaling has been used to overcome the limitations of simple GA which was pre-mature convergence of individuals that leads to local optimal value rather than global optima. Some advanced techniques niching and speciation can be tried out which may give results in only single run of GA.

## REFERENCES

[1] N. K. Gupta, M. K. Rohil, "Using Genetic Algorithm for Unit Testing of Object Oriented Software", International Journal of Simulation: Systems, Science and Technology, Vol.10, No.3, pp.308-313, July 2008.

[2] M. Alzabidi, A. Kumar, A. D. Shaligram, **"**Automatic Software Structural Testing by Using Evolutionary Algorithms for Test Data Generations", International Journal of Computer Science and Network Security, Vol.9, No.4, pp.390-395, April 2009.

[3] T. Dwivedi, J. Srivastava, "Software Testing StrategyApproach on Source Code ApplyingConditional Coverage Method", International Journal of Software Engineering and its Applications, Vol.6, No.3, pp.25-31, May 2015.

[4] S. Eyal, A. Kandel, M. Last, "Effective Black-Box Testing with Genetic Algorithms, Springer-VerlagBerlinHeidelberg, pp. 134-148, Nov. 2005.

[5] D. J. Mala, V. Mohan, E. Ruby, "A Hybrid Test Optimization Framework –Coupling Genetic Algorithm with Local Search Technique", Computing and Informatics, Vol.29, No.1, pp.133-164, 2010.

ISSN 2348 – 7968

[6] Z. Michaelewicz, "Genetic Algorithms + Data Structures = Evolution Programs", Springer, 3rd revised and extended edition, Oct. 1995.

[7] S. Mittal, O. P.Sangwan, "Metaheuristic Based Approach to Regression Testing", International Journal of Computer Science and Information Technologies, Vol.6, No.3, pp.2597-2605, 2015.

[8] T. H. Kim, P. R. Srivastava, "Application of Genetic Algorithm in Software Testing", International Journal of Software Engineering and Its Applications, Vol.3, No.4, pp.87-96, Oct. 2009.

[9] D. E. Goldberg, "Genetic Algorithms in search, optimization and machine learning", Pearson Education, 2009.

[10] F.A. Sadjadi, "Comparison of fitness scaling functions in genetic algorithms with applications to optical processing", Optical Information Systems 2, Vol.5557, pp.356-364, 2004.

[11] Y.Y. Chen, K.X. Wei, G. Gong, X.T. Hu, "Parameters selection of fitness scaling in genetic algorithm and its applications", IEEE, pp.2475-2480, May 2010.

[12] K.K. Aggarwal, Y. Singh, "Software Engineering Programs Documentation, Operating Procedures", New Age InternationalPublishers, Revised 2ndEdition, 2005.

[13] R. Jain, B. Pandey, "Soft Computing based Approaches for Software Testing", International Journal of Soft Computing and Engineering, Vol.4, Issue 2, pp.4-8, May 2014.

[14] A.P. Mathur, "Foundation of Software Testing", Pearson Education, 1stedition, 2008.

[15] G.S.V.P. Raju, V. Sumanlatha, "Object Oriented Test Case Generation Technique using Genetic Algorithms", International Journal of Computer Applications, Vol.61, No.20, pp.20-26, Jan. 2013.

[16] I. Somerville, "Software Engineering", Addison-Wesley, 7thEdition, May 2004.

[17] "Basics of Genetic Algorithm" retrieved from http://www.obitko.com/tutorials/genetic-algorithms/

[18] "Testing techniques" retrieved from http://www.tutorialspoint.com/software_testing/

[19] Gaurav Kumar, Pradeep Kumar Bhatia, "*Software Testing Optimization through Test Suite Reduction using Fuzzy Clustering*", CSI Transactions on ICT, Vol. 1, Issue 3, pp. 253-260, Sep. 2013.