ISSN 22773061

Test Suite Minimization using Hybrid Algorithm for GA generated Test Cases

P. Maragathavalli [1] and S. Kanmani [2]
[1]Assistant Professor, Department of Information Technology, Pondicherry Engineering College, Puducherry-14
marapriya@pec.edu
[2]Professor, Department of Information Technology, Pondicherry Engineering College, Puducherry-14
kanmani@pec.edu

Abstract: Software testing and retesting occurs continuously during the software development lifecycle to detect errors as early as possible. As the software evolves the size of test suites also grows. When the no of test cases generated are more, obviously size of the test suite will also be more. So the testing time is to be minimized by reducing the execution time of the algorithm used for test data generation and also by introducing minimization procedure for test suite reduction. Due to limited resources and timing constraints for testing, test suite minimization techniques are needed to eliminate redundant test cases as possible. By considering multiple objectives rather than the coverage alone, the test cases are being generated which satisfies the testing requirements. Most of the existing techniques are code-based. In this article we present an approach by modifying an existing heuristic for test suite minimization. Genetic algorithm has been used for random test data generation and the output of GA is given to the minimization procedure for reducing the total no of generated test cases, collectively named as Hybrid Algorithm (HA). The results are satisfactory and show significant improvements in reducing test suite size with minimum execution time. Experiments have been done for simple to medium complexity java programs taken from SIR and execution time is reduced to 5,685ms for a test set. The results are compared with existing method *Mutant Gene Algorithm* and size of test suite is minimized upto 13.6% using *Hybrid Algorithm*.

Keywords: multiple objectives, test suite minimization, genetic algorithm, redundant test cases, hybrid algorithm.

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol 6, No 1
editor@cirworld.com
www.cirworld.com, member.cirworld.com

ISSN 22773061

1. Introduction

Software testing involves identifying the test cases which detect errors in the program. As software grows and evolves, new test cases are generated and added to a test suite to exercise the latest modifications to the software. However, exhaustive testing of software is very time consuming. Genetic algorithms solve many search and optimization problems, effectively. In genetic algorithm, the optimized test vector is generated, which enhances the fault coverage and improve the global search. So, in order to generate optimized set of test cases, genetic algorithm is used so that more number of faults can be detected with the minimum number of test cases. Over several versions of software development, some test cases in the test suite may become redundant with respect to the testing requirements; these are also satisfied by other test cases in the suite that were added to cover modifications in the later versions of software.

Multi-objective formulations are realistic models for many complex engineering optimization problems. In many real-life problems, objectives under consideration conflict with each other, and optimizing a particular solution with respect to a single objective can result in unacceptable results. A reasonable solution to a multi-objective problem [1] is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution. Evolutionary algorithms are popular approaches to solving multi-objective optimization. The main advantage of evolutionary algorithms, when applied to solve multi-objective optimization problems, is the fact that they typically optimize sets of solutions, allowing computation of an approximation of the entire Pareto front in a single algorithm run. Test data generation is basically the process of identifying a set of data which satisfy the criteria set for testing.

In search based test data generation, the problem of test data generation is reduced to that of function minimization or maximization. Traditionally, for branch testing, the problem of test data generation has been formulated as a minimization problem. An appropriate minimized test suite should exercise different execution paths [2] within a program. However, minimization of test suites may result in significant fault detection loss. The coverage criteria in terms of fault, code and requirements are to be considered for testing. Minimization procedures have to be developed by considering the objectives like maximize fault coverage, minimizing no of test cases, maximizing condition coverage, minimize execution time, minimize cost, and so on simultaneously. Time and space complexities as well as computational complexity are also trying to be reduced. Due to time and resource constraints [3] for retesting the software every time it is modified, it is important to develop techniques that keep test suite sizes manageable by periodically removing redundant test cases and the process is known as test suite minimization.

The rest of this paper is organized as follows: Section 2 describes the related work done using genetic algorithms. Section 3 describes the proposed optimization technique with genetic algorithm. Section 4 describes the implementation of minimization algorithm, section 5 consists of experiment and result analysis and section 6 concludes the paper.

2. Related work

In paper [4], the concept of automatic test pattern generation (ATPG) is implemented using genetic algorithm. In this, the optimized test vector is generated which enhances the fault coverage and improve the global search. The GA operators used are two-point crossover, binary bit string mutation and for selection roulette wheel; Parameters considered for pattern generation are test suite reduction and improved fault coverage. Test pattern is experimented with VLSI circuit testing. Similar type of optimization in test case generation is done [5] using genetic algorithm based on decision coverage. Experimentation has been carried out for a model for smallest number system and in another paper [6] which uses GA for automatic path-oriented test data generation applied for four triangle classification programs.

Model-based test suite minimization using meta-heuristic techniques have been introduced in the thesis [7]. This paper discusses about the generation and minimization of test cases using multi-objective evolutionary algorithms for their selected models. They developed a framework using formal modeling technique for case studies. The parameters taken are test suite size, branch coverage, and weight per test case. Regression test suite minimization using dynamic interaction patterns are discussed in the paper [8]. The idea taken from this [7] is to eliminate redundant test cases by introducing minimization procedure in the generation algorithm, thereby trying to reduce the test suite size. The new bi-criteria heuristic algorithm, using cluster analysis of test cases execution profiles is proposed in this paper [9] which is used for regression testing and in paper [10] discusses about multi-faceted optimization including cost and adequacy values for test cases fitness. The concept of multi-criteria optimization is taken from these papers [9] [10].

In paper [11] presents a tool named HBG_TCS which combines bee colony optimization (BCO) with genetic algorithm in order to reduce the test suite size; results are compared with ACO-based approach for regression testing. Percentage reduction in test suite size and execution time are taken for comparison. The parameters calculated in % are considered while fixing performance metrics. In paper [12] incorporates the same hybridization for regression test suite reduction with the ability to cover all the faults in minimum time and the fault prediction are represented using the binary strings. A hybrid genetic algorithm (HGA) test optimization framework [13] combining GA with local search algorithm has been introduced for improving the quality of test cases and compared with Simple GA.

2.1 Motivation for the work

The following things motivate to do work on test suite minimization by taking test suite size as main criteria:

- Removing redundant test cases thereby reducing the size of test suite
- Reducing execution time

- Generating optimal test cases using GA (fittest individuals)
- Combining evolutionary with other optimization techniques
- Reducing time, space and computational complexities
- Testing object-oriented software

2.2 Problem objective

By considering the reduction in no of test cases and minimal execution time as main criteria for minimization is attempted. Our main goal is to achieve minimal test case selection from the entire set. Path coverage is to be maximized with less execution time requirement. Fault detection capability and the cost can also be considered as sub-objectives. Eliminating redundancy is one of the best way to remove unnecessary test cases from the whole set. Minimization procedures with various resource and time constraints need to be developed. In addition to this, storing the values in truth vectors require memory space and more computation power. Suitable optimization techniques for generation and minimization are considered to test object-oriented programs.

3. Proposed Work

The proposed algorithm for test suite minimization named TestsetMin with minimal computation time and reduced cost has been presented. The algorithm works in this way: For each test case, the no of conditions covered is taken into account along with essential tests. Once the condition has been satisfied, that particular condition is not been tested again; whereas, in the existing techniques, they used truth vectors for storing all the values. It takes more space as well as time. In our procedure, all the conditions covered are moved to the set and then immediately that test case is added to essential tests, at the same time removed from the total test set. So, computations required for finding out the essential and effective test cases are minimized as possible. The input of test cases is generated using GA and the output of genetic algorithm passes through TestsetMin and the final output is a reduced test suite.

The entire process is divided into two phases. First phase is test data generation and optimizing the no. of test cases is the second phase. Data-flow diagram shown in Fig.1 includes the steps involved in optimization process.

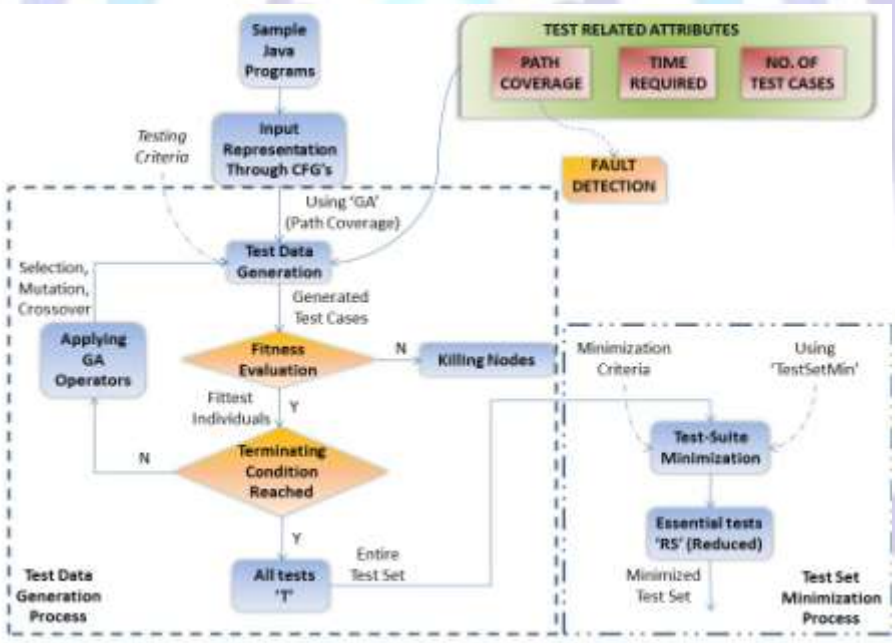

Figure 1.The data-flow diagram for test data generation and minimization

3.1 Input Representation

The input representation is in the form of control flow graphs (CFGs) [15] and the output will be the generated test cases. CFG will give the dependencies between associated variables with specific nodes whereas chromosome is represented using def-use associations. For example, combination of constructor, methods and followed by values are used to form the chromosome which is nothing but a test case.

3.2 Test Data Generation

For test data generation, the global searching algorithm, genetic algorithm is used for testing the object-oriented programs. The structure of a simple genetic algorithm is given below:

Simple Genetic Algorithm ()

{Initialize population;
Evaluate population;
While termination criterion not reached
{Select solutions for next population;
Perform crossover and mutation;
Evaluate population ;}}

Genetic algorithm generates the test data using the genetic operators for selection using tournament, for crossover modified arithmetic and for mutation multi-point random. Multiple objectives like path coverage [2] [6], no of test cases, and the execution time are considered while generating test data.

3.2.1 Tournament Selection

Rank selection technique first sorts the population which is usually considered unattractive for large problems. The tournament selection process involves randomly choosing a group of individuals from the current population, comparing their fitness, and then selecting the fittest from the group for mating. Various tournament selection parameter control schemes have been defined. Examples include fixed tournament size, probabilistic tournament selection, Boltzmann selection with annealing, self-adaptive tournament size and fuzzy tournament selection.

For example, test suite with the values

Test case1 = 3, 7, 9, 2, 4, 5
Test case2 = 2, 8, 5, 6, 3, 1
Test case3 = 0, 4, 3, 7, 2, 5

In fixed tournament size (0.5), the first-half values are same and the remaining is changed to,

Test case1 = 3, 7, 9, 0, 6, 2
Test case2 = 2, 8, 5, 4, 1, 3
Test case3 = 0, 4, 3, 1, 9, 2

3.2.2 Arithmetic crossover

The crossover technique used for getting next generation is modified from the basic arithmetic crossover named as modified arithmetic crossover. Usually, the formula for getting the offspring's is,

Offspring1=a*parent1+ (1-a)*parent2
Offspring2= (1-a)*parent1+a*parent2
where 'a' is a random weighting factor usually 0.5 or 0.7.

But, instead of using all operators, mul-div are used in modified arithmetic. When the value is > or = 0.5, then it is rounded-up to 1.The formula is given below:

offspring1=a*(parent1/parent2)
offspring2=a*(parent2/parent1)

For example, parent1: 5 4 3 4 2
parent2: 6 3 5 1 4

By fixing a = 2, the offsprings are,
offspring1: 2 3 1 8 1
offspring2: 2 2 3 1 4

3.2.3 Multi-point random mutation

This method makes the mutation in several points of the chromosome, the positions where the mutation happens is always varying. Each time new random positions are chosen for mutation.

For example, the test case with five variables,

Parent: 4 1 3 4 2
Offspring: 2 1 6 8 5

After mutation, the resultant chromosome is totally new except some few values. When compared to two-point random, multiple points' variation gives new set of chromosomes for next generation with less number of iterations.

3.3 Evaluating Fitness

The fitness of a chromosome, vi is calculated using the formula,

$$Eval(v_i) = \frac{\text{no of def-use paths covered by } v_i}{\text{total no of def-use paths}} + \frac{\text{no of newly covered path}}{\text{path not yet covered}}$$

3.4 Stopping Criteria

The process is repeated until covering all paths or up to maximum of 1000 generations and is shown in table 1.

Table 1: GA parameters

Operator	*Value*
Selection operator	*Tournament*
Crossover	*Modified Arithmetic*
Mutation	*Multi-point random*
Termination	*Covering all paths or max of 1000 generations*

3.5 Output Specification

The output of GA is the entire test suite 'T', almost all combinations called alltests. After generation of test cases, the whole set has to be minimized in order to get the essentialtests without redundancy.

4. Implementation of Minimization Algorithm

The proposed algorithm named TestsetMin is used for eliminating redundant test cases. Instead of storing the truth vector values the condition coverage for each path are checked for a test case and the satisfied test cases are moved to essentialtests immediately. The process involved in minimization is shown in Fig.2.

```
input: A test suite 'T' with Alltests and Allconditions
output:  Essentialtests: a reduced test suite

algorithm TestsetMin(Alltests, Allpaths, Allconditions)
  {
        TestSuite T = t1,t2 . . . tn ← Alltests;
        ConditionSet C = c1, c2 . . . ck ← Allconditions in a path;
        PathSet P = p1, p2 . . .  pm ← Allpaths;
        Essentialtests, Condcover = { };
while (T != 0) do
                {    for( I = 1 to n)
                       {        for( i = 1 to m )
                                  {                 for each path pi            // actual paths
                                  for(j = 1 to k)
                                  {                 for every condition cj    // in Allconditions
                          check if ((ti covers cj)&&(cj ! in Condcover)) // testcase covering condition cj
                                   {
                                        Condcover = Condcover ∪ {cj};
                                              // covered condition is added to Condcover set
                                        Essentialtests = Essentialtests ∪ { ti };
                                        Alltests = Alltests – { ti };   }
                                        j++;
                                  }
                                  i++;        }
                            I++;        }
                      }
    }                             // end TestsetMin
(To test overlapping conditions, the following steps are to be included)
// take only 3 conditions ci, cj & ck in a path

   if ((testcase1 covering ci & cj) && (testcase1 covering cj & ck)) then
        check if (ck is not covered by any other testcase) then,
                                  select both the testcases 1 & 2
                                  otherwise, select only testcase1
// when no of conditions are more in each path, then the overlapping is to be checked for all the conditions
```

Figure 2.Pseudo-code for TestsetMin Algorithm

4.1 Inputs for Minimization Procedure

The output of genetic algorithm consists of almost all possible combinations with varying fitness values is given as an input to minimization procedure which reduces the no of test cases. After minimization, the resultant test cases are only essential tests by satisfying test requirements.

4.2 Optimization Technique used for minimizing test cases

Initially, the entire test suite is stored in set 'alltests' and conditions are moved to 'allconditions'. In most of the existing techniques, use a truth vector or matrix for saving the status of each and every test case which allow redundancy up to some extent called selective redundancy. But in our approach, first paths have been selected and then the conditions in each path; for each test case, checking the condition coverage if test case satisfies the particular condition, that condition is moved to Condcover remaining conditions will be checked. This mechanism tries to eliminate redundancy thereby tries to minimize the execution time of the algorithm. The execution time of minimization algorithm could be reduced by introducing a simple procedure.

4.3 Effects of using GA with Minimization procedure

The following are the benefits of using test suite minimization procedure with genetic algorithm:

- No of test cases are reduced as possible
- Algorithm steps are reduced as well as simplified
- Overlapping condition checking is included in the procedure
- Execution time is reduced by using simple GA operators
- Simple minimization procedure reduces the space required for storing temporary values
- Reducing the computational complexity

5. Experimental Results

The data taken for result analysis includes both c++ and java programs that are given in the data set. Java objects are taken from SIR (Software Infrastructure Repository). Experimental results are tabulated in table 2 which MGA [14] is the existing algorithm as in the form of tool called mutant gene algorithm taken for comparison. Reduction in test suite as well as time is not that much significant because only small set of programs are taken for experiment.

5.1 Data Set

Initially, the programs with LOC of (35-130), no of variables from (5-7), no of classes max of 3, and no of conditions are less than 28 are taken for testing. For each program, the no of testcases generated 'T', the path coverage (no of conditions covered) and time required for generation are noted after executing hybrid algorithm, HA [GA (search algorithm) with TestsetMin (minimization procedure)]. And then, slowly increasing the no of variables and tested for more no of conditions in a program like Alarm-clock, Producer-consumer and Elevator. In Elevator which has 14 variables with 38 conditions results in 5685ms execution time reduction whereas MGA requires 5850ms. Actually, MGA is in the form of tool calculating the mutant score for each program; so, it is restricted to only simple programs; there is no such restriction in our hybrid algorithm.

5.2 Result Analysis

When no of conditions & variables are more, then no of testcases will also be more; increasing no of testcases is directly proportional to the testing time of a program. Therefore, procedure for minimizing total no of testcases is essential which in turn reduces the testing time; Hybridization of GA with minimization algorithm gives execution time reduction in ms and shown in Fig.4. Hybrid algorithm uses genetic algorithm for generating test cases which in turn uses the GA operators as modified arithmetic crossover, multi-point random mutation and tournament selection; other operators will also be tried for test case generation. The variation in no of testcases is shown in Fig.3; the table 2 records the % of test suite reduction. Reduction in no of testcases is achieved upto 13.6% whereas in existing 11.3% from the initial test cases for multiple stacks program.

Table 2: Results of Java Objects from Software Infrastructure Repository (SIR, 2010)

S. No.	Programs	No of variables	No of conditions	No of generated testcases		
				Initial	Test Suite Reduction	
					Mutant gene alg	Hybrid alg
1.	Binary-search-tree	4	5	120	11.4%	12.6%
2.	Array-partition	5	6	130	11.0%	11.8%

3.	OrdSet	5	7	150	10.4%	11.5%
4.	Binomial-heap	5	7	95	10.7%	11.5%
5.	Multiple stacks	6	18	170	11.3%	13.6%
6.	linkedlist	7	20	140	10.9%	12.7%
7.	Disjoint-set	7	28	120	11.4%	12.9%
8.	deadlock	8	32	105	10.8%	11.6%
9.	Alarm-clock	10	34	140	10.6%	11.2%
10.	*Producer-consumer*	12	37	100	10.7%	11.4%
11.	*Elevator*	14	38	110	10.6%	11.5%

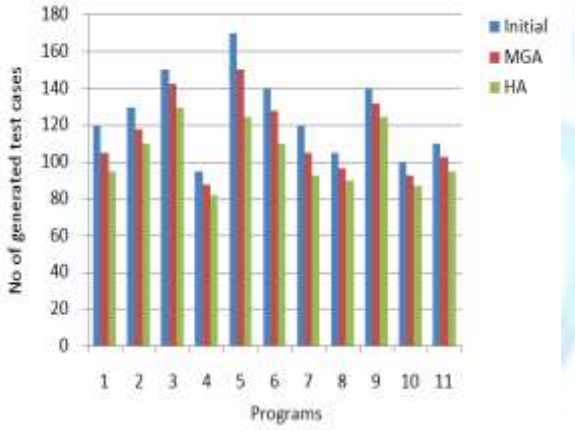

Figure 3.Reduction in no of test cases

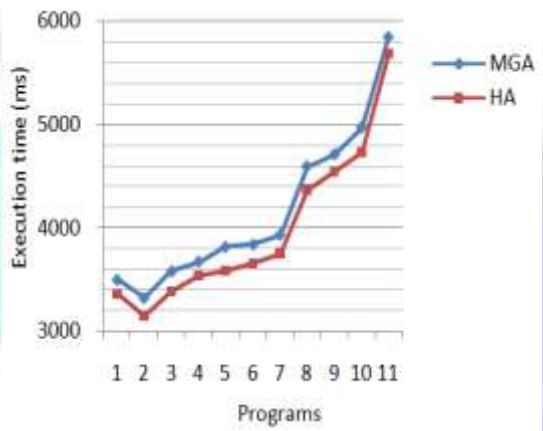

Figure 4.Reduction in execution time (ms)

6. Conclusion

Thus, the procedure for test suite minimization, TestsetMin is introduced in test data optimization where genetic algorithm used for random test data generation; the output of GA is given to the minimization procedure for reducing the total no of generated test cases into essential tests and the combination is named as hybrid algorithm, HA. The results are satisfactory and show significant changes in the size of test suite with minimal execution time requirement. Experiments have been done for simple to medium complexity java programs like Alarm-clock, Producer-consumer and Elevator; reduction in test suite is upto 13.6% and maximum execution time of 5,685ms for the entire test set. When compared to the existing method MGA, hybrid algorithm shows better improvements in test suite minimization as well as execution time reduction.

References

1. Deb, K. (2011) 'Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction', Technical Report 2011003, Indian Institute of Technology Kanpur.
2. Shimin, L. and Zhangang, W. (2011) 'Genetic Algorithm and its Application in the path-oriented test data automatic generation', Advanced in Control Engineering and Information Science, Procedia Engineering 15, pp. 1186-1190.

3. Siavash Mirarab, Soroush Akhlaghi, and Ladan Tahvildari (2012) 'Size-Constrained Regression Test Case Selection Using Multi-criteria Optimization', IEEE Transactions on Software Engineering, Vol. 38, No. 4, pp. 936-956.
4. Rachna, S. and Arvind, R. (2012) 'Implementation of Genetic Algorithm for Automatic Test Pattern Generation', International Journal of Scientific & Engineering Research, Vol. 3, Iss. 4, ISSN 2229-5518, pp. 1-6.
5. Abhishek, S. Swati, C. and Abhay, B. (2012) 'Optimization of Test Cases Using Genetic Algorithm', International Journal of Emerging Technology and Advanced Engineering, ISSN 2250-2459, Vol. 2, Iss. 3, pp. 367-369.
6. Premal B. Nirpal and Kale, K. V. (2011) 'Using Genetic Algorithm for Automated Efficient Software Test Case Generation for Path Testing', International Journal of Advanced Networking and Applications, Vol. 2, Iss. 6, pp. 911-915.
7. Usman Farooq (2011) 'Model Based Test Suite Minimization Using Metaheuristics', Ph.D thesis, School of Computer and Security Science, pp. 1-309.
8. Selvakumar, S. and Ramaraj, N. (2011) 'Regression Test Suite Minimization Using Dynamic Interaction Patterns with Improved FDE', European Journal of Scientific Research, ISSN 1450-216X, Vol. 49, No. 3, pp. 332-353.
9. Alireza Khalilian and Saeed Parsa (2012) 'Bi-criteria Test Suite Reduction by Cluster Analysis of Execution Profiles', International Federation for Information Processing, LNCS 7054, pp. 243-256.
10. Manoj Kumar, Arun Sharma, and Rajesh Kumar (2011) 'Towards Multi-Faceted Test Cases Optimization', Journal of Software Engineering and Applications, Vol. 4, pp. 550-557.
11. Bharti Suri, and Isha Mangal (2012) 'Analyzing Test Case Selection using Proposed Hybrid Technique based on BCO and Genetic Algorithm and a Comparison with ACO' , International Journal of Advanced Research in Computer Science and Software Engineering, ISSN 2277 128X, Vol. 2, Iss. 4, pp. 206-211.
12. Bharti Suri, Isha Mangal and Varun Srivastava (2011) 'Regression Test Suite Reduction using an Hybrid Technique Based on BCO And Genetic Algorithm', Special Issue of International Journal of Computer Science & Informatics (IJCSI), ISSN 2231–5292, Vol. 2, Iss.1, 2, pp. 165-172.
13. Dharmalingam Jeya Mala, Elizabeth Ruby, and Vasudev Mohan (2010) 'A Hybrid Test Optimization Framework – Coupling Genetic Algorithm with Local Search Technique', Computing and Informatics, Vol. 29, pp. 133-164.
14. Selvakumar, S. and Ramaraj, N. (2011) 'A Tool for Generation and Minimization of Test Suite by Mutant Gene Algorithm', Journal of Computer Science 7 (10), ISSN 1549-3636, pp. 1581-1589.
15. Maragathavalli, P. and Kanmani, S. (2012) 'Multi-objective Optimization for Object-oriented Testing using Stage-based Genetic Algorithm', Proceedings of Third International Conference on Advances in Communication, Network, and Computing (CNC'2012), LNICST, Springer, pp. 246-249.
16. Yoo, S. and Harman, M. (2011) 'Regression Testing Minimisation, Selection and Prioritisation: A Survey', Software Testing, Verification and Reliability, Inderscience Journals, pp. 1-60.