Online ISSN: 3107 - 7676

IJMR 2025; 1(6): 14-17

2025 November - December

www.allmultiresearchjournal.com

Received: 20-09-2025

Accepted: 22-10-2025

Published: 08-11-2025

DOI: https://doi.org/10.54660/IJMR.2025.1.6.14-17

A Review of a Randomized Approach to Test Case Generation Using Genetic Algorithms

Sumit Saxena [1]*, Dr. Qaim Mehdi Rizbi [2]

[1-2] Department of Computer Science, Shri Krishna University Chhatarpur, Madhy Pradaesh, India

Corresponding Author; **Sumit Saxena**

Abstract

The quality of software relies on testing in accordance with user specifications and requirements. Designing, prioritizing, and optimizing test cases to ensure quality presents significant challenges. Various testing tools are available for software testing, applicable in both manual and automated contexts. Recent studies have demonstrated that automated software testing outperforms manual testing when employing heuristic search methods. This paper presents a survey on the genetic algorithm approach for the random generation of test cases in functional software testing.

Keyword: Designing, Prioritizing, and Optimizing Test

Introduction

Software testing is a process that evaluates the runtime quality and performance of software to ensure it meets the highest standards of quality. The software industry experiences a significant loss of $500 billion attributed to a decline in software quality. Enhancing software quality can be achieved through the implementation of automated testing, ensuring that the final product aligns with user specifications and requirements. Testing techniques can be categorized into two main types: functional testing and structural testing. Functional requirements play a crucial role in functional testing, commonly referred to as black box testing, while structural testing, also known as white box testing, focuses on the internal coding aspects. Gray box testing refers to the integration of both black box and white box testing methodologies [2]. Software testing involves the systematic verification and validation processes to ensure that the software meets all specified business and technical requirements. Approximately 50% of the resources allocated for software development are utilized during the software testing phase. Automated testing can significantly reduce software development costs. A test case includes a unique identifier, references to requirements from a software specification, a sequence of steps, events, preconditions, input, output, expected results, and actual results, serving as input for software testing [10, 11]. Throughout the years, various software techniques have been effectively utilized in the domain of software testing, including genetic algorithms, neural networks, and fuzzy logic. The process of generating test cases can be framed as an optimization problem utilizing a meta-heuristic search technique known as the Genetic Algorithm [12, 13, 14]. Genetic Algorithms have been utilized in various applications Optimization challenges in software testing can be addressed through the automatic generation of

test plans for functionality testing, as parallelism and search space operations are critical characteristics. This paper provides an overview of the efficient application of genetic algorithms in the generation of test cases for software testing. The paper is divided into four sections. Section 1 includes the Introduction, section 2 provides a brief overview of a Genetic Algorithm, section 3 discusses the related work in the area of test case generation utilizing Genetic Algorithms for functional testing, and section 4 presents the conclusion and outlines future work.

An Introduction to Genetic Algorithm

A genetic algorithm is an evolutionary algorithm developed by John Holland in 1970. This tool is capable of addressing numerous complex real-life challenges by automatically generating high-quality test data [17, 18, 19]. This approach has developed into a practical and robust optimization technique and search method, drawing inspiration from the natural process of evolution through the selection of the fittest individuals. This approach effectively addresses a series of challenges with minimal information [11, 20]. A specific problem can be addressed through a population of chromosomes, which consist of strings of binary digits, with each digit referred to as a gene. This population can be generated randomly. There are three distinct types of operators utilized in the process of genetic algorithms: selection, crossover, and mutation [11, 18].

Selection: The selection operator is utilized to identify the most suitable parents for executing subsequent genetic algorithm operations. The selection process is typically conducted based on the fitness value of the individuals, derived from the fitness function. A fitness function is defined as a specific function that, based on certain criteria, returns a numerical value indicating the acceptability of the program. This function is utilized in the selection process to identify the optimal point, allowing the variants to progress to the subsequent iteration [21, 22]. The likelihood, proximity to boundary value, and branch coverage are critical components of an effective fitness function. There are six distinct types of selection methods: roulette wheel, stochastic universal sampling, linear rank, exponential rank, binary tournament, and truncation.

The crossover operation, also known as recombination, is implemented following the selection process. This operation involves the exchange of genes or sequences of bits between two selected individuals. Various types of crossover operators are employed for binary encoding, including one-point, two-point, uniform, and arithmetic methods. The crossover process is conducted multiple times using various parent individuals. Mutation is executed following crossover when the mutation probability is applicable for the specified iteration.

Mutation serves to preserve genetic diversity within a population by modifying chromosomes to incorporate beneficial traits. There are six primary types of mutation operators utilized in genetic algorithms: Bit string, flip bit, boundary, uniform, non-uniform, and Gaussian.

Related Work

This section offers a concise overview of the various hybridized genetic algorithm techniques utilized in software testing. Mark Last and colleagues [14] proposed a hybrid fuzzy-based genetic algorithm to generate test cases for mutation testing. This study identified a limited set of test cases. The deficiencies in test cases are revealed through the application of altered versions of the original method. The proposed approach employs a Fuzzy Logic Controller (FLC) to determine the probability of crossover, which varies based on the age intervals assigned throughout the lifespan. The age and lifespan of chromosomes (parents) are determined by the FLC state variables. An effective set of test cases is generated for a Boolean expression comprising 100 Boolean attributes through the application of three logical operators: AND, OR, and NOT. An external application generates the correct expression randomly, and a straightforward function is evaluated for each test case to produce an erroneous expression. Francisca Eanuelle *et al*. [20] have demonstrated the improvement of test plans through the application of the GA method for functionality testing. The method is implemented impartially to prevent any influence from the expert. The fitness function utilized by them is presented in equation (1).

$$f_p = \sum_{i=1}^{k-1} t(l_i \rightarrow l_{i+1})$$

Where p = l1, l2,…,lk is a test plan or sequence of operations and t is a transition function for converting one operation li to the next operation li+1 in a sequence. The sequence is considered as better if the value of fitness function is high. Ruilian Zhao *et al*. [24], used the GA and neural network for the functional testing of the software under test, and they applied the improved Genetic algorithm to the function model, created using neural network. The following fitness function, shown in equation (2) is used.

$$f = \begin{cases} \dfrac{1}{|c-g|}, & c \neq g \\ f_{max}, & |c-g| \leq 10^{-8} \end{cases}$$

In this context, c denotes the actual output, while g signifies the goal output of the software being evaluated. When the fitness value of the proposed algorithm meets or exceeds the maximum potential outcome, the algorithm concludes its execution, and the current individual is designated as the optimal test inputs for the respective outputs. The authors determined that the proposed GA is capable of generating superior test cases with enhanced efficiency.

Li, Zhang, and Kou [25] utilized the GA to obtain the local optimal solution for a specific problem, resulting in enhanced performance. A new algorithm known as the Genetic-Particle Swarm Mixed Algorithm (GPSMA) was applied to automatically generate software test data. The proposed technique employs the update mode for each individual to substitute the mutation process within the algorithm that relies on population division. The proposed algorithm is capable of generating and searching for specific test data within a domain to meet the test condition requirements.

Xuan Peng *et al*. [15] proposed an approach known as US-RDG for web applications, focusing on gray box testing. This method integrates User Session data with the Request Dependence Graph (RDG) to facilitate the automatic generation of test cases. Their simulated results indicate that US-RDG effects outperform traditional user session-based testing by achieving higher path coverage and fault detection rates with a smaller test suite. The concept was utilized as a transition relation represented by the sequence "page → request → page". Transition relations indicate the relationship between pages and requests. From structural analysis using RDG, specific transition relations in the

application can be extracted. Ultimately, a GA heuristic was proposed to generate test cases that cover the maximum number of transition relations by integrating various user sessions. The performance of US-RDG in test case generation for web applications has been found to be highly effective. The fitness function of a chromosome was utilized as demonstrated in equation (3).

$$\text{Fitness} = (\alpha * |\text{CDTR}| + |\text{CLTR}|) / (\alpha * |\text{DTR}| + |\text{LTR}|) \quad (3)$$

Where |CDTR| and |CLTR| denotes the number of data and link dependence transition relations covered in the chromosome. The fitness value achieves 1 when a chromosome covers all the data and link dependence in transition relations present in the specific application. The authors introduced a parameter α, indicating the coefficient of the data dependence transition relation. They have taken α as much as greater than 1 to increase the proportion of the data dependence transition relation and assigned 1 to the coefficient of the link dependence in transition relation. The chromosomes have a bigger fitness which covers more data dependence transition relations.

Ali Shahbazi *et al.* [5], used a multi objective optimization in black box string test case generation for random testing and adaptive random testing. The authors examined many string distance functions and hence they introduce two objectives for effective string test cases such as the length distribution of the string test cases and the diversity control of the test cases within a test set. They used one diversity- based fitness function to generate optimized test sets to reveal faults more effectively and it is shown in equation (4).

$$F_D = \sum_{i=1}^{test\ set\ size} dist\left(t_i, \beta\left(t_i, test\ set\right)\right)$$

In the above function ti denotes the ith test case and β is its nearest test case in the test set and the summation is performed between the two distances test cases. They found the higher value of the fitness function results, the more diverse distribution of test cases. After an extensive study of different testing techniques, we came to learn GA parameter is efficiently used for generating test cases in functional software testing.

Conclusion and Future Work

This paper examines the role of various hybridized Genetic Algorithms in enhancing the efficiency of software testing through the generation of an increasing number of test cases. An automatic test case generator is provided. The genetic algorithm can be utilized alongside neural networks and fuzzy systems to conduct various testing methods aimed at enhancing performance.

In the future, a new hybridized algorithm can be developed by utilizing an improved fitness function, which will assist in evaluating the efficiency of test cases. This approach will further enhance the effectiveness of test results by adjusting input parameters, increasing the number of generations, and obtaining values for varying population sizes. There are plans to implement a hybridized genetic algorithm in conjunction with other soft computing techniques, such as neural networks, to optimize test case generation for web-based application software.

References

1. Shivani A, Pandya V. Bridge between black box and white box – gray box testing technique. International Journal of Electronics and Computer Science Engineering. 2012;2(1):175–85.
2. Chauhan N. Software testing: principles and practices. Oxford: Oxford University Press; c2010.
3. Jørgensen PC. Software testing: a craftsman's approach. 3rd ed. Boca Raton: CRC Press; c2008.
4. Shahbazi A, Miller J. Black-box string test case generation through a multi-objective optimization. IEEE Transactions on Software Engineering. 2016, 42(4).
5. Michael CC, McGraw GE, Schatz MA, Walton CC. Genetic algorithms for dynamic test data generation. In: Proceedings of the 1997 International Conference on Automated Software Engineering (ASE'97). IEEE; c1997.
6. Doungsaard C, Dahal K, Hossain A, Suwannasart T. Test data generation from UML state machine diagrams using GAs. In: International Conference on Software Engineering Advances (ICSEA 2007). IEEE; c2007.
7. Srivastava PP, Kim T. Application of genetic algorithm in software testing. International Journal of Software Engineering and Its Applications. 2009;3(4):87–96.
8. Berndt DJ, Watkins A. High volume software testing using genetic algorithms. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences. Vol. 9. Washington (DC): IEEE Computer Society; c2005. p. 318–26.
9. Dixit S, Tomar P. Automated test data generation using computational intelligence. In: Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 4th International Conference on. IEEE; c2015.
10. Sharma A, Patani R, Aggarwal A. Software testing using genetic algorithms. International Journal of Computer Science and Engineering Survey. 2016, 7(2).
11. Moataz AA, Ali F. Multiple-path testing for cross-site scripting using genetic algorithms. Journal of Systems Architecture. 2016;64:50–62.
12. Yang S, Man T, Xu J, Zeng F, Li K. RGA: a lightweight and effective regeneration genetic algorithm for coverage-oriented software test data generation. Information and Software Technology. 2016;76:19–30.
13. Last M, Eyal S. Effective black-box testing with genetic algorithms. In: Lecture Notes in Computer Science. Springer; c2006. p. 134–48.
14. Peng X, Lu L. A new approach for session-based test case generation by GA. In: Communication Software and Networks (ICCSN), 3rd International Conference on. IEEE; c2011. p. 91–6.
15. Zhao R, Lv S. Neural network-based test case generation using genetic algorithm. In: 13th IEEE International Symposium on Pacific Rim Dependable Computing. IEEE; c2007. p. 97–100.
16. Srivastava PR, Kim TH. Application of genetic algorithm in software testing. International Journal of Software Engineering and Its Applications. 2009;3(4):87–96.
17. Ribeiro JCB, Zenha-Rela MA, de Vega FF. A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object-oriented software. In: AST'08 Proceedings. ACM; c2008.
18. Goldberg DE. Genetic algorithms in search, optimization and machine learning. Massachusetts: Addison-Wesley; c1989.

19. Wappler S, Lammermann F. Using evolutionary algorithms for unit testing of object-oriented software. In: GECCO Proceedings. ACM; c2005. p. 1925–32.
20. Vieira FE, Martins F, Silva R, Menezes R, Braga M. Using genetic algorithms for test plans for functional testing. In: 44th ACM SE Proceedings; c2006. p. 140–5.
21. Mathur AP. Foundations of software testing. 1st ed. Pearson Education; c2008.
22. Rauf A, Anwar S, Jaffer MA, Shahid AA. Automated GUI test coverage analysis using GA. In: Information Technology: New Generations (ITNG), Seventh International Conference on. IEEE; c2010. p. 1057–62.

How to Cite This Article
Saxena S, Rizbi QM. A Review of a Randomized Approach to Test Case Generation Using Genetic Algorithms. International Journal of Multi Research. 2025; 1(6): 14-17.