Optimization of Software Testing Using Genetic Algorithms

SANJEEV DHAWAN*, KULVINDER S. HANDA*, RAKESH KUMAR**

*Faculty of Computer Engineering, University Institute of Engineering & Technology (U.I.E.T), Kurukshetra University, Kurukshetra (K.U.K)- 136 119, Haryana, INDIA.
**Faculty of Computer Science, Department of Computer Science and Applications (D.C.S.A), Kurukshetra University, Kurukshetra (K.U.K)- 136 119, Haryana, INDIA.
E-mail: rsdhawan@rediffmail.com

Abstract: - This paper presents the study of optimization of software testing techniques by using Genetic Algorithms (GAs) and a sufficient testing convergence condition of GAs is presented. Some new categories of genetic codes are applied in some problem optimizations for the generation of reliable software test cases. These GAs have found their application in detecting errors in the software packages. For example, based on Symmetric Codes theory, new genetic strategy, GA with symmetric code is developed. In the current paper, some key definitions of genetic transformation have been used viz. crossover, mutation and selection. Some of our research shows that genetic encoding techniques have very important influence on the performance of software test cases. This paper is organized into three parts: part I describes the functionality of GAs, part II presents the usage of GAs in software testing to the alternatives of existing software testing techniques, part III discusses the implementation of GAs using MATLAB for the generation of optimized test cases.

Key-Words: - Genetic Algorithms, optimization, software testing, soft computation.

1 Introduction

Genetic Algorithms have been introduced in the sixties by Professor John Holland at university of Michigan as models of an Artificial Evolution [1, 2]. In the thirty past years, they have been successfully applied to a wide range of problems such as Natural Systems Modeling (e.g. Artificial Life environments, immune system modeling [2, 3], Machine Learning systems, and optimization. GAs basically handle a population of chromosomes (individuals) often modeled by vector of binary genes. Each one encodes a potential solution to the considered problem and is named by a so-called fitness value, which is directly correlated to how good it is to solve the problem. In general, there are two basic approaches to test software. The first, more appreciated in the academia, consists of using formal specifications to design an application and then use theorem provers to demonstrate the application's properties. This approach is very strict but unfortunately not often used because the breadth of formal specification methods does not encompass all the functionality needed in today's complex applications. The second approach consists of doing test as part of the traditional engineering models (e.g. waterfall, spiral, prototyping) that have a specific phase for testing generally occurring after the application has been implemented. The modifications to these traditional models have being incorporating testing in every phase of the software development with methodologies such as extreme programming [4] used in the implementation of Windows XP. Despite all the claims, the truth here is that current approaches are insufficient to test software appropriately, thus causing the current status of the field, which clearly seems to be loosing the battle of providing users with reliable software. It has been noticed that complete reliability is hard to achieve in empirical approaches to complete testing is impossible. This does not mean that a good set representing the full space of possible tests cannot be automatically generated thus reducing the cost of software development [5, 6].
Therefore, in the present paper, an attempt has been made to describe the basic nature of existing genetic transformations used for software testing and their related scenarios and applications for generating the efficient software test cases. Keeping in mind the above-mentioned requirement, we have been engaged in a number of activities involving study of software testing, genetic algorithms by using practical and theoretical analysis. Efforts have been made to understand the problem, and develop the corresponding high-level modules in C++ by using

MATLAB version 7.0 tools and libraries provided by the language using the basic parameters of genetic algorithms for the generation of reliable and cost effective test cases.Clearly explain the nature of the problem, previous work, purpose, and contribution of the paper.

2 Process of Genetic Algorithms

Starting with a random initial population, we iteratively apply a so-called Genetic Transformation, which consists of three genetic-like operators such as crossover, mutation and selection having the following specific role: -

a) Selection exponentially spreads, over generations, above average fitted chromosomes by allowing them more offsprings in next generation.
b) Crossover exploits genetic material of the current populations by breeding most interesting strings.
c) Mutation is a process that allows the re-introduction of new genetic material in the population. It introduces diversity in population by flipping randomly chosen genes. As a result, this overall non-linear dynamics leads to a homogeneous final population only featuring instances of the best chromosome found so far [7].

2.1 Selection

There are two important issues in the evolution process of the genetic search: population diversity and selective pressure. Strong selective pressure may force early convergence to a local optimum solution. In contrast, the search is ineffective if the population is too diverse in terms of the desired qualities, and the selective pressure is weak. Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Chromosome with bigger fitness will be selected more times.
This can be simulated by following algorithm: -

Step 1: Calculate the sum of all chromosome fitnesses in population—sum S.
Step 2: Generate random number from interval (0,S)—r.
Step 3: Go through the population and sum fitnesses from 0—sum S. When the sum S is greater then r, stop and return the chromosome where you are. Of course, step 1 is performed only once for each population.

2.2 Crossover

Crossover results in two chromosomes being selected as parents and then crossed to produce two offspring. This results in some of the features of one parent being combined with those of the other. For example:

$\underline{a\ f\ c\ b\ g\ d}$	Parent 1
D E C A F B	Parent 2
a f c A F B	Offspring 1
$\overline{D\ E}\ C\ \underline{b\ g\ d}$	Offspring 2

In this case, the two parents have been split exactly in half, with each child receiving half the parent. The Chromosomes with higher fitness values are more frequently selected for crossover (thus enhancing their chances of passing on their traits to the next generation). This is an example of the survival of the fittest. Crossover acts as an accelerator of the search process performed by the GA. That is, crossover allows a GA to quickly combine beneficial new traits in the population. This means that crossover makes it possible to merge good solutions to generate potentially better ones.

2.3 Mutation

The other evolutionary operator is mutation. Mutation is used to alter one or more elements in the chromosome. For example, given the chromosome:

a b d f a g

Mutation might select the element at position 4 and change it to a Z thus resulting in a new chromosome:

a b d Z a g

The new chromosome generated by either crossover or mutation can then be introduced to the population and evaluated. This can be done immediately (incremental GAs) or en masse when a certain number of new chromosomes have been generated (generational GAs). Whichever approach is used, it is normally the case that an equivalent number of chromosomes are deleted from the population. Mutation helps to ensure diversity in the population. In terms of the search space, it helps the GA to jump to other parts of the space.

3 Algorithmic Approach Used for Genetic Algorithms

The basis of all GAs is the algorithm illustrated in the following steps: -

a) initialize a set of genes.
b) evaluate genes in population.

c) until the maximum number of generations. have been created.
d) Crossover / mate genes in population.
e) mutate genes in new population.
f) evaluate genes in extended population.
g) select genes to populate next generation.

The algorithm uses point crossover as depicted in Figure 1. Point crossover works by selecting a break point in the chromosome of the two selected individuals and recombining them using each-others' half [8, 9].

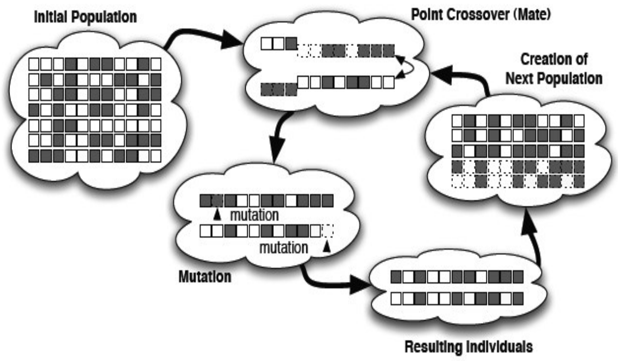

Fig. 1. Information flow for the GA steps.

4 Test Case Generation using Genetic Algorithms

For the test case generation, the following figure 2 has been designed, which consists of program P1 having multiple input and one output variables. X is the set of all input variables and Y is the set of all output variables [5, 6, 10].

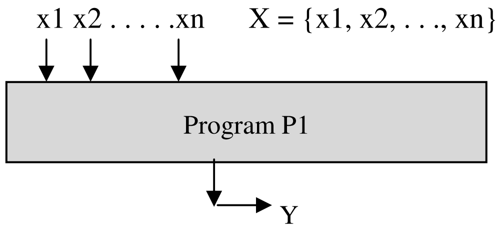

Fig. 2. Shows the program P1 with multiple inputs and outputs.

Test data for set X can be defined in terms of preconditions that describe valid and invalid data values for each input variable x. These preconditions may be determined from several sources, including the program's specification and the constraints of the computing environment. To create a test set, it is common to apply black-box test data selection criteria (such as equivalence-class partitioning, boundary value analysis, etc.) to each input variable of X with respect to the preconditions. After applying test selection criteria to each variable of X, we will have a set of test data values for each of the input variables:

$$D(x1), D(x2), \ldots, D(xn) \qquad \textbf{(1)}$$

Since program P1 has multiple input variables, we must now consider how to test combinations of program inputs. The most thorough approach is to test every possible combination of the selected test data values using GAs. The fitness functions based on other factors may be used to generate a sequence. In fact, in a more elaborate use of GAs, the data itself can be updated in a feedback loop based on the result of the execution of the test plan. Another point that needs to be noted is that the fitness function as defined earlier has a strong dependency to the application one needs to test. For instance, it may be more meaningful to use a function based on the standard deviation of the inconsistency so that test plans that do not fall within the norm are given a higher score and hence tested first. Our approach for the generation of optimized test plans is quite different from other nature-inspired strategies such as GA and Neural Networks [11, 12, 13].

5 Results

The algorithms presented in this paper have been implemented on MATLAB version 7.0 for the generation of optimized test cases using GAs. These algorithms have been tested extensively with different test inputs containing many special cases, including random testing and testing based on specification of software. The figure 3.1 shows the fitness value for the different number of input variables, figure 3.2 shows the optimized fitness value for the generation of reliable test cases., and the figure 3.3 shows the best, worst and mean test scores for the generation.

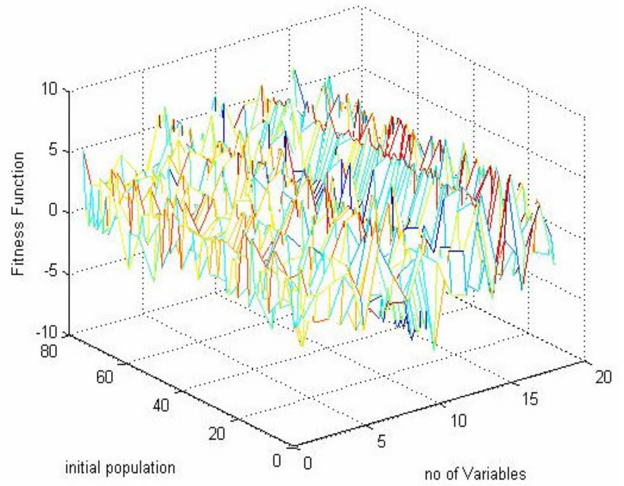

Fig. 3.1. Shows the fitness value vs. number of variables.

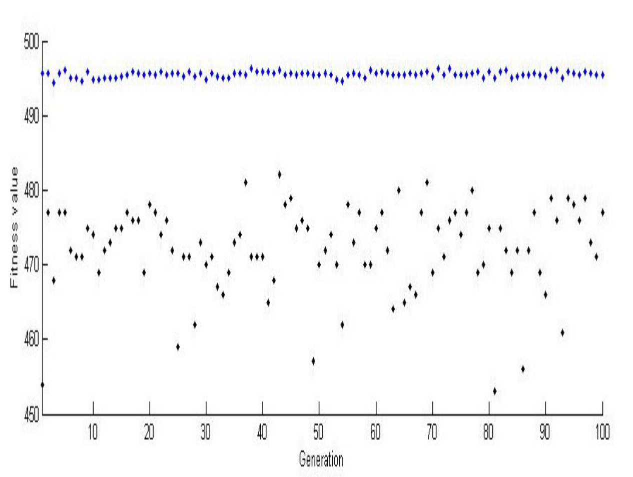

Fig. 3.2. Shows the optimized output of fitness value vs. generation.

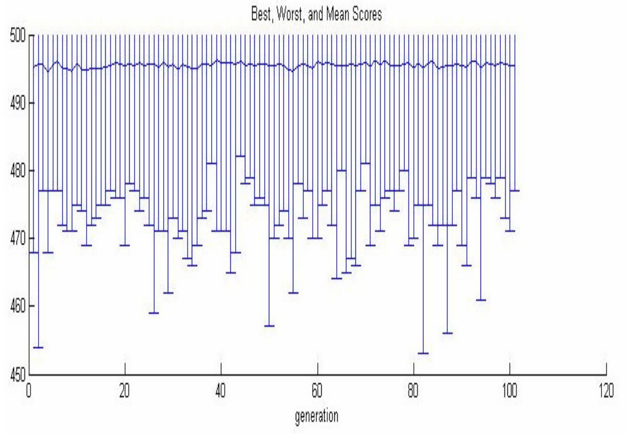

Fig. 3.3. Shows the best, worst and mean test scores vs. generation.

6 Perspectives and Further Work

Genetic encoding techniques have significant influence on GAS' performance in solving some problems with big algorithm complexity. Our study shows that, in some problems some special codes must be taken, or else the algorithms may not be able to get convergence or the solution got is poor. The simulation shows that the proposed GAS with the special codes can find solutions with better quality in shorter time than some classical GAs. This paper has presented the optimised behaviour for the generation of software test cases using Genetic Algorithms as a new efficient class. However, the generation of Genetic Algorithms based on real meta-level evolution of a suitable metrics over the search space is yet to be determined. From an empirical point of view, dynamical environments have revealed to be of a great help in characterizing metrics properties. Such metrics will explain the effectiveness of GAs in dynamic scenarios, which will open the new field for genetic encoding techniques. Further work will consist in studying in more details how can such an optimal meta-mask be evolved and what properties it must feature thus enlightening us on GAs dynamics.

Acknowledgements

A major part of the research reported in this paper is carried out at U.I.E.T, and D.C.S.A, K.U.K, Haryana, India. We are highly indebted and credited by gracious help from the Ernet section of K.U.K for their constant support and help while testing our proposed models on to different systems. The authors would like to thank those nameless individuals who worked hard to supply the data.

References:

[1] D.E. Goldberg, Genetic Learning in optimization, search and machine learning. Addisson Wesley, 1994.

[2] J.J. Grefenstette. Genetic algorithms for changing environments. In R. Manner abd B. Manderick, editor, Parallel Problem Solving from Nature 2, pages 465-501. Elsevier Science Publishers B.V., 1992.

[3] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis and implications. In Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy, 1996.

[4] C. E. Williams. Software testing and uml. In Proceedings of the 10th International Symposium on Software Reliability Engineering, Boca Raton, Florida, Nov. 1999. IEEE Press.

[5] Watkins, A., The automatic generation of test data using genetic algorithms, Proceedings of the 4th Software Quality Conference, vol. 2, 1995, pp. 300-309.

[6] J. Wegener, K. Buhr, H. Pohlheim, Automatic Test Data Generation For Structural Testing of Embedded Software Systems by Evolutionary Testing. GECCO, 2002, pp. 1233-1240.

[7] John Hunt, Testing Control Software using a Genetic Algorithm**,** Engg Appl. Artif. lntell. Vol. 8, No. 6, pp. 671-680, 1995, Elsevier Science Ltd.

[8] F. Vavak and T.C. Fogarty. A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. In Proceedings of the Society for the Study of Artificial Intelligence and Simulation of Behavior, workshop on

Evolutionnary Computation’96, pages 301-307. University of Sussex, 1996.

[9] J. J. Grefenstette, et al, "Genetic algorithm for the Traveling salesman problem", in Proc. Int. Conf. Genetic Algorithms and Their Applications, July 1985, pp. 160-168.

[10] Kenneth, D. Jong, et al, "Using Genetic Algorithms to solve NP-complete Problems", Intl. Conf 3rd. Genetic Algorithms and Their Applications. 1989, pp. 124-132.

[11] D. Berndt, J. Fisher, L. Johnson, J. Pinglikar, and A. Watkins. Breeding software test cases with genetic algorithms. In HICSS ’03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS’03), pages 338-347, Washington, DC, USA, 2003. IEEE Computer Society.

[12] D. J. Berndt. Investigating the performance of genetic algorithms-based software test case generation. In Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE’04), 2004.

[13] C. C. Michael, G. E. McGraw, M. A. Schatz, and C. C. Walton. Genetic algorithms for dynamic test data generation. In ASE ’97: Proceedings of the 1997 International Conference on Automated Software Engineering (ASE ’97) (formerly: KBSE), pages 307-308, Washington, DC, USA, 1997. IEEE Computer Society.