

Software Testing using Genetic Algorithm

¹Sanjay Kumar Sonkar, ²Dr. Anil Kumar Malviya, ³Dharmendra Lal Gupta, ⁴Ganesh Chandra

^{1,2,3,4}Dept. of CSE, KNIT, Sultanpur, UP, India

Abstract

Testing is a process used to identify the correctness, completeness and quality of developed computer software. Testing, apart from finding errors, is also used to test performance, safety, fault-tolerance or security. Testing is the most important quality assurance measure for software. Testing is time consuming and laborious process. Therefore, techniques to automatic test data generation would be useful to reduce the cost and time. Software testing is an important and valuable part of the software development life cycle. Due to time, cost and other circumstances, exhaustive testing is not feasible that's why there is a need to automate the software testing process. Testing effectiveness can be achieved by the State Transition Testing (STT) which is commonly used in real time, embedded and web based type of software systems. The objective of this paper is to present an algorithm by applying a Genetic Algorithm Technique, for generation of optimal and minimal test sequences for behaviour specification of software. Present paper approach generates test sequence in order to obtain the complete software coverage.

Keywords

Genetic Algorithms, Software Testing, Test Case Generation

I. Introduction

Testing is one of the most used software quality assessment methods. There are two important processes when testing object-oriented software are used. First, the software has to be initialized with a set of values. These values are used to set a number of variables that are relevant for the test case [3, 10]. The values of these variables define a single state from the possible set of states, the software can be determined. These values can either be a primitive value such as an integer or complex values such as an object. With the software testing initialized [9], its method takes one or more software specification, defines the output of the software and what is a valid input. Since the number of more objects as parameters, these objects also have to be initialized. To determine if the test case passed or fail, a software specification [8] has to be used. The number of possible states software may have is exponential [12], it is impossible to test all of them. Software testing [9, 12, 19], is one of the major and primary techniques for achieving high quality software. Software testing is done to detect presence of faults [6, 8], which cause software failure. However, software testing is a time consuming and expensive task.

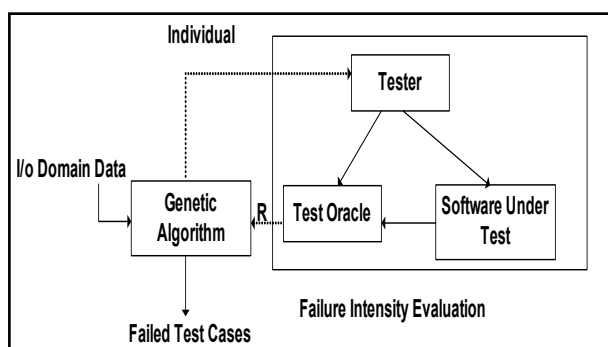


Fig. 1: GA Approach to Focused Software Usage Testing

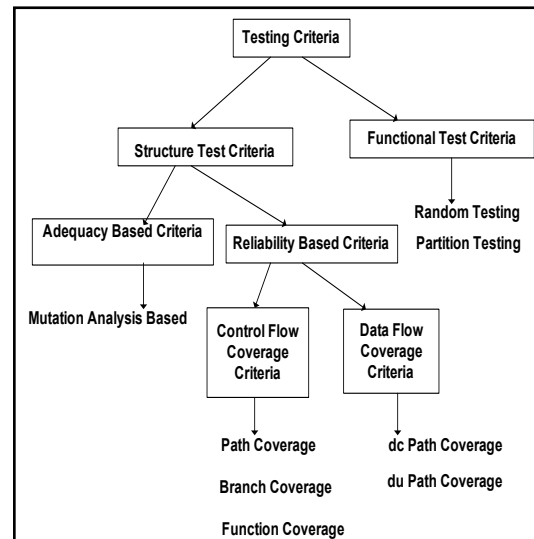


Fig. 2: General Classification Framework for Testing Criteria

This paper is design as follows Section I, Introduction, Section II, Key Research Concepts, Section III, Discuss the Conclusions. Finally, Section IV, Future Work of the paper.

II. Key Research Concepts

Our technique is based on the use of 'Genetic Algorithm' and 'Mutation Analysis'. These two concepts are the main focus of this paper.

A. Genetic Algorithms

Genetic Algorithms (GA) [4-5, 14], are search algorithms based on the natural selection and genetic reproduction as described by Charles Darwin. They are used to find solutions to Optimisation and search problems [12]. Genetic algorithms became popular when John Holland published the "Adaptation in Natural and Artificial Systems", in 1975 and De Jong finished an analysis of the behaviour of a class of genetic adaptive systems in the same year. The basic idea of a GA is to encode the values of the parameters of an Optimisation problem in a chromosome [13, 22] which is evaluated by an objective function. GAs have been successfully applied to a wide range of applications, including Optimisation, scheduling, and design problems. GA has been applied in many Optimisation problems for generation test plants [10-11] for functionality testing, feasible test cases and in many other areas. GA has also been used in model based test case generation.

As shown in fig. 3, the algorithm starts by initializing or randomly generating a set of chromosomes (population). At the end of each generation, each chromosome is evaluated and modified according to a number of generation operations in order to produce a new population [1, 4]. This process repeats until a predefined number of generations are computed.

Using genetic algorithms in test data generation for software testing is the process of identifying [10], a set of program input data, which satisfies a given testing criterion [10-11]. In translating the concepts of genetic algorithms to the problem of test-data [12] generation we perform the following tasks:

- First of all we consider the population to be a set of test data.

- Find the set of test data that represents the initial population. This set is randomly generated according to the format and type of data used by the program under test.
- Determining the fitness of each individual which is based on a fitness function that is problem-dependent.
- Select two individuals that will be mated to contribute to the next generation.
- Apply the crossover and mutation processes.

The above algorithm will iterate until the population has evolved to form a solution to the problem (satisfies a given testing criterion), or until a termination condition is satisfied.

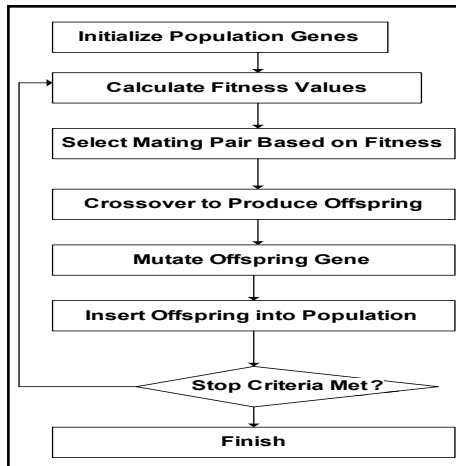


Fig. 3: Genetic Algorithm Flow Diagram

Basic Steps of a Typical Genetic Algorithm [5, 7]
Procedure GA

Initialize population;

While termination condition not satisfied do

```

{
  Evaluate current population;
  Crossover;
  Mutation;
  Set current population equal to be the new child population;
}
  
```

These primary operations include:

(1). Selection or Reproduction

A selection scheme is applied to determine how individuals are chosen for mating method based on their fitness [13]. Fitness can be defined as a capability of an individual to survive and reproduce in an environment. Selection generates the new population from the old one, thus starting a new generation.

This operation assigns the reproduction probability to each individual based on the output of the fitness function. The individual based on the output of the fitness function. The individual with a higher ranking is given a greater probability for reproduction. As a result, the fitter individuals are allowed a better survival chance from one generation to the next.

Some of the selection schemes include:

(a). Rank Schema

Selects the best individuals of the population every time.

(b). Roulette Wheel

Selects individuals according to their fitness values as compared to the population. The probability [20] of an individual being picked is:

$$p_i = \frac{\text{fitness}_i}{\sum_{i=0}^{\text{len}(\text{population})} \text{fitness}_i}$$

(c). Tournament Sampling

Uses the roulette wheel method to select two individuals. Then it picks the one with the higher fitness value.

(d). Uniform Sampling

Selects an individual randomly from the population.

(i). Stochastic Remainder Sampling

First computes the probability of each individual being selected, p_i and its expected representation, $E = p_i * \text{len}(\text{population})$. The expected representation is used to create a new population of the same size. For example, if an individual has E equal to 1.7, it will fill one position in the new population and it has a probability of 0.7 to fill another position. After the new population is created, the uniform method is used to select the individuals for mating.

(ii). Deterministic Sampling

Computes E of each individual as in the stochastic remainder sampling [11]. A new population is created and filled with all individuals with $E \geq 1$ and the remainder positions are filled by sorting the original population's fractional parts of E and selecting the highest individuals on the list.

(2). Crossover or Recombination

Crossover is process where two or more chromosomes are combined to form one or more chromosomes. The idea behind crossover is that the offspring may be better than both parents. Crossover is normally done between two individuals, but more can be used.

There are many crossover algorithms [6]; some important algorithms are as described below:

- Uniform crossover will randomly select the parent where each gene should come from.
- Even odd crossover will select the genes with even index from parents A and the genes with odd index from parent B.
- One point crossover will randomly select a position on the chromosome and all the genes to the left come from parent A and the genes to the right come from parent B.
- Two points crossover will randomly select two positions and pick the genes from parent A which have a greater index than the smaller position and a smaller index than the biggest position. The remaining genes come from parent B.
- Partial match crossover will produce two children C1 and C2. It initializes C1 by copying the chromosome of the parents A and C2 by copying the chromosome of parent B. It will then randomly select a number of positions and swap the genes between C1 and C2 at those positions.
- Order crossover produces two children C1 and C2. It initializes by copying the genes of the parents to children and deleting n genes randomly selected from each offspring. It then selects an interval with size n slides the genes such that the interval is empty. It then selects the original genes in that interval from the opposite offspring [4].
- Cycle crossover produces two children C1 and C2. It initializes C1 and C2 by copying the chromosomes of the parents A and B respectively. Then it selects n random positions and replaces the genes from C1 with genes from parent B in those positions. The process is repeated for C2 with parent A.

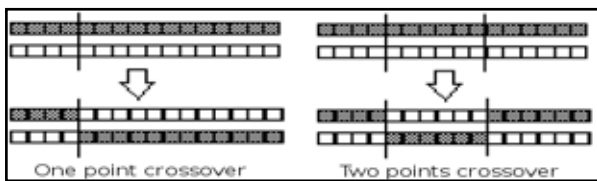


Fig. 4: One and Two Point's Crossover [6]

This operation is used to produce the descendants that make up the next generation. This operation involves the following crossbreeding procedures:

- Randomly select two individuals as a couple from the parent generation.
- Randomly select a position of the genes, corresponding to this couple, as the crossover point. Thus, each gene is divided into two parts.
- Exchange the first parts of both gene corresponding to the couple. Add the two resulted individuals to the next generation.

Step1	c ₁	A E B C ..	D H O J	K . F N .
	c ₂	F . A N K .	L M I G	. E B C .
Step2	c ₁	H O J K F N		A E B C D
	c ₂	M I G E B C		F A N K L
Step3	c ₁	H O J K F N	L M I G	A E B C D
	c ₂	M I G E B C	D H O J	F A N K L

Fig. 5: Order Crossover Examples [4]

B. Mutation

Mutation testing [24-25, 28] is a fault based testing technique used to find the effectiveness of test cases. It is a powerful and computationally expensive technique to find the adequacy of test cases [22, 26]. High quality software cannot be done without high quality testing.

Mutation testing measures how “good” our tests are by inserting faults into the program under test. Each fault generates a new program, a mutant, which is slightly different from the original. The idea is that the tests are adequate if they detect all mutants [16-17].

This operation picks a gene at random and changing its state according to the mutation probability [12]. The purpose of the mutation operation is to maintain the diversity in generation to prevent premature convergence to a local optimal solution. The mutation probability is given intuitively since there is no define way to determine the mutation probability.

Three basic operations are described below:

- Flip mutator [18], will change a single gene of the chromosome to a random value according to the range specified by the alleles (alleles is an alternative form of a gene that is located at a specific position on a specific chromosome).
- Swap mutator will randomly swap a number of genes of the chromosome.
- Gaussian mutator will pick a new value around the current value using a Gaussian distribution [3].

The mutation operation is defined according to the structure of the chromosome. When the chromosome is stored in a tree, one possible mutation is to swap subtrees as shown in fig. 6.

1. Purposes

- They are provided a basis on which to decide whether something should be mutated.
- They help in understanding and critiquing an existing set of mutation operators.

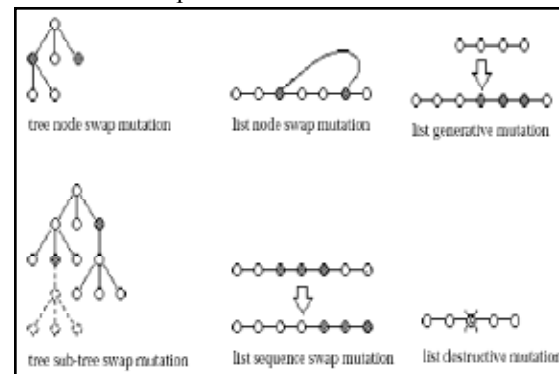


Fig. 6: Examples of mutation algorithms [3]

The main characteristics of GAs [2], are listed below:

- Concentrates on chromosomes with above average fitness.
- Exploits information about a large number of values while processing a small population.
- Prevents search from stagnating at a local optimum.
- They take advantage of old knowledge held in a population of solutions to generate new solutions with improved performance.

III. Conclusion

This paper introduces a genetic algorithm approach to software usage testing that is used to explore the space of input data and identify and focus on regions that cause failure. Analysis of the examples in this paper demonstrates that genetic algorithms can be used as a tool to help a software tester search, locate, and isolate failures in a software system testing. The use of genetic algorithms supports automated testing and helps to identify those failures that are most severe and likely to occur for the user.

IV. Future Work

Despite of the big improvement achieved by evolutionary testing [2], there are many improvements that should increase this improvement even further. Some possible improvements [6] and research directions are presented below:

A. Efficiency

The efficiency of the algorithm can be improved by combining the genetic algorithm and Auto test into a single system. At the moment, every time Auto test is invoked from the genetic algorithm, it has to load and parse the class under test.

B. Reusing Strategies

The strategies evolved by the genetic algorithm may be reused when evolving a new strategy [7] for a new class.

C. Static Analysis

This system uses a vary naive static analysis technique; a more advanced technique may be able to improve the adaptation of the evolutionary strategy.

D. Code Metrics

At the moment the information about the complexity of the methods being tested is not taken into account. When initializing the testing strategies, code metrics [22] could be used to have a smarter initialization.

E. Evolution of Strategies

By evolving a testing strategy for a longer time may increase the number of faults found, specially for classes [5, 9] with a low number of faults.

There are other problems as flag and enumeration variables and unstructured control flow. Additional researches are required to overcome these problems.

One of the major difficulties [10, 13] in software testing is the automatic generation of test data that satisfy a given adequacy criterion. To solve this difficult problem there were a lot of research works, which have been done in the past. Perhaps the most commonly encountered are random test-data generation, symbolic (or path-oriented) test-data generation based on genetic algorithm [12, 14].

References

- [1] Goldberg, D.E., "Genetic Algorithms in Search of Optimisation", 1989.
- [2] B. Korel, "Dynamic method for software test data generation," *Software Testing, Verification & Reliability*, vol. 2, 1992, pp. 203-213.
- [3] Alan C. Schultz, John J. Grefenstette, and Kenneth A. De Jong, "Test And Evaluation by Genetic Algorithms", IEEE, 1993.
- [4] Jefferson Offutt, W. Michael Craft, "Using Compiler Optimisation Techniques to Detect Equivalent Mutants", *The Journal of Software Testing, Verification and Reliability*, 4(3), pp. 131-154, 1994.
- [5] Painton, L., Campbell, J., "Genetic algorithms in Optimisation of system reliability", *IEEE Trans. Reliab.* 44 (2), pp. 172-178, 1995.
- [6] Michalewicz, Z., "Genetic Algorithms, Data Structures Evolution Programs", Verlag, Heidelberg, Berlin, Third Revised and Extended Edition, 1999.
- [7] Tian, J., "Measurement and continuous improvement of software reliability throughout software life-cycle", *J. Syst. Software* 47, 1999.
- [8] Roy P Pargas, Mary Jean Harrold, Robert R Peck, "Test Data Generation Using Genetic Algorithms", *Journal of Software Testing, Verification and Reliability*, 1999.
- [9] W. Caarls, "Genetic algorithm visualisation. Master's thesis, Faculty of Science", University of Amsterdam, The Netherlands, September 2002.
- [10] S. Yang, "Statistics-based adaptive non-uniform crossover for crossover for genetic algorithms", In J. Bullinaria, editor, *Proceedings of the 2002 UK Workshop on Computational Intelligence (UKCI-02)*, pp. 201-208, 2002.
- [11] P. McMinn, "Search-Based Software Test Data Generation: A Survey", *Software Testing, Verification and Reliability*, vol. 14, No. 2, pp. 105-156, 2004.
- [12] Phil. McMinn, "Search-Based Software Test Data Generation: A Survey", *Software Testing, Verification and Reliability*, Vol. 14, No. 3, pp. 212-223, 2004.
- [13] M. R. Girgis, "Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm", *Journal of Universal Computer Science*, vol. 11, No. 6, pp. 898-915, 2005.
- [14] B. Antonia, "Software Testing Research: Achievements, Challenges, Dreams", in *2007 Future of Software Engineering: IEEE Computer Society*, 2007.
- [15] S. Bandyopadhyay, S. K. Pal, "Classification and Learning Using Genetic Algorithms", *Application in Bioinformatics and Web Intelligence*. Berlin, Germany: Springer, 2007.
- [16] M. Kay, "XSL Transformations (XSLT) version 2.0", W3C, W3C Recommendation, (2007), [Online] Available: <http://www.w3.org/TR/2007/REC-xslt20-20070123/>.
- [17] Kim S., Clark J.A., Mcdermid J.A., "Class mutation: mutation testing for object-oriented programs", *Proc. Net. Object Days*, October 2000, [Online] Available: <http://wwwusers.cs.york.ac.uk/~jac/papers/ClassMutation.pdf>, accessed, March 2008.
- [18] H. Shahriar, "Mutation-based testing of buffer overflows, SQL injections, and format string bugs", Ph.D. dissertation, University of Queen, Aug. 2008.
- [19] H. Shahriar, M. Zulkernine, "MUSIC: mutation-based SQL injection vulnerability checking", in *Proceedings of the 8th International Conference on Quality Software (QSIC '08)*, Aug. 2008, pp. 77-86.
- [20] Yong Chen, Yong Zhong, Tingting Shi, Jingyong Liu, "Comparison of Two Fitness Functions for GA-based Path-Oriented Test Data Generation", *2009 Fifth International Conference on Natural Computation*, IEEE, 2009.
- [21] Y. S. Ma, Y. R. Kwon, S. W. Kim, "Statistical investigation on class mutation operators", *ETRI Journal*, Vol. 31, No. 2, pp. 140-150, 2009.
- [22] M. Singh, S. Mishra, "Mutant generation for aspect oriented programs," *Indian Journal of Computer Science and Engineering*, Vol. 1, No. 4, pp. 409-415, 2010.
- [23] U. Praphamontripong, J. Offutt, "Applying mutation testing to Web applications", in *Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops (Mutation 2010)*. IEEE Computer Society, Apr. 2010, pp. 132-141.
- [24] "Quantitative evaluation of mutation operators for WSBPEL compositions", in *Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops (Mutation 2010)*. IEEE Computer Society, Apr. 2010, pp. 142-150.
- [25] Y. Jia, M. Harman, "An Analysis and Survey of the Development of Mutation Testing", *IEEE Transactions of Software Engineering*, Vol. To appear, 2010.
- [26] H. Zhang, "Mutation operators for C++", Feb. 2010. [Online] Available: <http://www.people.cis.ksu.edu/~hzh8888/mseproject.htm>.
- [27] L. Madeyski, N. Radyk, "Judy - A mutation testing tool for Java," *IET Software*, Vol. 4, No. 1, pp. 32-42, 2010.
- [28] F. Lonetti, E. Marchetti, "X-MuT: a tool for the generation of XSLT mutants", in *Proceedings of the Seventh International Conference on the Quality of Information and Communications Technology*. IEEE Computer Society, 2010, pp. 280-285.



Sanjay Kumar Sonkar was born at Varanasi, (U.P), in India. He received his B. Tech degree in Computer Science & Engineering in 2010 from Radha Govind Engineering College, Meerut, India. Presently, he is an M. Tech Student in Computer Science & Engineering from Kamla Nehru Institute of Technology, (KNIT), Sultanpur, U.P., India.



Dharmendra Lal Gupta is currently working as an Assistant Professor in the Department of Computer Science & Engineering at KNIT, Sultanpur (U.P.) India. And he is also pursuing his Ph.D. in Computer Science & Engineering from Mewar University, Chittorgarh (Rajasthan). He received B. Tech. (1999) from Kamla Nehru Institute of Technology (KNIT) Sultanpur, in Computer Science & Engineering, M.Tech. Hon's (2003) in Digital Electronics and Systems from Kamla Nehru Institute of Technology (KNIT) Sultanpur. His research interests are Cryptography and Network Security, Software Quality Engineering, and Software Engineering.



Dr. Anil Kumar Malviya is an Associate Professor in the Computer Science & Engineering Department at Kamla Nehru Institute of Technology, (KNIT), Sultanpur. He received his B.Sc. & M.Sc. both in Computer Science from Banaras Hindu University, Varanasi respectively in 1991 and 1993 and Ph.D. degree in Computer Science from Dr. B.R. Ambedkar University; Agra in 2006. He is Life Member of CSI, India. He

has published about 25 papers in International/ National Journals, conferences and seminars. His research interests are Data mining, Software Engineering, Cryptography & Network Security.



Ganesh Chandra was born at Kanpur, India. He received the B. Tech. Degree in Computer Science and Engineering in 2009 from Dr. Ambedkar Institute of Technology for Handicapped Kanpur, India. He is currently pursuing M. Tech in Computer Science and Engineering from Kamla Nehru Institute of Technology, Sultanpur, U.P, India.