International Journal of Innovation, Management and Technology, Vol. 1, No. 1, April 2010
ISSN: 2010-0248

Optimization of Functional Testing using Genetic Algorithms

Kulvinder Singh and Rakesh Kumar

***Abstract*— Software system should be reliable and available failing which huge losses may incur. To achieve these objectives a thorough testing is required. Adequacy of test cases is the key to the success. This paper presents the study of optimization of software testing techniques by using Genetic Algorithms (GAs) and specification based testing. Some new categories of genetic codes are applied in some problem optimizations for the generation of reliable software test cases based on the specification of the software. These GAs have found their application in detecting errors in the software packages. Based on new genetic strategy and GAs 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 techniques have very important influence on the performance of software test cases.**

***Index Terms*— Genetic Algorithms, optimization, specification based software testing, soft computation**

I. 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 handle a population of individual (chromosomes) often modeled by vector of binary genes. Each one encodes a potential solution to the problem and so-called fitness value, which is directly correlated to how good it is to solve the problem. In general, the basic approaches are to test software consists of using formal specifications to design an application. 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 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 has 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. This paper is organized into three parts: part I describes the functionality of GAs, part II presents the usage of GAs in specification based 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.

II. Approach used for Genetic Algorithms

GA is a search technique used to find exact or approximate solutions to optimization and search problems. GAs represents a class of adaptive search techniques & procedures based on the processes of natural genetics & Darwin's principal of the survival of the fittest. There is a randomized exchange of structured information among a population of artificial chromosomes. When gas are used to solve optimizations problems, good results are obtained surprisingly quickly. A problem is defined as maximization of a function of the kind $f(x_1, x_2, \ldots x_m)$ where $(x_1, x_2, \ldots, x_m)$ are variables which have to be adjusted towards a global optimum. Three basic operators responsible for GA are (a) selection, (b) crossover & (c) mutation. Crossover performs recombination of different solutions to ensure that the genetic

Manuscript revised feburary2, 2010.

Kulvinder Singh is with Department of Computer Science and Engineering, University Institute of Engineering & Technology (U.I.E.T), Kurukshetra University, Kurukshetra (K.U.K), India- 136 119 (Phone: +91-94162-24353, E-mail: kshanda@rediffmail.com).

Rakesh Kumar is with the Department of Computer Science and Applications (D.C.S.A), Kurukshetra University, Kurukshetra (K.U.K), India- 136 119(Phone: +91-98963-36145; e-mail: rsagwal@rediffmail.com)

information of a child life is made up of the genes from each parent.

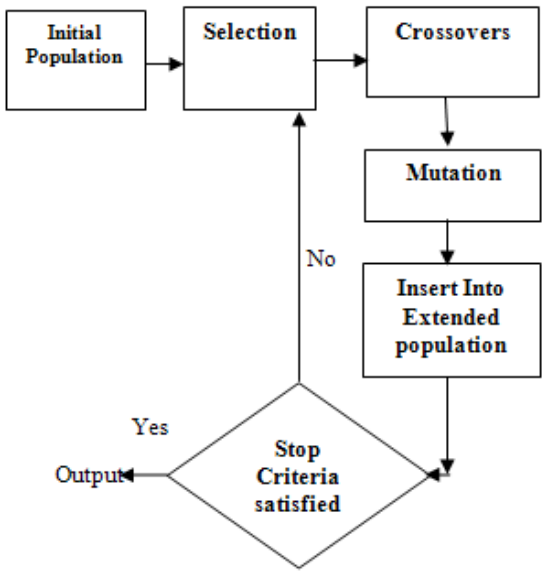

Figure 1: information flow for the GA steps

GAs differentiated from other conventional techniques due to: (i) GA a representation for the sample population must be derived; (ii) GAs manipulates directly the encoded representation of variables, rather than manipulation of the variables themselves; (iii) GAs use stochastic rather then deterministic operators; (iv) GAs search blindly by sampling & ignoring all information except the outcome of the sample; (v)GAs search from a population of points rather than from a single point; thus reducing the probability of being stuck at a local optimum, which make them suitable for parallel processing. In the context of software testing, the basic idea is to search the domain for input variables, which satisfy the goal of testing.

III. Test Case Generation using Genetic Algorithms

In this, implementation of genetic algorithm using MATLAB software has been dealt. Given the versatility of MATLAB's high-level language, problems can be coded in m-files in a fraction of the time that it would take to create c or FORTRAN programs for the same purpose. Couple this with MATLAB's advanced data analysis, visualization tools and special purposes application domain toolbox and the user is presented with a uniform environment with which to explore the potential of genetic algorithms. The genetic algorithm GUI toolbox plays a major role for obtaining optimized solution and best fitness value. GA's also require that a number of parameters be set. The first one is the *population size*. It has an impact on the speed of the GA convergence towards a near optimal solution and its capability to avoid local optima. Larger population sizes increase the amount of variation present in the initial population at the expense of requiring more cost function evaluations and longer execution times. Typical population sizes in the literature range between 25 and 100. As a heuristic, having a population size of two or three times the number of classes should be sufficient. Mutation prevents the GA search to fall into local optima, but they should not happen too often or the search will converge towards a random search. The mutation rate is defined as the probability for a chromosome to undergo a mutation. 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 as shown in fig 2.

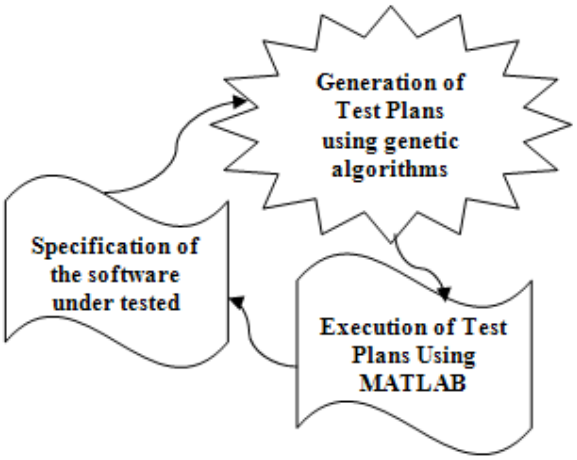

Figure2. Shows the generation of test cases

With the above defined, GA is defined as follows:

Procedure GA(φ, θ, n, r, m)
// φ is the fitness function for ranking individuals
// θ is the fitness threshold, which is used to determine when to halt
// n is the population size in each generation (e.g., 100)
// r is the fraction of the population generated by crossover (e.g., 0.6)
// m is the mutation rate (e.g., 0.001)
P: = generate n individuals at random
// initial generation is generated randomly
while max (φ (h_i)) $< \theta$ **do**
//define the next generation S (also of size n)
Reproduction step: Probabilistically select (1-r) n individuals of P and add them to S, where the probability of selecting individual h_i is
Prob(h_i)= φ (h_i) / $\sum(\varphi(h_j))$
Crossover step: Probabilistically select r*n/2 pairs of individuals from P according to Prob (h_i)
For each pair (h_1, h_2), produce two offspring by applying the crossover operator and add these offspring to S
Mutate step: Choose m% of S and randomly invert one bit in each
P: = S
End while
Find b such that φ (b) = max (φ (h_i))
Return (b)
End proc

IV. Consideration for Test Case Generation

In order to make the experiment realistic, an attempt was made to choose an application that would normally be a candidate for the inclusion of fault tolerance. The problem that was selected for programming is a simple and realistic data structure largest number system. Program read some data that represents as test cases an array (integer or float). To check the efficiency of system and the completeness of the

test set, tests are performed by introducing the mutant in the software. This program was originally written in MATLAB, and the program has been subjected to several thousands test cases.

Assumptions made are as under:

A. Encoding

Direct value encoding can be used in problems where some more complicated values such as real numbers are used. In the value encoding, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as (real) numbers, chars or any objects.

B. Selection

From a population of individuals, we wish to give the fitter individuals a better chance to survive to the next generation. We do not want to use the simple criterion "keep the best n individuals." It turns out nature does not kill all the unfit genes. They usually become recessive for a long period. Then they may mutate to something useful. Therefore, there is a tradeoff for better individuals and diversity. The individuals are selected according to Rank selection criteria. Rank selection ranks the population first and then every chromosome receives fitness value determined by this ranking. The worst will have the fitness 1, the second worst 2 etc. and the best will have fitness N (number of chromosomes in population).

C. Crossover

Two-point crossover - two crossover points are selected, binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again

D. Mutation

Randomly change one or more digits in the string representing an individual.

V. Results

For the test case generation, the following figure 3 has been designed, which consists of program P1 (as black box) having multiple input and one output variables.

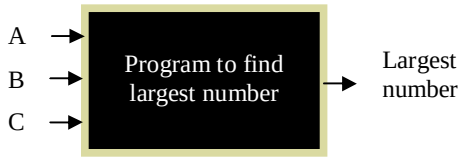

Figure 3: Illustrates the program P1 to find largest number.

Test data for inputs can be defined in terms of preconditions that describe valid and invalid data values for each input. 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 applies to black-box test data selection criteria (such as equivalence-class partitioning) to each input variable with respect to the preconditions. After applying test selection criteria to each variable, we will have a set of test data values for each of the input variables. 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. In fact, in a more elaborate use of GAs, the data itself updated in a feedback loop based on the result of the execution of the test plan.

The program has been tested extensively with different test inputs containing many special cases, including random testing and testing based on specification of software.

Figure 4 depicts the distance between individual, as the numbers of generations increases, the distance between the individual are decreases to zero.

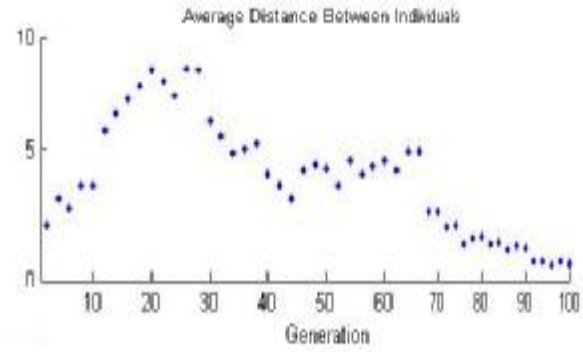

Figure 4: depict the Average distance between individuals.

The figure 5 shows the results generated by the MATLAB for the software under test after inserting mutation in the software. The value differs from the specification (expected value) of the software is depict as error.

Figure 6 shows the graph, which describes the errors detected in the software under test

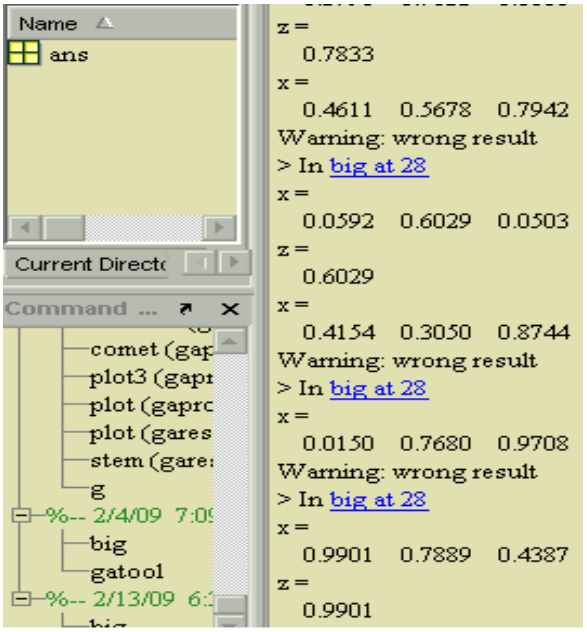

Figure 5 shows the output generated by MATLAB

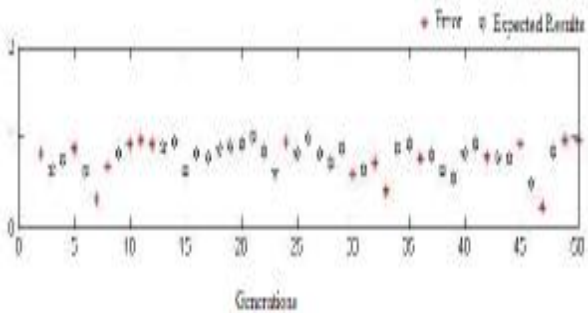

Figure 6: highlights the error detected vs. generation

These results are the focus for the software i.e. the software is mainly designed to process the optimization of test cases using GAs.

VI. Conclusion

Genetic Algorithms are easy to apply to a wide range of optimization problems, like the traveling salesperson problem, inductive concept learning, scheduling, and layout problems. Software testing is also an optimization problem with the objective that the efforts consumed should be minimized and the number of faults detected should be maximized. Software testing is considered most effort consuming activity in the software development. Although a number of testing techniques and adequacy criteria have been suggested in the literature but it has been observed that no technique/criteria is sufficient enough to ensure the delivery of fault free software consequential to the need of automatic test case generation to minimize the cost of testing. The simulation shows that the proposed GAs with the specification can find solutions with better quality in shorter time. The developer uses this information to search, locate, and segregate the faults that caused the failures. While each of these areas for future consideration could be further investigated with respect to applicability for software testing, as demonstrated by the examples of this paper, the simple genetic algorithm approach presented in this paper provides in itself a useful contribution to the selection of test cases and a focused examination of test results.

Acknowledgment

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.

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 .

[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] A. Watkins, 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 ofArtificial 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] A. 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.