Journal of Theoretical and Applied Information Technology
10th March 2015. Vol.73 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPLEMENTATION OF RANDOMIZED TEST PATTERN GENERATION STRATEGY

[1] DR.T.PREM JACOB
[1]Asstt Prof., Department of Computer Science and Engineering, Sathyabama University, Chennai
E-mail: [1]premjac@yahoo.com

ABSTRACT

Software testing involves running a piece of software on selected input data and checking the output for correctness. In order to test the software units randomized testing will be effective. Thoroughness depends on certain parameters settings like relative frequency on which the method are being called. This system that uses genetic algorithm for finding the parameters that optimize the test coverage of random unit testing. For designing the GA we have used the feature subset selection tool for accessing the content representation and the size within genetic algorithm. The proposed system ensures that all possibilities of errors in the code are randomly checked. It is an effective system to detect possibility of runtime errors. The randomized testing using genetic algorithm will call the test cases in a random manner. They will provide the inputs for testing also automatically and it starts testing by calling a test case of random length it will stop the working when it finds any error it can be viewed using minimization. It is faster than conventional testing methods and it is effective in finding the error in the code effectively.

Keywords: *Randomized Testing, Genetic Algorithm, Test Case, Unit Testing, Test Pattern.*

1. INTRODUCTION

Software testing is a complex activity and it is inevitable in ensuring the quality of the software. It accounts for nearly 50% of the total development cost of the software[1]. Knowing this, the software quality managers try to reduce testing costs and time.

Many methodologies and metrics have been developed towards the improvement of the software quality. In order to reduce the number of residual faults, various approaches have been proposed. Software testing detects the presence of faults that cause failure in a program. It is time consuming and expensive task[2][3]. It consumes nearly 50% of the software system development resources.

Software testing involves running a piece of software on selected input data and checking for correctness[4]. The software product is released only after undergoing proper development process such as bug fixing and testing[5].

The randomized testing found faults more often for the same CPU time, and never took prohibitively longer, compared to the conventional test suites that we used. A genetic algorithm is a programming technique that mimics biological evolution as a problem-solving strategy[6][7].

2. MOTIVATION

The objective of testing is to write quality code but doing this requires testing it with inputs to see it behavior. Random testing is a testing in which to test function, f(a,b) randomly select arguments, a and band apply them to f. If there is an error, a bug has been found. Depending upon the dimensionality and domain of f, one might wait a very long time before getting a representative set of inputs to f.

Random Testing is still random, but consciously select new “random” inputs to f() that are “well away” from any previous input attempt to cover the input space of f() in a more intelligent manner. As the title suggest it will generate test cases in a random manner and help in unit testing of the software. Random Testing selects test cases from the entire input domain randomly and independently[8]. The main advantages are, it is intuitively simple and also it allows statistical quantitative estimation of the software’s reliability. In this randomization is used in the selection of target method call sequence and arguments to method calls.

Here genetic algorithm is used to find the parameters for randomized unit testing. Genetic algorithm by the use of fitness evaluation, cross over and mutation find good parameters and it will be to the Randomized unit testing engine[9].

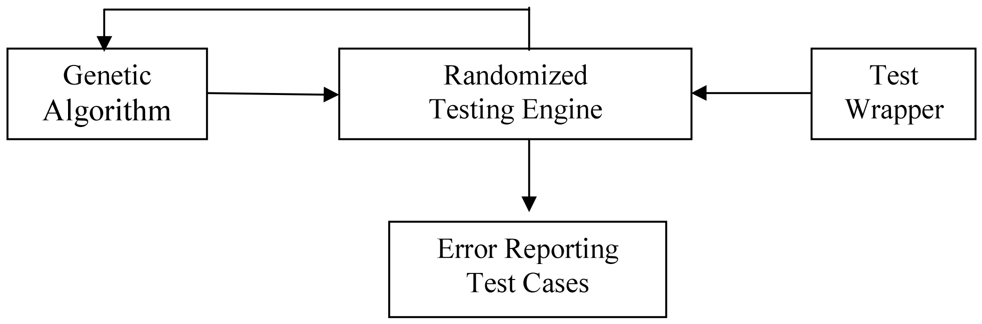

Figure 1: Overall Approaches

Wrapper class is written for each class under testing. Functions having test functions are tested randomly by supplying random inputs using genetic algorithm. Whenever a failure occurs testing is stopped there.

Genetic Algorithm is quite effective for rapid global search of large, non-linear and poorly understood spaces. It is a model of Machine learning. Behavior derived from metaphor of some of the mechanisms of evolution in nature [10]. It is based on population. The main aim is to find the run time errors in a software unit under test.

3. ARCHITECTURE

The Randomized Testing Engine selects a test fragment randomly from the Test wrapper which will make a call to the function to be tested. It then executes the GA. The Genetic algorithm monitors the chromosomes, performs GA operations to create off -springs and retains the chromosome that has the highest fitness ever encountered. This most fit chromosome in each generation is the output of each iteration of GA. The chromosome contains the parameter settings for testing like number of method calls, value pool range etc. After finding the most fit chromosome (based on maximum code coverage), that chromosome is returned to the randomized testing engine.

The Randomized testing engine which takes the chromosome description as input selects the value from the value pool of appropriate data type encoded in best chromosome and invokes the chosen random function from the test wrapper. The testing engine runs test cases until a bug is detected or for a user-specified number of times. Randomized unit testing generates new test cases with new data every time it is run, so if Nighthawk finds a parameter setting that achieves high coverage, a test engineer can automatically generate a large number of distinct, high-coverage test cases in very less time.

4. PROPOSED SYSTEM

GA generating unit tests should search method parameter ranges, value reuse policy, and other randomized testing parameters.
The test runner selects a random test fragment from the test wrapper. Then, it will execute the genetic algorithm by creating an initial population for GA The initial population will be considered as the zeroth generation of GA. The current generation undergoes crossover and mutation operations to create offspring's. From this offspring's the fitter chromosomes are selected as the next population of GA and less fit offspring's are discarded. Selection is done probabilistically based on fitness of the chromosomes.

Simultaneously, the best chromosome among the generated offspring's is returned to the test runner. A chromosome having maximum code coverage is considered as fitter than other chromosomes. The test runner based on the parameter of each test fragment chooses an appropriate value pool from the list of value pools in the chromosome and invokes the test fragment. Each command generated is stored in a data structure which can be reproduced in case the test fails. Thus the test runner creates the test case by choosing values and invoking the methods in test wrapper one by one randomly.

Meanwhile the Genetic Algorithm will be executing in the background returning the next best chromosome to the testing engine to carry out the testing until the code fails or test Case of specified length completes. In the wrapper class testing functions are written[11][12].

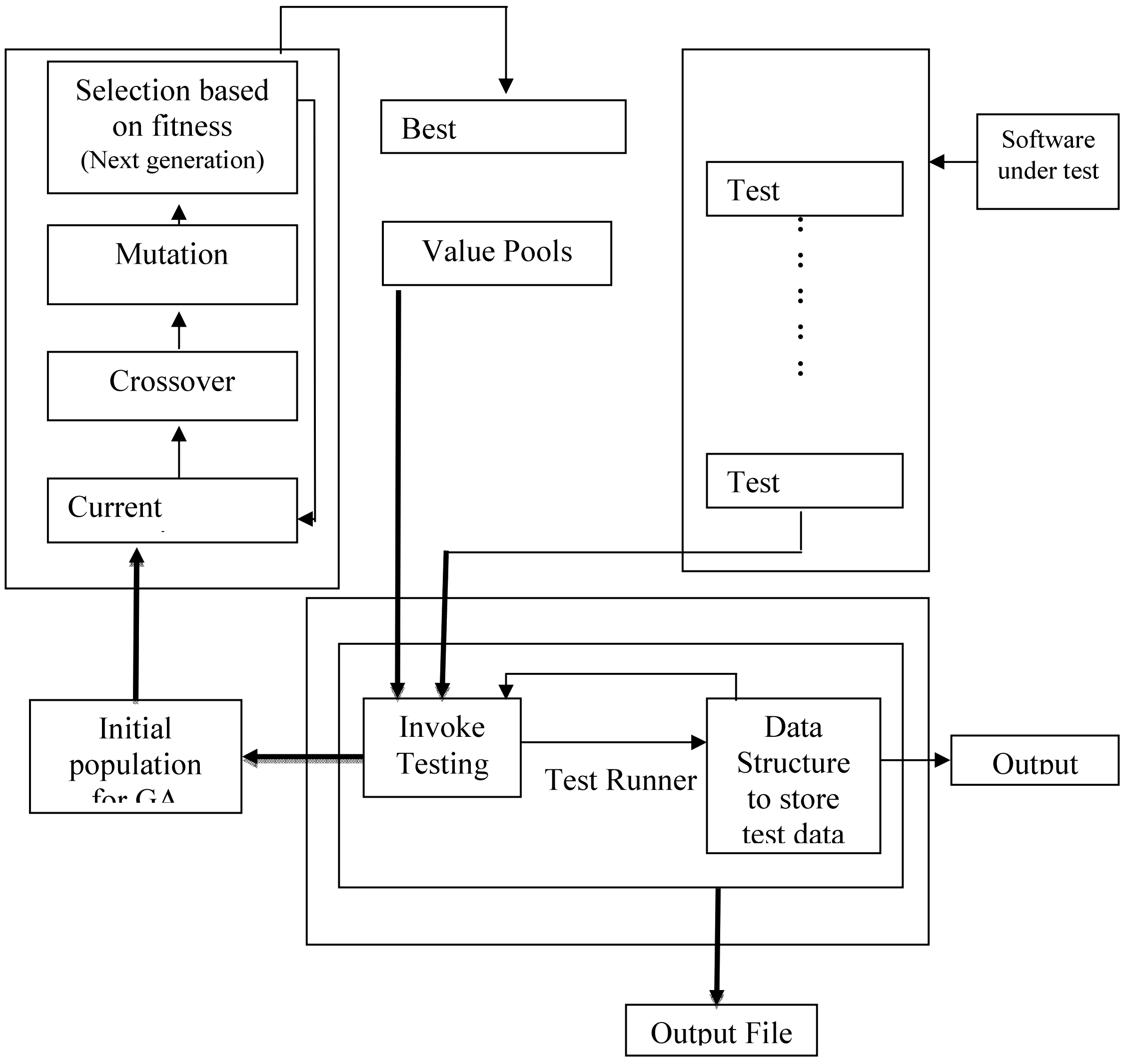

Figure 2: Architecture Of The Proposed System

The Testing Engine will select a method from it and find type of methods, arguments etc. The best chromosome found out using genetic algorithm will select the input values from the value pools. Based on this testing is done and the failure is reported[13][14][15]. Whenever the functions are called in a random sequence, if a bug is found it will stop working there.

5. ALGORITHM

constructRunTestCase

Input: A set M of target methods, a chromosome c

Output: A test case

Steps:

1) For each element of each value pool of each primitive type in IM, choose an initial value that is within the bounds for that value pool.
2) For each element of each value pool of each other type t in IM.
   a) If t has no initializers, then set the element to null.
   b) Otherwise, choose an initialize method I of t, and call tryRunMethod(I,c). If the call returns a non-null value, place the result in the destination element.

3) Initialize test case k to the empty test case.

4) Repeat n times, where n is the number of method calls to perform:

    a) Choose a target method m € CM.

    b) Run tryRunMethod(m,c). Add the returned call description to k.

    c) If tryRunMethod returns a method call failure indication, return k with a failure indication.

5) Return k with a success indication.

It takes a set M of target methods and a chromosome c as inputs. It begins by initializing value pools, and then constructs and runs a test case, and returns the test case. It uses an auxiliary method called tryRunMethod, which takes a method as input, calls the method, and returns a call description. In the algorithm descriptions, the word "choose" is always used to mean specifically a random choice which may partly depend on c.

6. ALGORITHM constructRunTestCase

Input: A method m, a chromosome c.

Output: A call description.

Steps:

1) If m is non-static and not a constructor:

    a) Choose a type t € IM which is a subtype of the receiver of m.

    b) Choose a value pool p for t.

    c) Choose one value recv from p to act as receiver for the method call.

2) For each argument position to m:

    a) Choose a type t € IM which is a subtype of the argument type.

    b) Choose a value pool p for t.

    c) Choose one value v from p to act as the argument.

3) If the method is a constructor or is static, call it with the choosen arguments. Otherwise, call it on recv with the chosen arguments.

4) If the call throws AssertionError, return a failure indication call description.

5) Otherwise, if the call threw another exception, return a call description with an exception indication.

6) Otherwise, if the method return is not void and the return value ret is non null:

    a) Choose a type t € IM that is a supertype of the return value.

    b) Choose a value pool p for t.

    c) If p is not a primitive type, or if t is a primitive type and ret does not violate the p bounds, then replace an element of p with ret.

    d) Return a call description with a success indication.

tryRunMethod considers a method call to fail if and only if it throws an AssertionError. It does not consider other exceptions to be failures since they might be correct responses to bad input parameters. We facilitate checking correctness of return values and exceptions by providing a generator for "test wrapper" classes.

Return values may represent new object instances never yet created during the running of the test case. If these new instances are given as arguments to method calls, they may cause the method to execute statements never yet executed. Thus, the return values are valuable and are returned to the value pools when they are created.

7. RESULT AND PERFORMANCE ANALYSIS

a) Significance of Genetic Algorithm

With the application of genetic algorithm the time taken by randomized testing to detect the bugs was considerably reduced.

Table 4.3: Time For Imoney Unit Testing With And Without GA

Without GA (in sec)	With GA (in sec)
0.109	0.046
0.094	0.047
0.063	0.047
0.047	0.094
0.078	0.063
0.094	0.063
0.078	0.031
0.078	0.046
0.047	0.078
0.156	0.040
0.0844 s	**0.0555 s**

This table shows the significance of using GA by doing time comparison. This is done by performing ten runs.

After the ten runs when we take the average time, the time taken for testing without GA is 0.0844 seconds.

The time taken for testing using GA is 0.0555 seconds. From this it is clear that time for testing using GA is less compared to the testing without GA.

Table 4.4: Time Taken For Testing Of Bank Account With And Without GA

Without GA (in sec)	With GA (in sec)
0.078	0.063
0.031	0.047
0.015	0.062
0.016	0.016
0.125	0.047
0.031	0.031
0.063	0.078
0.063	0.062
0.062	0.031
0.047	0.016
0.0531 s	**0.0453 s**

Table 4.5: Performance Analysis For Test Cases With And Without GA

Random without GA	Random GA
Test case I Timing **0.0844**	Test case I Timing **0.0555**
Test case II Timing **0.0531**	Test case II Timing **0.0453**
Needs manual intervention in designing value pools to detect some bugs in test case II a) Unable to detect Withdraw()/Deposit() bug with a zero amount b) Unable to detect Withdraw()/Deposit() bug with -ve amount	Automatic detection

8. CONCLUSION AND FUTURE ENHANCEMENT

Software units are tested effectively by randomized testing. The main aim of testing is to write quality code but doing this requires testing it with inputs to see it behavior r test. This system is able to find the run time errors in a code. Future enhancement can be done in the system by using some methods of fuzzy system in it. It can be applied to generate efficient value pools and cross over rules. Another enhancement is the same framework may be ported to suit to other programming languages like c, c++, .net etc.

REFRENCES:

[1] Prem Jacob .T and Ravi .T, 2013, "An Efficient Method for Regression Test Selection", International Journal of Software Engineering and Technology, Vol.5, No.6, pp.218-222.

[2] Prem Jacob .T and Ravi .T, 2013, "Optimization of test cases by prioritization", Journal of Computer Science, Vol.9, No.8,
pp.972-980.

[3] Prem Jacob .T and Ravi .T, 2013, "Optimal Regression Test Case Prioritization using genetic algorithm", Life Science Journal, Vol.10, No.3, pp.1021-1033.

[4] Prem Jacob .T and Ravi .T, 2014, "A Novel Approach for Test Suite Prioritization", Journal of Computer Science, Vol.10, No.1, pp.138-142.

[5] Prem Jacob .T and Ravi .T, 2013, "Regression Testing: Tabu Search Technique for Code Coverage", Indian Journal of Computer Science and Engineering, Vol.4, No.3, pp.208-215.

[6] Prem Jacob .T and Ravi .T, 2013, "An Optimal Technique for Reducing the Effort of Regression Test", Indian Journal of Science and Technology, Vol.6, No.8, pp.5065-5069.

[7] Wappler, 2006, "Evolutionary unit testing of object-oriented software using a hybrid evolutionary algorithm", In Evolutionary Computation, pp.851-858.

[8] Last, 2006, "Effective black-box testing with genetic algorithms", In Hardware and Software, Verification and Testing, Springer Berlin Heidelberg, pp.134-148.

[9] Srivastava, 2009, "Application of genetic algorithm in software testing", International Journal of software Engineering and its Applications, Vol.3, No.4, pp.87-96.

[10] Jones, 1996, "Automatic structural testing using genetic algorithms", Software Engineering Journal, Vol.11, No.5, pp.299-306.

[11] Ribeiro, 2008, "A strategy for the evaluation of feasible, unfeasible test cases for evolutionary testing of the object-oriented software", Proceedings of the 3rd international workshop on Automation of software test, pp.85-92.

[12] Rajappa, 2008, "Efficient software test case generation using genetic algorithm based graph theory", In First International Conference on Emerging Trends in Engineering and Technology, pp.298-303.

[13] Gupta, 2008, "Using genetic algorithm for unit testing of object oriented software", In Emerging Trends in Engineering and Technology, First International Conference, pp.308-313.

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

[15] Doungsard, 2006, "An automatic test data generation from UML state diagram using genetic Algorithm", In Proceedings of International Conference on Software, Knowledge, Information Management and Applications, pp.1-5.