International Journal of IT & Knowledge Management (ISSN: 0973-4414)
Volume 15 • Issue 1 pp. 23-27 June 2021 - Dec 2021 www.csjournals.com

A Hybrid Evolutionary Approach for Software Test Suites Optimization Using Genetic Algorithm

Dr. Surjeet Singh
Department of Computer Science, G.M.N. College, Ambala Cantt., India

Abstract: Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, history is full of such examples. Software testing is a vital part of the software development process, and automate test data generation contributes to reduce cost, time and efforts. It is the process use to access the quality of computer software. Software testing is a critical element of software quality assurance and represents the final review of specification, design, and coding. Software testing is a crucial phase in the complete software development life cycle, aimed at ensuring quality, reliability, and correctness of software systems. However, exhaustive testing is often impossible and impractical due to limited time, cost, and resources. Test optimization techniques are therefore essential to reduce testing effort and time while maintaining high fault detection capability. Genetic Algorithms (GAs), a class of evolutionary computation techniques, have been widely used in search-based software testing for test case generation, selection, and prioritization. Despite their effectiveness, traditional GAs often suffers from premature convergence and limited adaptability. To address these challenges, this paper proposes a Hybrid Genetic Algorithm (HGA) approach that integrates GA with heuristic and local search techniques to enhance test optimization efficiency. The proposed model aims to optimize test suites by minimizing redundancy, maximizing code coverage, and improving fault detection capability. Experimental analysis and comparative evaluation demonstrate that the hybrid approach outperforms conventional genetic algorithms in terms of efficiency, scalability, and solution quality.

Keywords: Software Testing, Genetic Algorithm, Hybrid Genetic Algorithm, Test Case Optimization, Search-Based Software Engineering.

1. Introduction

Quality of the software is purely depends upon the quantity and quality of the software testing. Software testing is any activity aimed at evaluating an attribute or proficiency of a program or system as well as determining that it meets its required significances. Although crucial to software quality and widely deployed by programmers and testers, software testing still remains an art, due to imperfect understanding of the ethics of software. To assist in program testing various kinds of techniques have been developed including automation process. A black box approach to testing is supported by generating test cases that cover the program's expected functionality**.** Test data can be automatically generated to support a black box testing. Test data generation in program testing is the process of identifying a set of test data which satisfies given testing criterion. Most of the existing test data generators use symbolic evaluation to derive test data. Rapidly changing software and computing environments present many challenges for successful and efficient testing in practice. Past research in testing of evolving software has resulted in techniques that attempt to automate the process and few of these techniques have been successfully implemented for various kind of software, existing techniques show promise for use in industry. Software systems have become increasingly complex and integral to modern

society, spanning domains such as healthcare, transportation, and communication. Ensuring software quality is therefore of paramount importance. Software testing plays a vital role in identifying bugs, validating requirements, and ensuring reliability before deployment. However, software testing is often time-consuming, resource-intensive, and expensive, accounting for a significant portion of the total development cost. Traditional testing techniques, such as manual testing and exhaustive testing, are no longer feasible for large-scale and complex systems. As a result, automated and intelligent testing approaches have gained considerable attention. Among these, Search-Based Software Testing (SBST) has emerged as an effective prototype that formulates testing problems as optimization problems and applies metaheuristic algorithms to solve the problem. Genetic Algorithms (GAs), inspired by the principles of natural evolution, are one of the most widely used metaheuristic techniques in SBST.GA have been successfully applied to test case generation, test suite minimization, test case prioritization, and regression testing. Despite their success, standard GAs face many limitations, such as slow convergence, susceptibility to local optima, and dependency on different parameters. To overcome these limitations, hybrid approaches that combine GAs with other optimization or learning techniques have been proposed. This paper presents a Hybrid Genetic Algorithm (HGA) approach for efficient software test optimization. The proposed method integrates GA with heuristic-based local search mechanisms to improve exploration and exploitation capabilities, resulting in better optimized test suites.

2. Background and Related Research

There are many approaches to select test cases while doing software testing. One simple approach is called Random Testing (RT), which randomly selects test cases/sequences of events from the input domain (that is, the set of all possible inputs) [Ham02, Mye79]. The advantages of RT include its low cost, ability to generate numerous test cases automatically, and the generation of test cases in the absence of the software specification and source code. Apart from these, RT brings “randomnesses” into the testing process. Such randomnesses can best reflect the chaos of system operational environment; as a result, RT can detect certain failures unable to be revealed by deterministic approaches. All these advantages make RT irreplaceable and so popularly used in industry for revealing software failures [CM90, DKSM03, FM2000, MFS90, MKLMMNS95, Mill05, Nym, SMA05, Slu98, YSO03]. This approach may yield a large number of event sequences that are not legal & hence not executable, wasting valuable resources. Moreover, the test designer has no control over choice of event sequences; they may not have acceptable test coverage. Random testing selects arbitrarily test data from the input domain & then these test data are applied to the program under test. The automatic production of random test data, drawn from uniform distribution, should be the default method by which other systems should be judged, [DCI87]. The random generation of tests identifies members of the subdomains arbitrarily, with a homogeneous probability which is related to the cardinality of the subdomains. Under these circumstances, the chances of testing a function, whose subdomain has a low cardinality with regard to the domain as a whole, is much reduced. A random number generator generates the test data with no use of feedback from previous tests. The tests are passed to the procedure under test, in the hope that all branches will be traversed [WWF87].

2.1 Software Test Optimization

Software test optimization aims to improve the efficiency and effectiveness of the testing process by reducing redundant test cases, minimizing execution time, and maximizing fault detection and coverage. Common test optimization techniques include:

- Test suite minimization
- Test case selection
- Test case prioritization
- Automated test generation

These techniques are particularly important in regression testing, where frequent software changes require repeated testing.

2.2 Genetic Algorithms in Software Testing

Genetic Algorithms are population-based optimization techniques that simulate the process of natural selection. A typical GA consists of:

- Population initialization
- Fitness function evaluation
- Selection
- Crossover
- Mutation
- Termination

In software testing, GA chromosomes often represent test cases or test suites, while fitness functions measure criteria such as code coverage, fault detection, or execution cost. GAs are effective in handling large search spaces and complex optimization problems. However, they may converge prematurely or require extensive parameter tuning.

2.3 Hybrid Genetic Algorithms

Hybrid Genetic Algorithms combine GA with other techniques such as local search, hill climbing, simulated annealing, fuzzy logic, or machine learning. The goal of hybridization is to balance global search capability (exploration) with local refinement (exploitation). Hybrid approaches have shown improved performance in various optimization domains, including scheduling, routing, and software engineering.

3. Hybrid Genetic Algorithm Approach

3.1 Overview of the Proposed Model

The proposed Hybrid Genetic Algorithm (HGA) framework integrates a standard GA with a heuristic local search component. The GA performs global exploration of the solution space, while the local search enhances promising solutions by fine-tuning them.

The main objectives of the proposed approach are:

- Minimize the size of the test suite
- Maximize code coverage
- Improve fault detection effectiveness
- Reduce test execution cost

3.2 Chromosome Representation

Each chromosome represents a candidate test suite. Individual genes correspond to test cases, encoded in binary or integer form depending on the application domain. A value of ‘1’ indicates inclusion of a test case, while ‘0’ indicates exclusion.

3.3 Fitness Function Design

The fitness function plays a critical role in guiding the optimization process. In the proposed model, a multi-objective fitness function is defined as:

Fitness = w_1 \times Coverage + w_2 \times FaultDetection - w_3 \times Cost

where w_1, w_2, w_3 are weighting factors representing the relative importance of coverage, fault detection, and execution cost.

3.4 Genetic Operators

- Selection: Tournament selection is used to choose parents for reproduction.
- Crossover: Single-point or uniform crossover is applied to generate offspring.
- Mutation: Bit-flip mutation introduces diversity and prevents premature convergence.

3.5 Hybridization with Local Search

After genetic operations, a heuristic local search is applied to elite individuals. The local search explores neighboring solutions by modifying a small number of genes to further improve fitness. This hybridization significantly enhances convergence speed and solution quality.

3.6 Algorithm Workflow

1. Initialize population
2. Evaluate fitness
3. Apply selection, crossover, and mutation
4. Apply local search to elite individuals
5. Update population
6. Repeat until termination condition is met

4. Experimental Evaluation and Results

To evaluate the effectiveness of the proposed HGA, experiments were conducted on benchmark software programs with varying complexity. The performance of HGA was compared with a standard Genetic Algorithm.

4.1 Evaluation Metrics

- Test suite size
- Code coverage percentage
- Fault detection rate
- Execution time

4.2 Results and Analysis

Experimental results indicate that the proposed HGA:

- Achieves higher code coverage with fewer test cases
- Detects more faults compared to standard GA
- Converges faster due to local search enhancement
- Demonstrates better scalability for large test suites

The hybrid approach consistently outperformed the traditional GA across all metrics, confirming its effectiveness for software test optimization.

5. Conclusion

This paper presented a Hybrid Genetic Algorithm approach for efficient software test optimization. By integrating genetic algorithms with heuristic local search techniques, the

proposed model addresses key limitations of traditional GAs, such as premature convergence and limited adaptability. Experimental results demonstrate that the hybrid approach significantly improves test suite optimization in terms of coverage, fault detection, and cost efficiency. The proposed framework provides a robust and scalable solution for automated software testing and can be effectively applied to regression testing and large-scale software systems. Future work may explore integration with machine learning techniques, adaptive parameter tuning, and application to cloud-based and microservices architectures.

References:

[CM90] R. Cobb and H. D. Mills. Engineering software under statistical quality control. IEEE Software, 7(6):45–54, 1990.
[DCI87] Ince, D.C.:'The automatic generation of test data', The Computer Journal, Vol. 30, No. 1, pp. 63-69, 1987.
[DKSM03] T. Dab'oczi, I. Koll'ar, G. Simon, and T. Megyeri. Automatic testing of graphical user interfaces. In Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference 2003 (IMTC '03), pages 441–445, Vail, CO, USA, 2003.
[FM2000] J. E. Forrester and B. P. Miller. An empirical study of the robustness of Windows NT applications using random testing. In Proceedings of the 4th USENIX Windows Systems Symposium, pages 59–68, Seattle, 2000.
[HAM02] R. Hamlet. Random testing. In J. Marciniak, editor, Encyclopedia of Software Engineering. John Wiley & Sons, second edition, 2002.
[MFS90] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. Communications of the ACM, 33(12):32–44, 1990.
[Mill05] E. Miller. Website testing. http://www.soft.com/eValid/Technology/White.
Papers/website.testing.html, Software Research, Inc., 2005.
[MKLMMNS95] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services. Technical Report CS-TR-1995-1268, University of Wisconsin, 1995.
[Mye79] G. J. Myers. The Art of Software Testing. Wiley, New York, second edition, 1979.
[Nym] N. Nyman. In defense of monkey testing: Random testing can find bugs, even in well engineered software. http: //www.softtest.org/sigs/material/nnyman2.htm, Microsoft Corporation.
[Slu98] D. Slutz. Massive stochastic testing of SQL. In Proceedings of the 24th International Conference on Very Large Databases (VLDB 98), pages 618–622, 1998.
[SMA05] K. Sen, D. Marinov, and G. Agha. Cute: a concolic unit testing engine for c. In ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, pages 263–272, New York, NY, USA, 2005.ACM Press.
[WWF87] Watt D. A., Wichman B. A., & Sayward F. G., Findlay W.: 'ADA language and methodology', 1987.
[YSO03] T. Yoshikawa, K. Shimura, and T. Ozawa. Random program generator for Java JIT compiler test system. In Proceedings of the 3rd International Conference on Quality Software (QSIC 2003), pages 20–24. IEEE Computer Society Press, 2003.