International Journal Of Computer Science And Applications Vol. 1, No. 1, June 2008
ISSN 0974-1003

Intelligent Test Case Optimizer - An automated Hybrid Genetic Algorithm based test case optimization framework

D.Jeya Mala
Lecturer, Dept.of Computer Applications, Thiagarajar College of Engineering, Madurai, Tamil Nadu, India
djmcse@tce.edu

S.Ruby Elizabeth
Software Engineer, Tata Consultancy Services, Chennai, Tamil Nadu,India.
rubielizabeth.s@tcs.co

Dr.V.Mohan
Professor & Dean, Thiagarajar College of Engineering, Madurai,Tamil Nadu India

ABSTRACT
In software testing process, generation and selection of test cases requires a lot of human intervention. Such manual testing process leads to ineffective software testing process which in turn increases the total time and cost needed in it. To deliver zero-defect software, the number of test cases needed to test the software will be infinite. Since exhaustive testing is not possible, we need to reduce the number of test cases by selecting the optimal test cases that have the high potential of identifying the errors in Software under Test (SUT). Hence, we need to generate the optimal test cases, which are able to identify the artificially seeded faults or errors (Mutation Score) in the Software under Test (SUT). In our approach, we proposed a Hybrid Genetic Algorithm (HGA) based novel testing methodology that can generate optimal test cases from the set of test cases based on the mutation score (the number of artificially seeded errors in the Software) that achieves high statement coverage criterion by finding more seeded bugs in the SUT. The reduction in the number of test cases by means of Hybrid Genetic Algorithm based optimization results in the saving of testing resources. Finally, we compared Simple Genetic Algorithm (SGA) based test case optimization approach with our approach and proved that, HGA based approach is producing better optimization results.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Quality.

General Terms
Measurement and Verification.

Keywords
Software Testing, SUT (Software Under Test), Test Optimization, Simple Genetic Algorithm (SGA), Hybrid Genetic Algorithm (HGA), and Mutation Score.

1 INTRODUCTION
The application of Evolutionary Computation (EC) techniques in Software Engineering (SE) is an emerging area of research that brings about the cross fertilization of ideas across two domains [6]. A number of published works, for examples [2] [3] and [9], have begun to examine the effective use of EC for SE related activities which are inherently knowledge intensive and human-centered.

Some of the research works applied Simple Genetic Algorithms (SGA) for test data generation [4] [13]. In Simple Genetic Algorithm (SGA), although the search is not exhaustive, the algorithm can get stuck in local extreme, thus fails to find a global optimum of the fitness function. In other words, GA is usually very fast in finding a good solution, but in general they will not find the best solution [2]. In our paper, we concentrate on the optimal generation of test cases from the solution space using HGA (Hybrid Optimization Algorithm).

The method such as Genetic Algorithm (GA) is inefficient for fine-tuning solutions once a near global minimum is found. For problems that contain several local minima, a hybrid approach starting with a global method and then fine-tuning with a local method may be more attractive, especially if the decision space is reasonably well behaved near the solution [10][15]. Hence, in this paper, we implemented HGA based test case generation and optimization framework using Java. Also, we compared SGA (Simple Genetic Algorithm) and HGA (Hybrid Genetic Algorithm) methods for generating the optimal test cases from the repository of test cases and proved that our framework achieves higher code coverage criterion and higher mutation score for error detection.

2 HYBRID GENETIC ALGORITHM (HGA) – INTRODUCTION
The Hybrid Genetic Algorithm (HGA) is also called as Memetic algorithm. It is a population based approach for heuristic search in optimization problems. They are more efficient than Simple Genetic Algorithms (SGAs) in which a local search algorithm is also included so that, the final result will have global optima [11]. Search based approaches have been adopted for test case generation in highly safety critical systems [14]. HGA combines both GA and local search (GA-LS) approaches. They are generally more effective than using stand alone versions of each

method [7]. They execute order of magnitude faster than traditional Genetic algorithms [12].

Here the generations are called as memes not as genes and they are processed and improved based on the local search algorithm employed by the individuals. They are used to solve complex problems in which the shortest path to the goal to be identified. They are also called as meta-heuristic algorithms [11].

3 PROPOSED APPROACH – HYBRID GENETIC ALGORITHM BASED TEST CASE GENERATION AND OPTIMIZATION FRAMEWORK

The effectiveness of the test cases is depending on the application, testing approach or the availability of the testing resources. Optimizing the test case generation is one of the ways to reduce the number of test cases and improving the effectiveness of test cases. In our approach, we employed Hybrid Genetic Algorithm for optimal generation of test cases.

In the proposed framework the Software under Test (SUT) is given as input. The framework has a mutant generator which generates the mutated versions of the given SUT. Based on coverage analysis and mutation score analysis the test case is either selected or rejected [5].

The algorithm works as below:

Step 1:

Initialize population randomly

Step 2:

1. Apply RemoveSharp algorithm to all test cases in the initial population

2. Apply LocalOpt algorithm to all test cases in the initial population

Step 3:

- Select two parents based on the mutation score.
- Apply Crossover between parents and generate an offspring
- Apply RemoveSharp algorithm to offspring.
- Apply LocalOpt algorithm to offspring.
- If Mutation Score(offspring) > Mutation Score(any one of the parents) then replace the weaker parent by the offspring

Step 4:

Mutate the selected test cases / parents to generate the next population

Step 5:

Repeat steps 3 and 4 until end of specified number of iterations.

A Hybrid Genetic Algorithm is designed to use heuristics for improvement of off-spring produced by crossover and mutation. Initial population is randomly generated. The off-spring is obtained by n-point crossover and mutation between two selected parents.

The improvement heuristics RemoveSharp and LocalOpt are used to bring the offspring to a local maximum [7]. If the fitness of the offspring thus obtained is greater than the fitness of any one of the parents then the parent with lower fitness is removed from the population and the offspring is added to the population. If the fitness of the offspring is lesser than that of both of its parent then it is discarded.

3.1 Chromosomal Representation

Here each meme is considered as a test case. Each chromosome represents a legal solution to the problem and is composed of a string of memes.

In software testing we represent the chromosome as the stream of methods to be called in each class along with the arguments.

For example, the chromosome of Binary Search Tree Algorithm construction program may be represented as follows:

(insert (3), search(1),insert(1),search(0))

Where insert and search are methods and the numbers in the brackets represents the value to be inserted or searched and is passed as an argument to the method.

When selecting a particular method with a value, it must produce the right result. The sequence of operations are given here as the test cases. In our case it is insert and search. They are combined for checking different conditions in the binary search tree.

3.2 Initial Population

Once a suitable representation has been decided upon for the chromosomes, it is necessary to create an initial population to serve as the starting point for the genetic algorithm. This initial population is usually created randomly.

3.3 Crossover and Mutation

Step1: Apply Multipoint cross over on parents to generate a population of off springs.

Step 2: Select two optimal off springs from the generated off springs based on their mutation score using local search technique.

Step 3: Replace a parent with the optimal individual by using the following condition:

If mutation_score(offspring) > mutation_score(parent) then replace the parent with offspring

Else retain the existing parent.

Step 4: Mutate the selected parent before going to the next generation.

Step 5: Repeat steps 1 to 4 till the termination condition is reached.

3.4 Selection

We need to select chromosomes from the current population for reproduction. If we have a population of size 2N, the selection procedure picks out two parent chromosomes, based on their fitness values, which are then used by the crossover and mutation operators (described below) to produce two offspring for the new population [15].

ISSN 0974-1003

Fitness evaluation involves defining an objective or fitness function against which each chromosome is tested for suitability for the environment under consideration [1]. As the algorithm proceeds we would expect the individual fitness of the "best" chromosome to increase as well as the total fitness of the population as a whole. We have chosen the mutation score as the fitness value.

Let's assume that we have a perfect test suite, one that covers all possible cases. Let's also assume that we have a perfect program that passes this test suite. If we change the code of the program which is called as *mutation* and we run the mutated program which is called as a *mutant* against the test suite, we will have two possible scenarios:

- The results of the program were affected by the code change and the test suite detects it. If this happens, the mutant is called a killed mutant [8] [9].
- The results of the program are not changed and the test suite does not detect the mutation. The mutant is called an equivalent mutant [8] [9].

The ratio of killed mutants to the total mutants minus equivalent mutants provides the measures on how sensitive the program is to the code changes and how accurate the test suite is [8].

Mutation score = (dead mutants)/(total mutants – equivalent mutants) * 100

We used the tool *Jester* [16] to create mutants in the software and mutation score is extracted from the resultant XML file to judge the fitness of test suite.

3.5 Remove Sharp Algorithm

Mutation Score of all the offspring produced by the n point crossover is calculated. The offspring which are leaving the more number of mutants in survival will be deleted from the memory. That is the test cases having very less mutation score will be left. Test cases having higher mutation score will be memorized.

3.6 Local Opt Algorithm

At every generation of offspring by the n point crossover, the offspring which is having highest mutation score is selected as local optima. This local optimal solution compared against the 2 parents. If any of the offspring better than the any one of the parent then the weakest parent will be replaced by the optimal offspring.

4 IMPLEMENTATION AND RESULTS

The bench mark problem we took here is a "Binary Search Tree Algorithm". We considered that it has two methods namely search and insert. We should generate test cases to identify whether the elements are inserted at the right position in the tree and whether searching of elements are properly done.

We implemented the Test case optimization by using both Simple Genetic Algorithm and Hybrid Genetic Algorithm using Java. Seed population is randomly generated and 30 generations are generated by using SGA and HGA. Mutation score of each test case is calculated using Jester tool (which is used to test the test cases written in junit). The output produced by Jester tool is an XML file that has the mutation score of each of the test cases generated.

We developed XML parser code using Java to parse the XML output file for getting the mutation score. This is used as input for the Local Optimum and Remove Sharp algorithms. Then by applying our proposed algorithm, we finally arrived at an optimal test suite when the termination condition is reached.

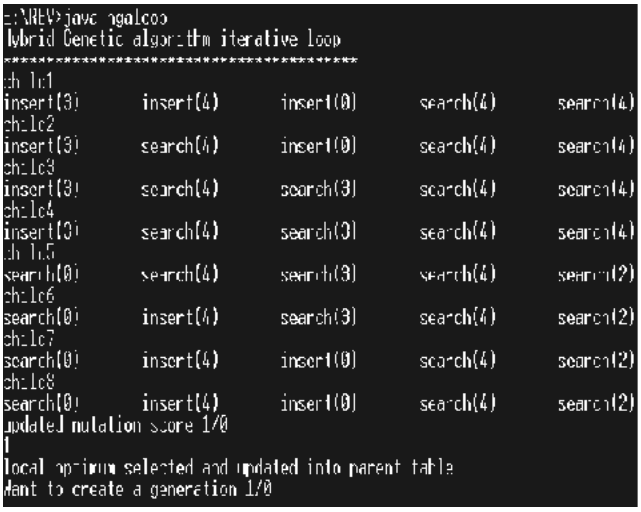

Figure 1 - Initial Population Generation

Cross Over Results:

Parent1: insert(3), search(4), insert(0), search(4), search(2)

Parent2: search(0), insert(4), insert(0), search(4), search(4)

offspring or children:

Parent 1 with 2:

1. insert(3),insert(4),insert(0),search(4),search(4)
2. insert(3),search(4),insert(0),search(4),search(4)
3. insert(3),search(4),search(3),search(4),search(4)
4. insert(3),search(4),search(3),search(4),search(4)

Parent 2 with 1:

1. search(0),search(4),search(3),search(4),search(2)
2. search(0),insert(4),search(3),search(4),search(2)
3. search(0),insert(4),insert(0),search(4),search(2)
4. search(0),insert(4),insert(0),search(4),search(2)

Figure 2 - Local Optimal Algorithm Implementation

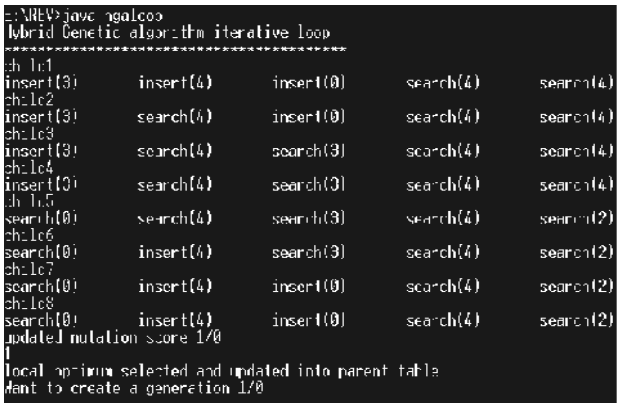

Figure 3 - Final Optimal Test Case Generation

4.1 Comparison - SGA Vs. HGA

Mutation score of each test case generated using SGA and HGA are listed in the table 1. The comparison chart shows that, SGA is striking at local optima and is also suffering a lot of fluctuations whereas HGA shows the gradual improvement in the generation of test cases having higher mutation score.

Table 1: Mutation Score for SGA and HGA

Generation	GA (ms)	HGA(ms)
0	13	36
1	8	37
2	20	52
3	31	52
4	37	52
5	49	64
6	46	64
7	36	64
8	40	64
9	34	68
10	22	72

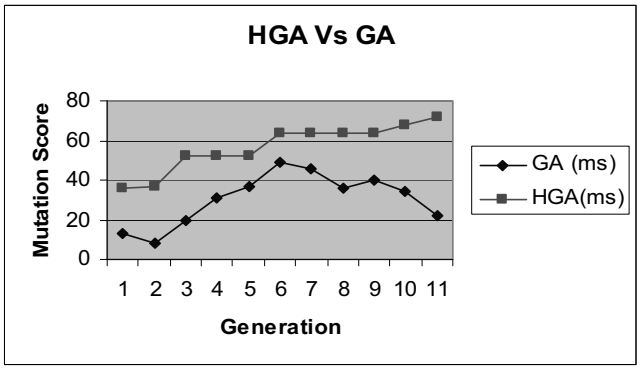

Figure 4 - Comparison Chart of SGA Vs. HGA

5 CONCLUSION

Our proposed approach has proved that, HGA will be a better solution for Test Case Optimization problems. By using HGA we can improve the cost and time factors in automated testing process. Less human intervention makes this technique an intelligent one. As a future enhancement, we need to compare HGA against Bacteriologic algorithm which is having the swarm intelligence in it and by means of that; we can identify how for HGA will be suitable in producing optimized test cases in test optimization.

6 ACKNOWLEDGMENTS

We thank the Lord Almighty for finishing this work and developed the framework as a complete tool. Also, we would like to extend our regards to our institution for providing us enough resources to complete this work.

7 REFERENCES

[1] Andrew Baresel, Harman Sthamer, Michel Schmidt, ”Fitness function design to improve evolutionary testing”, SBSE-2001

[2] Benoit baurdy , Frank fleurey, Jean marc and Yves Le Traon, ”Automatic test case optimization: A Bacteriologic algorithm”, Mar/Apr 2005 IEEE software,pp-76-81

[3] Briand, L. C.,“On the many ways Software Engineering can benefit from Knowledge Engineering”, Proc. 14th SEKE, Italy, pp. 3-6, 2002

[4] “EvoTest : Test Case Generation using Genetic Programming and Software Analysis”- Thesis by Arjan Seesing, Delft University of Technology

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

[6] Huaizhong Li and C.Peng Lam, “Software Test Data Generation using Ant Colony Optimization”, Transactions on Engineering, Computing and Technology, 2004, ISSN 1305-5313.

[7] G.Andal Jayalakshmi, S.Sathiamoorthy ,R.Rajaram, ,”A Hybrid Genetic Algorithm- A new approach to solve traveling salesman problem”, Proceedings of DETC’02 ASME 2002 Design Engineering Technical Conferences and Computers and Information in Engineering Conference

[8] Jeromy S.Bradbury, “Using Mutation for the Assessment and optimization of tests and properties”, ACM, 2006

[9] Louire Williams, “Mutation Testing”, NC State University, Offutt & Untch, Mutation 2000: Uniting the Orthogonal, 2004

[10] Li, H., Lam, C.P., “Optimization of State-based Test Suites for Software Systems: An Evolutionary Approach”, International Journal of Computer & Information Science, Vol. 5, No. 3, pp. 212-223, 2004

[11] Natalio Krasnogor and Jim Smith, “A Tutorial on Competent Memetic Algorithms: Model, Taxonomy and Design Issues”, IEEE Transactions on Evolutionary Computation, Vol.ANo.B.CCC200D, 2005

[12] Pedrycz, W., Peters, J. F., "Computational Intelligence in Software Engineering", World Scientific Publishers, 1998.

[13] Roy P. Pargas, Marry John Harrold, Robert R.Perk, "Test data generation using genetic algorithm" Journal of Software testing, verification and reliability, 1999,pp-1-19

[14] Tracey, N., Clark, N., .Mander K., and McDermid, N., "A Search Based Automated Test Data Generation Framework for Safety Critical Systems", in Systems Engineering for Business Process Change (New Directions), Henderson P., Editor, Springer Verlag, 2002

[15] Vincent Kelner, Florin Capitanescu, Oliver Leonard and Louis Wehenkel, "An Hybrid Optimization Technique coupling Evolutionary and Local Search Algorithms", 2004

[16] www.jester.com