Performance-Oriented Software Testing Through Genetic Algorithm–Based Automatic Test Case Generation

Anil Kumar[1], Neha Agarwal[2]
[1] Professor, Department of Computer Applications, JECRC University, Jaipur, India
[2] Research Scholar, JECRC University, Jaipur, India

ABSTRACT
Software testing is most effort consuming phase in software development. One would like to minimize the efforts and maximize the number of faults detected. Hence test case generation may be treated as an optimization problem. One of the major difficulties in software testing is the automatic generation of test data that satisfy a given adequacy criterion. Generating test cases automatically will reduce cost and efforts significantly. In this paper, test case data is generated automatically using Genetic Algorithms and results are compared with Random Testing. It is observed that Genetic Algorithms outperforms Random Testing.

KEYWORDS: Automatic test case generation, Equivalence Class Partitioning, Random Testing etc.

I. INTRODUCTION

Testing is a process which needs to be done effectively. Exhaustive testing is not possible due to limitation of resources. In past, it is observed that test cases lie in different classes. Equivalence classes are to form a partition of set, where partition refers to a collection of mutually disjoint subsets where the union is the entire set [1] [2]. This has two important implications for software testing: the fact that the entire set is represented provides a form of completeness and disjointedness ensures a form of non-redundancy. As the subsets are determined by a equivalence relation, the elements of one subset have something in common [3, 4, 5]. So the idea is to identify test cases by using one element from each equivalence class. If the classes were chosen wisely, it greatly reduces the potential redundancy in test cases. For example for an equilateral triangle test case, if one chose (3, 3, 3) as test case, then one would not expect to learn much from (6, 6, 6) or (50, 50, 50). The key of equivalence class testing is the choice of the equivalence relation, which partition the classes. For the sake of drawings, a function F of two variables x1, x2 will be used. For example, Equivalence class partitions for a nextDate module, which return the very next date of the entered current date [6] [7]. To solve optimization problems there are a number of techniques and one of them is Genetic Algorithms. Genetic algorithms are population based search based on the Darwin's principle of survival of the fittest. Genetic Algorithm is basically an evolutionary technique inspired by biological evolution. It was developed in 1970's by J. Holland and his colleagues and his students at University of Michigan's [10] [12]. It mimics the process of natural evolution. Genetic Algorithm starts with an initial population and then applies genetic operators like selection, crossover, mutation and replacement on that population to evolve better and better individuals. Genetic Algorithm can be terminated in either of two cases: maximum number of generations achieved or optimum value found. It can be done as under:
It is a function of three variables and the boundaries are as follows:
M1 = month (1 <= month <=12)
D1 = date (1 <= date <=31)
Y1 = year (1951 <= year <= 2051)
The invalid equivalence classes were:
M2 = month < 1   M3 = month > 12

D2 = date < 1	D3 = date > 31
Y2 = year < 1951	Y3 = year > 2051

So the robust test cases with equivalence class test may be as under:

Table 1: Robust test cases for Equivalence Class Partitioning

Month	Date	Year	Remarks
5	15	1962	All valid inputs
-1	15	1962	M2 class
15	15	1962	M3 class
5	-1	1962	D2 class
5	45	1962	D3 class
5	15	1900	Y2 class
5	15	2100	Y3 class

In this research, the researcher carried out the identification of these boundaries/intervals automatically through Genetic algorithm and random testing and then compares the results of both techniques. Genetic algorithm and random testing both starts with some random initial population and then Genetic Algorithm use the fitness of individuals to progress towards the optimums, whereas random testing works randomly throughout the run. For this experiment, the distance from the boundaries is taken as the fitness of the individual chromosome. Unlike the boundary value analysis approach where this distance should be minimized, here the optimal distance is the distance between the boundary and a point p, where the location of p is to be somewhere in the middle of the two boundaries. The algorithms are coded in MATLAB 2016a.

II. Genetic Algorithm for test case generation

The proposed genetic algorithm for test case generation for Equivalence Class Partitioning is presented here. Firstly, the major components of Genetic Algorithm are discussed and then overall algorithm is presented. In proposed Genetic Algorithm value encoding is used in the chromosome i.e. real values are used to represent the input variables x1,x2,.... of the program. The length of the chromosome depends on the number of variables [14] [15] [19]. Initial population is generated randomly. Fitness of each chromosome is determined by its difference from the boundaries of the variable. The optimal distance is the distance between the boundary and a point p, where the location of p is to be somewhere in the middle of the two boundaries. The more a variable is close to the boundaries the more it is declared fit. Selection is done to select parents for reproduction [16]. There are many methods to do this process. Roulette wheel and Random selection were used for Genetic Algorithm and Random testing respectively in experiments.

- Rank Selection: It is a selection process based on fitness of an individual and total cumulative fitness.
- Random Selection: This selection in pure random, no knowledge is used in selection process.

Crossover create new child from existing parents. It is one of important operators of genetic algorithm. Crossover operator used in experiments is arithmetic crossover.

Following parameters are used in experiments:

1. Population Size: various population sizes are tried and best ones are taken for comparison i. e. 50 & 100.

2. Generations: program is executed with different number of generations and analysis of less number of generations and more number of generations is also taken into consideration i. e. 200, 500.
3. Encoding: chromosomes (test cases) are coded in real values, so Value encoding scheme of Genetic Algorithm is used.
4. Selection: Rank selection is used for Genetic Algorithm, and Random selection is implemented for Random Testing.
5. Crossover: number of crossovers available for real value coding, out of which arithmetic crossover is applied with 0.9 probability.
6. Mutation: uniform mutation is applied in experiments with 0.01 probability.
7. Replacement: Simple genetic algorithm replacement takes place, in which whole new population replaces the old one.

III. Results & observations

All inputs are taken from user, so that testing with different parameters can be done easily. User interface while running in MATLAB is as follows: -
Inputs:
No. of individuals in population : 50, No. of Variables : 2, No. of Generations : 500
limits of 1st variable : Lower limit : 10 & Upper limit : 20
limits of 2nd variable : Lower limit : 10 & Upper limit : 20
limits of 3rd variable : Lower limit : 10 & Upper limit : 20
Outputs: With Genetic Algorithm, test cases generated as follows:
Table 2: Genetic Algorithm versus random testing for ECP

	Genetic Algorithm			Random Testing		
Variables / Runs	Variable 1	Variable 2	Variable 3	Variable 1	Variable 2	Variable 3
1	10.2	11.5	9.9	9.1	8.4	16.7
2	19.5	10.3	8.9	8.0	12.7	24.5
3	12.4	11.6	19.8	19.8	12.4	11.5
4	12.8	9.7	18.7	11.7	14.5	7.8
5	10.7	20.8	19.7	18.6	11.6	23.6

Figure 1 to **Figure 5** shows the results of these executions as under:

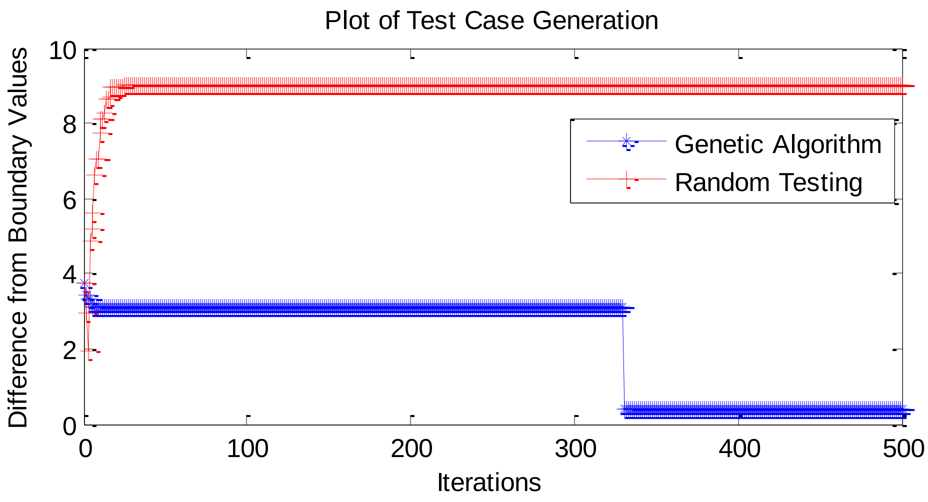

Figure 1: Genetic Algorithm versus random testing refer to row 1 of Table 2

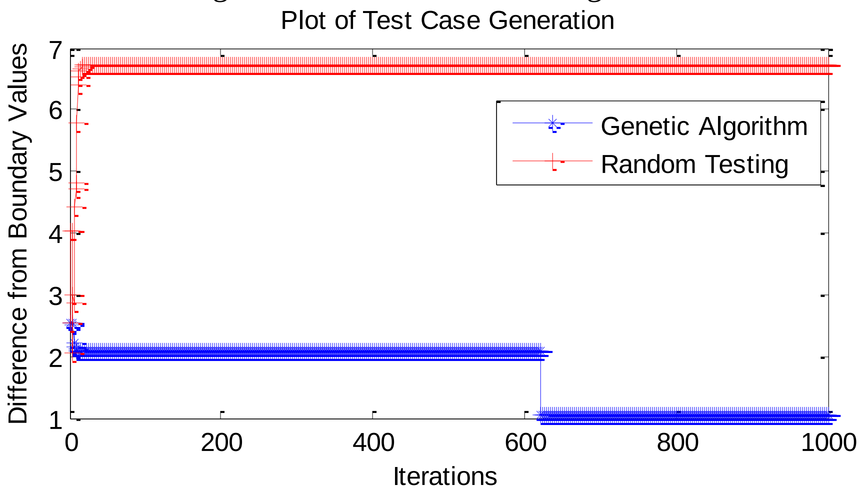

Figure 2: Genetic Algorithm versus random testing refer to row 2 of table 2

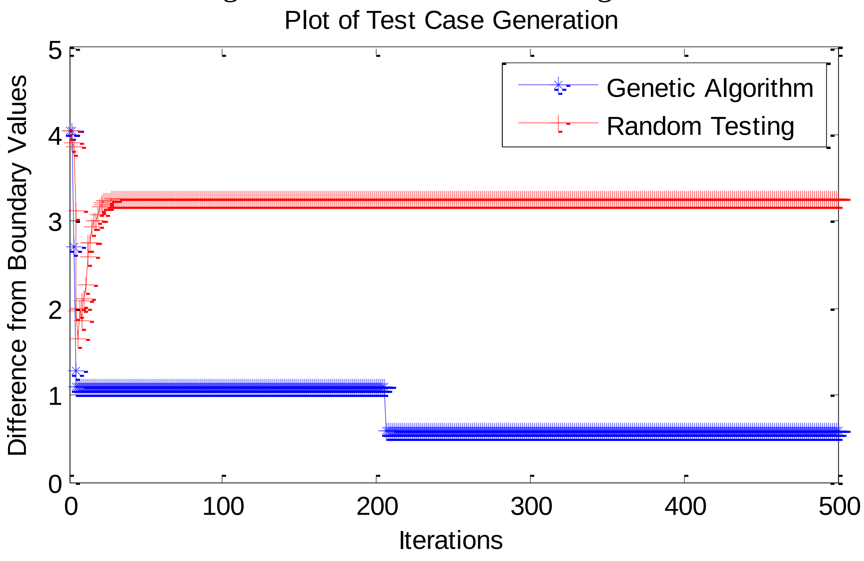

Figure 3: Genetic Algorithm versus random testing refer to row 3 of table 2

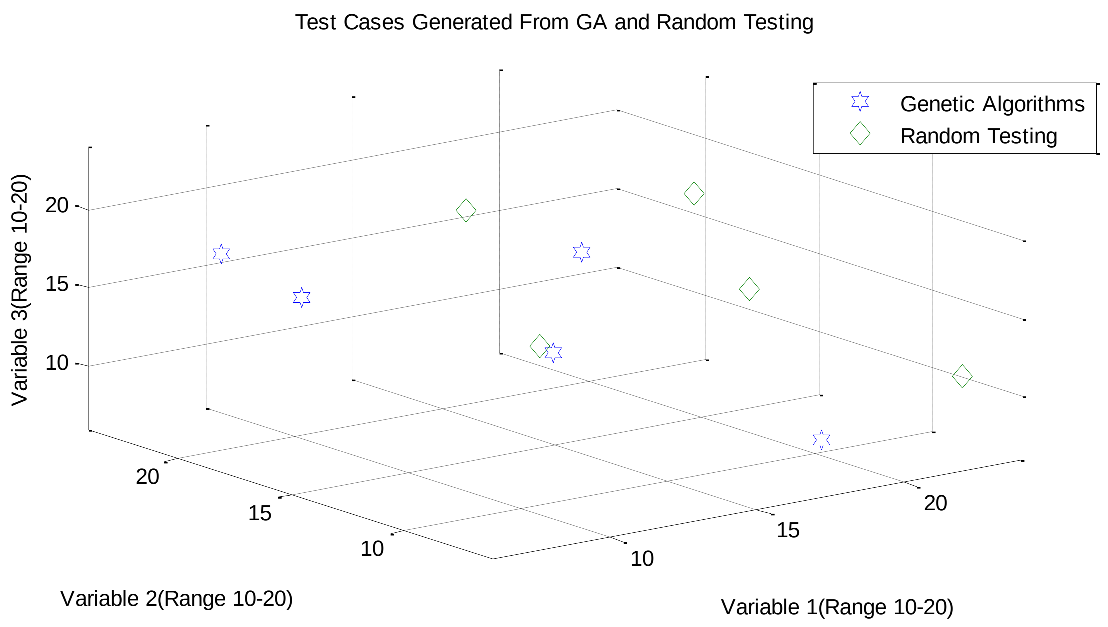

Figure 4: Results showing the final test cases analysis

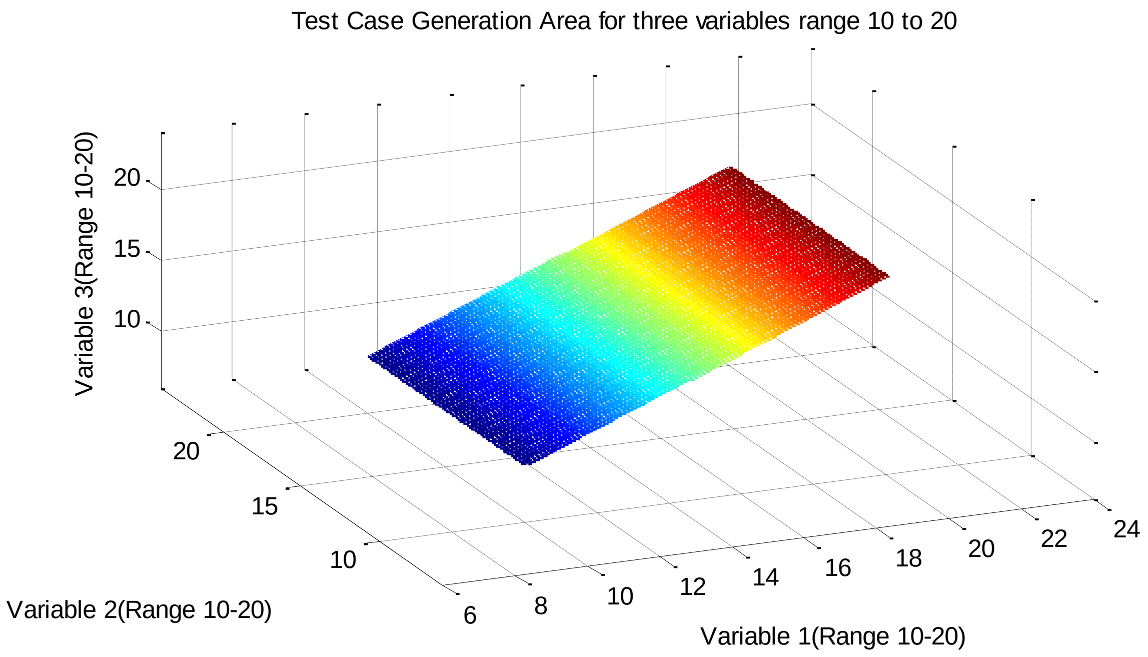

Figure 5: The area under the three variables as per their ranges (10-20)

It is clearly visible from experiments that test cases with genetic algorithms spread over all classes, whereas random testing will generate test cases for small number of equivalence classes. So one can use Genetic algorithm to generate test cases automatically and get better and useful test cases as outputs.

IV. CONCLUSION

The overall results show evolutionary testing to be a promising approach for fully automating test case design for equivalence class partitioning technique of testing. To increase the efficiency and effectiveness, and thus to reduce the overall development cost for software-based systems, a systematic and automatic test case generator is required. Genetic algorithms search for relevant test cases in the input domain of the system under test. Due to the full automation of test case generation, the overall quality of software is also enhanced in comparison of using random testing.

V. REFERENCES

[1] Beizer B. “Software Testing Techniques”. Van Nostrand Reinhold, 2nd edition, 1990.

[2] Jorgenson C. Paul, “Software Testing: A Craftsman Approach”, CRC Press, 2nd Edition 2002 pp 97-103.

[3] Bertolino, A.: 'An overview of automated software testing', Journal Systems Software, Vol. 15, pp. 133-138, 1991

[4] Deason, W. H., Brown, D. B., Chang, K. H., and II, J. H. C. (1991). “A Rule-Based Software Test Data Generator”. IEEE Trans. on Knowl. and Data Eng., 3(1):108–117.

[5] Dijkstra, E. W., Dahl, O. J., Hoare, C. A. R.: “Structured programming”, Academic Press., 1972.

[6] Duran J. W. and Ntafos S. C.: 'An Evaluation of Random Testing', IEEE Transactions on Software Engineering, Vol. SE-10, No. 4, pp. 438-444, July 1984

[7] Duran, J. W. and Ntafos S., 'A report on random testing', Proceedings 5th Int. Conf. on Software Engineering held in San Diego C.A., pp. 179-83, March 1981

[8] Ferguson R. and Korel B. “The chaining approach for software test data generation”. IEEE Transactions on Software Engineering, 5(1):63-86, January 1996.

[9] Girard, E. and Rault, F. C.: 'A programming technique for software reliability', IEEE Symp. Computer Software Reliability, pp. 44-50, 1973

[10] Goldberg D. E., “Genetic algorithms in search, optimization, and machine learning”, Addison Wesley Longman, Inc., ISBN 0-201- 15767-5, 1989.

[11] Hamlet D. & Taylor R., “Partition testing does not inspire confidence”, IEEE Transactions on Software Engineering, Vol. 16, 1990, pp. 1402-1411.

[12] Holland J., “Adaptation in natural and artificial systems”, University of Michigan Press, Ann Arbor, 1975.

[13] Ince, D. C.: “The automatic generation of test data”, The Computer Journal, Vol. 30, No. 1, pp. 63-69, 1987.

[14] Michalewicz Z., “Genetic Algorithms + Data Structures = Evolution Programs”, Springer-Verlag, 2nd edition, 1994.

[15] Myers, G. J. (1979). Art of Software Testing. John Wiley & Sons.

[16] Offutt J. and Hayes J., “A semantic model of program faults”. In International Symposium on Software Testing and Analysis (ISSTA 96), pages 195{200. ACM Press, 1996.

[17] Taylor R.: 'An example of large scale random testing', Proc. 7th annual Pacific North West Software Quality Conference, Portland, OR, pp. 339-48, 1989.

[18] Tsoukalas M. Z., Duran J. W. and Ntafos S. C.: 'On some reliability estimation problems in random and partition testing', IEEE Transactions on Software Engineering, Vol. 19, No. 7, pp. 687-697, July 1993

[19] Sonia Bhargava, Bright Keswani, “Generic ways to improve SQA by meta-methodology for developing software projects”, International Journal of Engineering Research and Applications, Vol. 3, Issue 4, pp 927-932, July 2013.