SOFTWARE TESTING USING GENETIC ALGORITHMS

Akshat Sharma[1], Rishon Patani[1] and Ashish Aggarwal[1]

[1]School of Computer Science and Engineering, VIT University, Vellore, Tamil Nadu, India

ABSTRACT

This paper presents a set of methods that uses a genetic algorithm for automatic test-data generation in software testing. For several years researchers have proposed several methods for generating test data which had different drawbacks. In this paper, we have presented various Genetic Algorithm (GA) based test methods which will be having different parameters to automate the structural-oriented test data generation on the basis of internal program structure. The factors discovered are used in evaluating the fitness function of Genetic algorithm for selecting the best possible Test method. These methods take the test populations as an input and then evaluate the test cases for that program. This integration will help in improving the overall performance of genetic algorithm in search space exploration and exploitation fields with better convergence rate.

KEYWORDS

Genetic algorithm, Fitness function, Test data.

1. INTRODUCTION

Software testing is a process in which the runtime quality and quantity of a software is tested to maximum limits. The basic test of software is done in the environment for which it is has been designed. It's run through is checked for correct and efficient outputs. Also software testing is done in foreign environments also so as to explore about the possibilities of scalability [8, 12]. Each software is tested under various strategized environments.

The testing done is expected to produce the correct results under the assumptions of specific functions, but at no time all the defects can be identified. Instead, it furnishes a comparison that compares the state and behavior of the product —principles or mechanisms by which users might be able to recognize the problem.

A test case is generally the data which acts as input for the software testing to done. Its consists of unique identifier, requirement references from a software specification, a series of steps to follow, events, preconditions, input, output, expected result, and actual result.[17] It is also coined as an expected output to the tested environment. This can be as pragmatic as "for condition your derived result is b", whereas other test cases described the input scenario in detailed analysis and showed results that were expected [2, 9, 16].

Evolutionary Testing uses a kind of meta-heuristic search technique, the Genetic Algorithm (GA), to convert the task of test case generation into an optimal problem [4, 13, 27]. Evolutionary testing is used to search for optimal test parameter combinations that satisfy a predefined test criterion. This test criterion is represented by using a "cost function" that measures how well each of the automatically generated optimization parameters are satisfying the given test criterion [7, 22].

In our paper a study of different types of genetic algorithms is done. Different algorithms have been run on different tools and analyzed for their performance. All these algorithms follow the same basis of evolutionary testing but have different cost functions. On running these cost functions on different tools, observations on how these functions respond are made [1, 3, 5, 11].

2. INTRODUCTION TO GENETIC ALGORITHM

Genetic algorithms are one of the best ways to solve a set of problems for which little information is given. Genetic algorithms are a very general algorithm and so they will work well in any search space [1, 25, 30, 33]. All you need to know is what you need the solution to be able to do well, and a genetic algorithm will be able to create a high quality solution. Genetic algorithms use the principles of selection and evolution to produce solution for various complex problems.

Genetic algorithms tends to thrive in an environment in which there is a very large set of candidate solutions and in which the search space is not favorable and has many hills and valleys[12,15,16]. True, genetic algorithms can do well in any environment, but they might be greatly outclassed by more situation specific algorithms in the simpler search spaces. Therefore you must keep in mind that genetic algorithms are not always the best choice in random scenarios. Sometimes they might take quite a while to run and are therefore not always feasible for real time use. They are, however, one of the most powerful methods with which high quality solutions are created quickly to a problem [4,8,21]. There are few basic methodology/Terminology that will be used while implementing genetic algorithm like:

Individual – Possible solutions
Population - Set of all *individuals*
Search Space - All possible solutions to the specified problem
Chromosome – Blueprint for an *individual*
Trait - Possible aspect of an *individual entity.*
Allele - Possible settings for a *trait*
Locus - The position of a *gene* on the *chromosome*
Genome - Collection of all *chromosomes* for an *individual entity.*

3. BACKGROUND

Genetic algorithms use the following three operations on its population.

3.1 Selection:

A selection process is applied to determine a way in which individuals are chosen for mating from a population based on their fitness. Fitness is defined as a characteristic and capability of an individual to survive and reproduce in an environment. Selection generates the new population from the old one, thus starting a new generation. The fitness value of each chromosome in present generation is determined by an appropriate evaluation. Thus, the fitness value is used to select a set of better chromosomes from the population for the next generation.[5,6].

3.2 Crossover:

After the selection process, the crossover operation is applied to the chromosomes selected from the population. Crossover involves swapping of sequence of bits or genes in the string between two individuals [8, 10]. This process of swapping is carried out and repeated each time with different parent individuals until the next generation has optimum individuals.

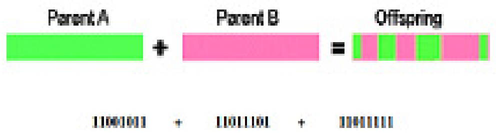


Figure 3.2: Uniform Crossover

3.3 Mutation:

After the crossover process, the mutate operation is applied to a randomly select subset of the population. Mutation leads to an alteration of chromosomes in small new ways to introduce good traits. The main aim of mutation is to bring diversity in population.[1,5]

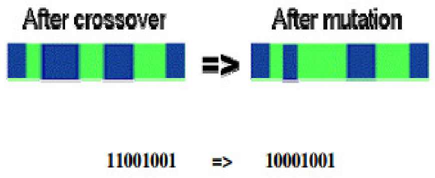


Figure 3.3: Mutation (Bit Inversion)

Factors essential in a fitness function are:

- Likelihood.
- Close to Boundary Value.
- Branch Coverage.

It has been proven that GAs required less CPU time in reaching a global solution in software testing **[**13**].**

3.4 Need for Genetic Algorithms in Software Testing:

Drawbacks of manual testing: [7, 12]
Speed of operation is limited as it is carried out by humans.
High investment in terms of cost, time.
Limited availability of resources.

Redundancy in test cases.
Inefficient and inaccurate test checking.
Pros of using genetic algorithms in software testing:
Parallelism is a important characteristic of genetic testing [11,19].
Less likely to get stuck in extreme ends of a code during testing since it operates in a search space.
With the same encoding, only fitness function needs to be changed according to the problem.

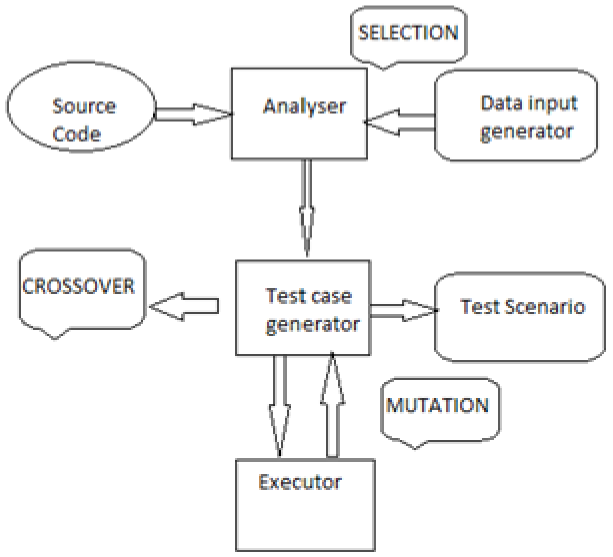


Figure 4.1: Test Case Generation in Software Testing Using GA

4. GENETIC ALGORITHM WORKING

The genetic algorithm is an evolutionary approach to computing, which has the ability to determine appropriate approx. solutions to optimization problems. The basic process adopted by Genetic algorithms typically involves creating an initial set of random solutions (population) and evaluating them [2, 5, 9, 12]. Followed by a process of selection, the better solutions are identified (parents) and are then used to generate new solutions (children). These values can be used to replace other lesser members of the population. This new population (generation), is then reevaluated and the process for generating new values continues, reproducing new generations until either a final solution is determined or some other criterion for determination of result is reached[18,22,32].

Genetic Algorithm borrows its terms from the biological world. For example, Genetic algorithm uses different representations for potential solutions which are referred to as a chromosome and the operators that are used to generate new child solutions are such as crossover and mutation are derived from nature.

In their generic and most basic form, Genetic Algorithms were used mainly for single objective search and optimization algorithms. Common to most Genetic algorithms is the use of a chromosome, genetic operators, a selection mechanism and an evaluation mechanism [23, 27].

In our case, all we have to do is determine the parameters we need in order to build an efficient function and bundle them into a binary string, this will be the definition of our chromosome. Therefore, each chromosome will fully describe a Function.

A common approach when working with Genetic Algorithm is to start by making a 'population' of random chromosomes (Test Variables) perhaps a 100. You may remember earlier that we said we can test each one individuals and score it, or to use the correct terminology, 'evaluate its fitness' (function).

This calculated Fitness function will help us to evaluate the efficiency of the method used [11, 27, 29].And further to increase the efficiency of the test result we can change the input parameters and obtain values for different number of population.

5. Applications of Genetic Algorithm

Genetic Algorithm have been used for solving complex problems (such as NPC and NP-hard), for machine learning and is also used for evolving simple test programs. They are a very effective way of quickly finding a reasonable solution to a complex problem.

Genetic algorithms are most efficient and effective in a search space for which little is known. Then again, genetic algorithms can be used to produce solutions to problems working only in the test environment and deviates once you try to use them in the real world [17, 24].

So when put simply, genetic algorithm can be used to create solutions for problems that are not very easy to calculate and analyze.

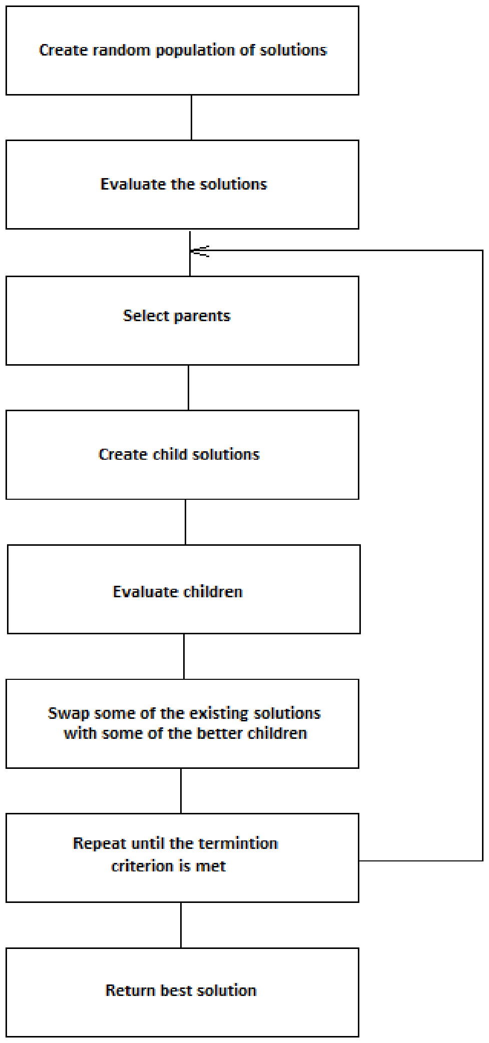

Create random population of solutions
Evaluate the solutions
Select parents
Create child solutions
Evaluate children
Swap some of the existing solutions with some of the better children
Repeat until the termintion criterion is met
Return best solution

Figure 5.0: Flow chart of the workflow of Genetic Algorithm used for Test Case Generation n Software Testing.

6. IMPLEMENTATION OF GA IN SOFTWARE TESTING

6.1 Test case generation using GA in Ruby

Algorithm:

Start with randomly generated test cases from the population.
Calculate the fitness f(x) of each pair of test cases (chromosome x) in the population.
Repeat the following steps until a n child test cases have been generated.

a. Select a pair of parent test cases from the current population where the probability of selection is an increasing function of fitness. Selection is done "with replacement," meaning that the same pair of test case can be selected more than once to become a parent. i.e. (Selection process is carried out)
b. With the crossover probability Pc, cross over the pair at a randomly chosen point to form two child cases or off springs. If no crossover takes place, form two test cases that are exact copies of their respective parent cases.
c. Mutate the two child cases with mutation probability Pm, and place the resulting pair of test cases in the new population. If n is odd, one new population member can be discarded at random.
Replace the current test cases with the new test cases.

6.1.1

Population size = 50
Number of generations = 500
Crossover rate=0.7
Mutation rate = 0.001

Ruby Implementation with First Case:

Generation	Average fitness	Max fitness
1	31.88	33
2	32.88	34
3	32.67	34
4	32.75	35
5	32.96	34
10	34.17	38
25	37.29	42
50	41.21	44
100	39.67	47
150	44.33	49
200	46.33	47
250	47.63	49
500	50.23	50
750	51.79	51
1000	52	51

Table 1: Average fitness value with mutation rate = 0.001

6.1.2

Population size = 50
Number of generations = 500
Crossover rate=0.7
Mutation rate = 0.01

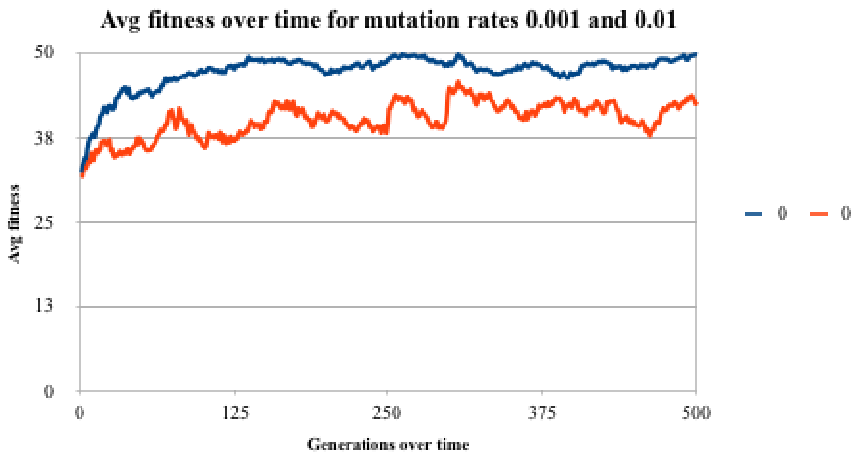
Ruby Implementation with Second Case:

Generation	Average	Max
1	32.25	32
2	32.13	32
3	33.92	33
4	32.42	33
5	33.79	33
10	34.71	34
25	38.42	36
50	39.08	37
100	37.42	36
150	35.54	40
200	38.79	40
250	41.83	39
500	42.18	43

Table 2: Average fitness value with mutation rate = 0.01

Mutation rate has a great impact on the average on the average fitness of genetic algorithms during testing. Smaller the rate, better the fitness function value will be.

Below is a graph which represents the average fitness overtime for different mutation rates.



6.2 Genetic Algorithm Implementation in C++

Pseudo-code for genetic algorithm:

```
choose initial_population:
evaluate individual_fitness function
determine population's_average
            fitness_function
  Repeat
            select best_case individuals to
            reproduce;
            mate_pairs at random;
            apply crossover_operator;
            apply mutation_operator;
            evaluate Individual fitness;
            determine population's average
            fitness;
```

The second step consists for generating data consists of the outer loop, which will generate the possible test cases remaining. To account for the possibility of unfeasible test requirements, which includes branches and statement values, the loop will produces iterations until it satisfies the test results for the given population values. The algorithm will produce values which will be applied for the crossover and mutation operator. Then the fitness function for individual values are generated and the population's individual average fitness functions in calculated.

In the final step, the algorithm will assign the combined values of the test cases and find at least one individual desired fitness function values until enough test generations have been passed.

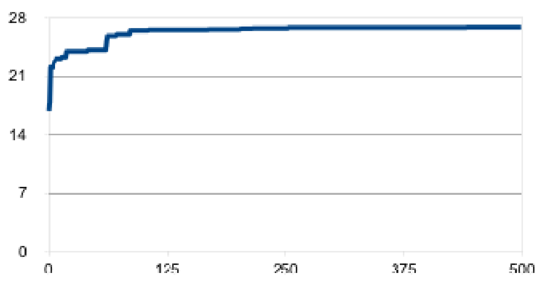
6.2.1

Population size = 50
Number of generations = 250
Crossover rate=0.7
Mutation rate = 0.001

Table 3: Best fitness value generation

Iteration	Best fitness	Average fitness	Standard Deviation
1	16.77	11.59	5.54
2	20.39	15.09	3.26
3	20.39	15.71	3.27
4	22.99	15.84	3.79
5	22.99	16.03	3.89
10	23.24	17.65	3.46
25	25.12	20.87	3.61

50	25.89	21.42	4.32
100	26.27	20.88	5.33
150	26.77	23.28	3.77
200	26.77	20.38	6.91
250	26.78	22.41	4.42
500	26.98	22.68	5.54



Graph: Number of generations *(x-axis)* vs. average fitness *(y-axis)*

6.3 Genetic Algorithm Implementation using Matlab

In this work both the genetic algorithm and the random testing method were compared and detailed analysis of the best fitness has been evaluated. In order to compare Genetic algorithm and a pure random method, 150 test cases were generated and tested by both methods [22]. From this we can see that the average response time of test cases created by genetic algorithm is much efficient than that of the random method.

Case 1 shows the plot of the best and mean score of the population at every generation. The second plot function is stopping criteria, which plots the percentage of stopping criteria satisfied.

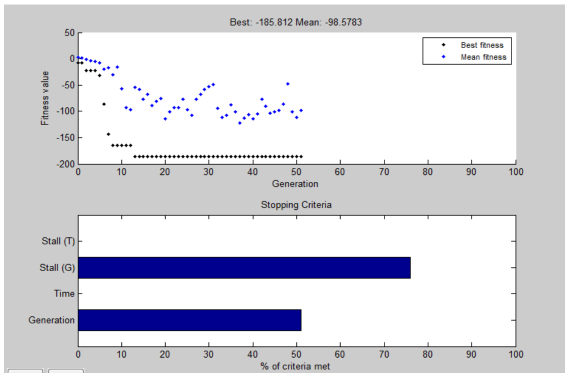


Figure 1

Results generated for Case 1:

The number of generations: 124
The number of function evaluations: 6250
The best function value found: -186.

Case 2 shows a better detailed analysis of the best and the mean fitness function by changing the test cases to 20 instead of 10. This iteration generates better results than the previous iteration and by further iterating the test cases the result is obtained.

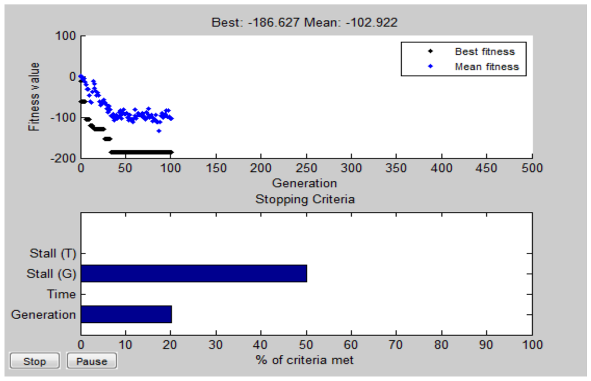


Figure 2

Results generated for Case 2:

The number of generations: 68
The number of function evaluations: 650
The best function value found: -176.997

Case 3

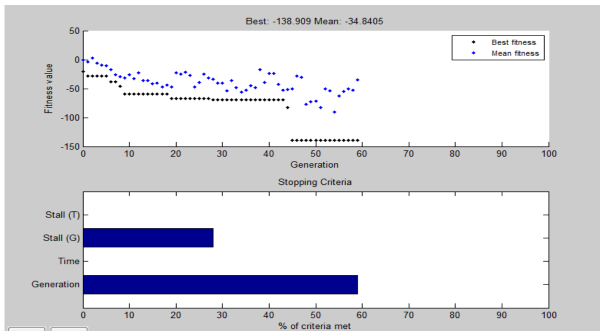


Figure 3

Results generated for final Case 3:

The number of generations was: 101
The number of function evaluations was: 5100
The best function value found was: -186.627

Genetic algorithm for test case generation using Mat lab was implemented. It can be found out that for a population of 50 and maximum generations 500, the best fitness function is carried out until maximum generation is reached or until a certain value after which no significant change occurs. The iterative generation is stopped itself after such a condition hence providing us with a further optimization and avoid redundant solutions.

7. Conclusions

In this paper, we analyzed how evolutionary techniques such as GAs helped in software testing. The results show how software testing using Genetic Algorithms becomes efficient even with increasing number of test cases. In Random Testing Methods, since data points do not have a dependence with time, it becomes inefficient as code becomes complex. Thus, to increase the efficiency and process time of Software testing, Genetic Algorithms are being used and provide us a means of an automatic test case generator. The evolutionary generation of test cases can be applied and proves to be efficient and cost effective than Random Testing.

Acknowledgement

We would like to acknowledge the immense support of our Professor Dr Manjula R without whom it was not possible to write the paper. Her guidance formed the base of the paper and her expertise comments greatly enhanced the manuscript. We would also like thank our university (VIT University, Vellore, Tamil Nadu, India) which has always encouraged the students to go in the field of research and develop an interest towards exploration and innovation.

References

[1] Goldberg, D.E, “Genetic Algorithms: in Search, Optimization & Machine Learning,” Addison Wesley, MA. 1989.

[2] Horgan, J., London, S., and Lyu, M., “Achieving Software Quality with Testing Coverage Measures”, IEEE Computer, Vol. 27 No.9 pp. 60-69, 1994.

[3] Berndt, D.J., Fisher, J., Johnson, L., Pinglikar, J., and Watkins, A., “Breeding Software Test Cases with Genetic Algorithms,” In Proceedings of the Thirty-Sixth Hawaii International Conference on System Sciences (HICSS-36), Hawaii, January 2003.

[4] Mark Last, Shay Eyal1, and Abraham Kandel, “Effective Black-Box Testing with Genetic Algorithms,” IBM conference.

[5] Lin, J.C. and Yeh, P.L, “Using Genetic Algorithms for Test Case Generation in Path Testing,” In Proceedings of the 9th Asian Test Symposium (ATS’00). Taipei, Taiwan, December 4-6, 2000.

[6] André Baresel, Harmen Sthamer and Michael Schmidt, “fitness function design to improve evolutionary structural testing,” proceedings of the genetic and evolutionary computation conference, 2002.

[7] Christoph C. Michael, Gary E. McGraw, Michael A. Schatz, and Curtis C. Walton, “Genetic Algorithms for Dynamic Test Data Generation,” Proceedings of the 1997 International Conference on Automated Software Engineering (ASE'97) (formerly: KBSE) 0-8186-7961-1/97 © 1997 IEEE.

[8] Somerville, I., “Soft ware engineering,” 7th Ed. Addison-Wesley,

[9] Aditya P mathur,”Foundation of Software Testing”, 1st edition Pearson Education 2008.

[10] Alander, J.T., Mantere, T., and Turunen, P, “Genetic Algorithm Based Software Testing,” http://citeseer.ist.psu.edu/40769.html, 1997.

[11] Nashat Mansour, Miran Salame,” Data Generation for Path Testing”, Software Quality Journal, 12, 121–136, 2004,Kluwer Academic Publishers.

[12] Praveen Ranjan Srivastava et al, "Generation of test data using Meta heuristic approach" IEEE TENCON (19-21 NOV 2008), India available in IEEEXPLORE.

[13] Wegener, J., Baresel, A., and Sthamer, H, "Suitability of Evolutionary Algorithms for Evolutionary Testing," In Proceedings of the 26th Annual International Computer Software and Applications Conference, Oxford, England, August 26-29, 2002.

[14] Berndt, D.J. and Watkins A, "Investigating the Performance of Genetic Algorithm-Based. Software Test Case Generation," In Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04), pp. 261-262, University of South Florida, March 25-26, 2004.

[15] B. Korel. Automated software test data generation. IEEE Transactions on Software Engineering, 16(8), August 1990.

[16] Bo Zhang, Chen Wang, "Automatic generation of test data for path testing by adaptive genetic simulated annealing algorithm", IEEE, 2011, pp. 38 – 42.

[17] Chartchai Doungsa et. al., "An automatic test data generation from UML state diagram using genetic algorithm",http://eastwest.inf.brad.ac.uk/document/publication/DoungsaardSKIMA.pdf.

[18] D.J Berndt, A. Watkins, "High volume software testing using genetic algorithms", Proceedings of the 38t h International Conference on system sciences (9), IEEE, 2005, pp. 1- 9.

[19] Francisca Emanuelle et. al., "Using Genetic algorithms for test plans for functional testing", 44th ACM SE proceeding, 2006, pp. 140 - 145.

[20] Goldberg, D.E, Genetic Algorithms: in search, optimization and machine learning, Addison Wesley, M.A, 1989.

[21] Girgis, "Automatic test generation for data flow testing using a genetic algorithm", Journal of computer science, 11 (6), 2005, pp. 898 – 915.

[22] Giuseppe A. et. al., "Testing Web –applications: The State of Art and Future Trends".Information and Software Technology. Elsevier, 2006, pp. 1172-1186.

[23] Jin- Cherng Lin, Pu- Lin Yeh, "Automatic test data generation for path testing using Gas", International journal of information sciences. Elsevier, 2000, pp. 47- 64.

[24] Jose Carlos et. al., "A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object- oriented software", AST' 08. ACM, 2008, http://www.cs.bham.ac.uk/~wbl/biblio/cache/http___jcbri beiro.googlepages.com_ast12-ribeiro.pdf, Accessed on 6.11.2012.

[25] Liang You, YanSheng Lu, "A genetic algorithm for the time – aware regression testing reduction problem", International conference on natural computation, IEEE, 2012, pp. 596 – 599.

[26] McMinn, "Search based software test generation: A survey", Software testing, Verification and reliability 14 (2), 2004, pp. 105-156.

[27] Mark Last et. al., "Effective black-box testing with genetic algorithms", Lecture notes in computer science, Springer, 2006, pp. 134 -148.

[28] Maha alzabidi et. al., "Automatic software structural testing by using evolutionary algorithms for test data generations", International Journal of Computer science and Network Security 9 (4), 2009, pp. 390 – 395.

[29] Velur Rajappa et. al., "Efficient software test case generation Using genetic algorithm based graph theory" International conference on emerging trends in Engineering and Technology, IEEE, 2008, pp. 298 - 303.

[30] Xuan Peng, Lu Lu, "A new approach for session - based test case generation by GA". IEEE, 2011, pp. 91- 96.

[31] Peter M. Kruse et. al., "A Highly Configurable test systems for evolutionary black box testing of embedded systems" GECCO. ACM, 2009, pp.1545 – 1551.

[32] Ruilian zhao, shanshan lv, "Neural network based test cases generation using genetic algorithm" 13th IEEE international symposium on Pacific Rim dependable computing. IEEE, 2007, pp.97 - 100.

[33] Robert M .Patton et. al. "A genetic algorithm approach to focused software usage testing" Annals of software engineering,http://www.cs.ucf.edu/~ecl/papers/03.rmpatto n.pdf.