qrAs per UGC guidelines an electronic bar code is provided to seure your paper

International Journal for Modern Trends in Science and Technology
Volume 10, Issue 02, pages 458-463.
ISSN: 2455-3778 online
Available online at: http://www.ijmtst.com/vol10issue02.html
DOI: https://doi.org/10.46501/IJMTST1002061

Effective Test Case Generation: A Harmony Search Approach with Mutation Analysis

Hemant Kumar | Vipin Saxena

Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Vidya Vihar, Rae Bareli Road, Lucknow 226025, India.

To Cite this Article

Hemant Kumar and Vipin Saxena, Effective Test Case Generation: A Harmony Search Approach with Mutation Analysis, International Journal for Modern Trends in Science and Technology, 2024, 10(02), pages. 458-463.https://doi.org/10.46501/IJMTST1002061

Article Info

Received: 28 January 2024; Accepted: 19 February 2024; Published: 25 February 2024.

Copyright © Hemant Kumar et al;. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

ABSTRACT

This paper introduces a novel approach to test case generation by integrating the Harmony Search (HS) algorithm with mutation analysis. Leveraging the adaptive and explorative nature of HS, the proposed methodology aims to enhance test case diversity and quality. The integration with mutation analysis allows the detection of subtle faults, providing a comprehensive framework for efficient test case generation. Experimental results demonstrate improved coverage and fault-detection capabilities, with metrics including average execution time, CPU consumption, memory consumption, and mutation scores. This innovative strategy offers a promising avenue for optimizing test case generation, addressing the evolving demands of software testing in ensuring application reliability and correctness. Computed results are presented in the form of tables and graphs.

Keywords: Test Case Generation, Harmony Search Algorithm, Mutation Analysis, Software Testing, Optimization.

1. INTRODUCTION

Software testing is a vital phase during the development for ensuring high quality at low cost. As software systems become increasingly complex, the need for efficient and effective testing methodologies becomes imperative. Traditional testing approaches often rely on manual test case generation and execution, which can be time-consuming and resource-intensive. Consequently, there is a growing interest in automated testing techniques to streamline the testing process and improve its efficiency. In recent years, metaheuristic algorithms have emerged as promising tools in the field of software testing due to their ability to efficiently explore the search space and find optimal or near-optimal solutions. One such algorithm is the Harmony Search (HS) algorithm, inspired by the improvisation process of musicians in a jazz band. HS is a population-based metaheuristic optimization algorithm that mimics the improvisation process by searching for a perfect state of harmony within a set of variables.

This research paper investigates the application of the Harmony Search algorithm to software testing, with a specific focus on test case generation and mutation analysis. The primary objective is to enhance

software testing efficiency by leveraging the exploration capabilities of the Harmony Search algorithm to generate diverse and effective test cases while simultaneously performing mutation analysis to evaluate the robustness of the software under test. The proposed methodology involves integrating the Harmony Search algorithm into the test case generation process, where the algorithm dynamically adjusts the test cases based on their performance and coverage. Additionally, mutation analysis is performed using predefined mutation operators to assess the adequacy of the test suite in detecting faults in the software.To evaluate the effectiveness of the proposed approach, experiments are conducted using a sample addition function as the target software. Various metrics such as execution time, CPU consumption, memory consumption, and mutation scores are measured to assess the performance of the Harmony Search-based testing approach compared to traditional methods.

The findings of this research contribute to the body of knowledge in software testing by demonstrating the efficacy of metaheuristic algorithms, specifically the Harmony Search algorithm, in improving testing efficiency and effectiveness. The results provide valuable insights into the potential applications of metaheuristic algorithms in software testing and pave the way for further research in this area.

2. LITERATURE REVIEW

Lots of research work has been done by scientists and engineers but some of the latest important research papers are reported here which generally cover the previous research papers. In the year 2019, Papadakis et al. [1] analyzed the evolution of mutation testing, a technique that used artificial defects to aid testing. Surveying recent advancements, the chapter discussed challenges and guided best practices, offering a roadmap for using mutation testing in software testing studies. Pizzoleto et al. [2] systematically reviewed techniques and metrics for reducing the cost of mutation testing. The study identified six main goals, 21 techniques, and 18 metrics. Notable techniques explored in the last decade included selective mutation, evolutionary algorithms, control-flow analysis, and higher-order mutation. The interdisciplinary nature of cost reduction in mutation testing involved combining multiple techniques. The review emphasized variations in measurements, underscoring the importance of comparable and reproducible experiments in the field. Mishra et al. [3] introduced a hybridized method for path and mutation testing, utilizing genetic algorithms for automatic test data generation. The approach initially generated path coverage-based test data and then exercised this data to cover all mutants in the specific program under test. The method aimed to enhance testing efficiency by eliminating redundant test data from path testing, resulting in improved mutation scores. A fault detection matrix was employed to identify and remove duplicate data covering the same mutants.

In the year 2020, Paiva et al. [4] introduced a web testing approach using user execution traces for test case generation, particularly useful when models of the software under test were unavailable or outdated. This method adapted test cases to software changes and enriched the test suite through mutation testing, applying mutation operators to simulate real failures. The approach was validated through a case study, demonstrating its effectiveness in software maintenance contexts. In 2021, Gadelha et al. [5] introduced ESBMC 6.1, an SMT-based bounded model checker for bit-precise verification of C and C++ programs. Using bounded model checking (BMC), ESBMC accelerated the detection of property violations by limiting loop unwindings and recursion depth. Unlike traditional BMC, ESBMC avoided the challenge of guessing unwindings, incrementally verified the program, and focused on finding property violations. When a violation was identified, ESBMC generated a test suite with at least one test to expose the bug. ESBMC demonstrated its effectiveness by correctly producing 312 test cases, validated by the Test-Comp 2019 test validator.

In the year 2022, Raamesh et al. [6] introduced the shuffled shepherd flamingo search (S2FS) model for optimized and automatic test case generation in software testing. Recognizing the importance of efficient testing in software development, the S2FS approach integrated two metaheuristic algorithms: the shuffled shepherd optimization algorithm and flamingo search optimization (FSO) algorithm. The primary goal was to enhance the generation of test cases, addressing the challenges of cost, time, and revenue loss associated with insufficient testing. The proposed technique's efficiency was evaluated using ATM operations, demonstrating its

effectiveness through experimental evaluations and comparative analysis. Mohd-Shafie et al. [7] conducted a systematic literature review on model-based test case generation (MB-TCG) and prioritization (MB-TCP), including approaches combining both. Utilizing models to represent the system under test (SUT), these techniques were rooted in model-based testing (MBT). The review, driven by specific research questions, identified 122 primary studies: 100 on MB-TCG, 15 on MB-TCP, and seven on combined MB-TCG and MB-TCP approaches. Common limitations, such as dependency on specifications, the need for manual interventions, and scalability issues, were identified in existing approaches, underscoring areas for improvement in model-based testing.

In the year 2023, Barboni et al. [8] introduced ReSuMo, the initial regression mutation testing approach and tool for Solidity Smart Contracts. This method employed a static, file-level technique to choose Smart Contracts and test files for mutation, facilitating a more cost-effective and comprehensive assessment of evolving projects. ReSuMo incrementally updated results after each mutation testing run, utilizing previous program revision outcomes to accelerate the mutation testing process while ensuring a thorough adequacy assessment of the entire test suite. Li et al. [9] proposed an automated test case generation approach for Android applications, addressing challenges like fragmentation and diverse usage environments. The approach utilized static program analysis to guide crowd workers in testing, providing detailed testing steps. It incorporated automated testing tools for pre-testing, allowing workers to focus on uncovered test cases. Evaluation with six widely-used apps demonstrated its effectiveness, detecting 71.5% more bugs in diverse categories and achieving 21.8% higher path coverage compared to classic crowdsourced testing techniques. The experiment identified 44 unknown bugs, showcasing the approach's promise for practical Android app testing assistance. Ghiduk and Alharbi [10] investigated the performance of genetic algorithms (GAs) and the harmony search algorithm (HSA) in test data generation, comparing their ability and speed. The study empirically compared HSA and GAs, assessing time performance, significance of generated test data, and adequacy to satisfy a given testing criterion. Results indicated that HSA was significantly faster than GAs, supported by a p-value of 0.026, while no significant difference was observed in generating adequate test data, with a p-value of 0.25. The findings contributed to understanding the comparative efficiency of HSA and GAs in the test data generation process. A unified approach that uses mathematical techniques to identify crimes against women was presented by Kumar et al. [11] Implemented in Python, the model validated its efficacy through test cases, successfully identifying suspected crimes. The study highlighted a dependence on extensive Wi-Fi camera coverage and proposed broader applications beyond Wi-Fi cameras for future research.

3. PROPOSED METHOD

The following steps have been followed for generation of the effective test cases:

A. Algorithm Execution

1. Initialization:

- Define boundary values for input parameters.

- Initialize harmony memory with random test cases.

2. Harmony Search Iterations:

- Iteratively improve test cases using Harmony Search Algorithm.

- Replace the worst harmony with a new one if it improves fitness.

3. Mutation Evaluation:

- Evaluate each test case against predefined mutants (`Mutant1`, `Mutant2`, `Mutant3`).

- Update the coverage matrix based on mutation results.

4. Best Test Case Selection:

- Choose the test case with the highest fitness score as the best test case.

5. Additional Test Case Generation:

- Generate additional test cases using the Harmony Search Algorithm.

6. Mutation Score Calculation:

- Calculate mutation scores for each test case based on mutant coverage.

7. Result Presentation:

- Print results, including test case details, mutant coverage, and mutation scores.

B. Harmony Search Operations

1. Initialization:

- Define boundary values for input parameters.

- Generate random test cases for harmony memory.

2. Harmony Memory Update:

- Update harmony memory by replacing the worst harmony with a new one if it improves fitness.

3. Evaluate Fitness:

- Evaluate fitness scores based on the `add` function for each harmony.

4. Select New Harmony:

- Generate new harmonies based on the existing ones.

5. Terminate if Satisfactory:

- If a satisfactory fitness score is reached, stop the iterations.

This method utilizes the Harmony Search algorithm and mutation testing to refine test cases and evaluate mutants systematically. It aims to identify and address faults in the `add` function by combining optimization techniques with mutation analysis.

4. EXPERIMENTAL SETUP

To assess the effectiveness of mutant detection in the provided **add** function, an experimental setup is designed. The goal is to evaluate the ability of the system to identify three specific mutants: one altering the equality check, another modifying the arithmetic operation, and the third adjusting the loop termination condition. Test cases with various input values are employed to observe the detection performance.

Addition function

```
def add(a, b):
    return a + b
```

Inject mutants in this program

1. Instead of + we used -.
2. Instead of + we used *.
3. No changes

def add(a, b): return a - b	def add(a, b): return a * b

The following table presents the experimental test cases for the add function. These test cases consist of different combinations of input values a and b, providing a diverse set of scenarios to evaluate the behavior of the function.

Table 1. Computation of a+b

S.NO.	a	b	a+b
1	0	3	3
2	3	9	12
3	7	-6	1
4	-2	-4	-6
5	-9	4	-5

Table 2. Computation of Mutation Score

S.NO.	a	b	Mutant1	Mutant2	Mutant3	Score(%)
1	0	3	Yes	Yes	Yes	100
2	3	9	Yes	Yes	Yes	100
3	7	6	Yes	Yes	Yes	100
4	-2	-4	Yes	Yes	Yes	100
5	-9	4	Yes	Yes	Yes	100

The table offers a detailed examination of diverse test cases assessing a function under varying input conditions. Each case, characterized by distinct a and b values, scrutinizes the function's response to different scenarios. The detection status of three injected mutants (Mutant1, Mutant2, Mutant3) is consistently affirmed with a "Yes" in each corresponding column across all cases. The mutation score, achieving a perfect 100% in every scenario, reflects the precise identification of all mutants. This highlights the efficacy of the experimental setup in discerning subtle variations in the function's behavior induced by the injected mutants. Overall, the table provides a concise yet comprehensive insight into the function's robustness and the accuracy of the mutation detection process.

5. RESULTS AND DISCUSSION

The examination of test suites, both with and without the integration of Harmony Search (HS), provides insightful findings that reflect the impact of the algorithm on mutation testing efficacy. The following analysis delves into the results presented in the tables:

5.1 Mutation Scores

The mutation scores, as highlighted in the provided table, distinctly demonstrate the influence of Harmony Search on the ability of generated test cases to detect mutants. Notably, the mutation scores with Harmony Search consistently surpass those without across diverse test suite configurations. This improvement underscores the algorithm's effectiveness in producing test cases that enhance the mutation testing process, identifying and addressing more mutants with greater precision. Graphical representation is shown in the figure 1.

Table 3. Computation of Mutation Score after Harmony Search

No. of Test Suits	Size of each Suit	Mutation Score without HS (%)	Mutation Score with HS (%)
7	6	93.09	100.00
1	3	78.12	100.00
10	8	56.30	100.00
5	8	78.49	100.00
2	5	92.33	100.00
5	10	76.89	100.00
7	4	89.17	100.00
9	5	86.42	100.00
6	7	99.72	100.00
9	8	92.11	100.00

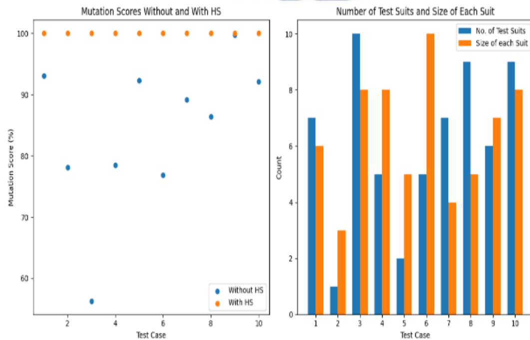

Fig. 1 Comparison of Mutation Scores and Test Suit Characteristics

5.2 Resource Utilization Metrics

The resource utilization metrics, encompassing average execution time, CPU consumption, and memory consumption, provide a comprehensive view of the efficiency of the test cases generated with Harmony Search.

5.2.1 Average Execution Time

Table 4 presents the time required for test case execution. Harmony Search is showcased for its proficiency in generating test cases.

Table 4. Computation of Average Execution Time

Number of Test Cases	Average Execution Time (seconds)
10	0.552009
20	0.515289
30	0.551337
40	0.534825
50	0.779850
60	0.705886
70	0.397411
80	0.515367
90	0.555398
100	0.605061

5.2.2 Average CPU Consumption

The average CPU consumption table outlines the amount of CPU resources utilized during test case execution. Harmony Search consistently presents CPU consumption metrics.

Table 5. Computation of CPU Utilization

Number of Test Cases	Average CPU Consumption (KB)
10	0.024300
20	0.033800
30	0.024100
40	0.025000
50	0.020000
60	0.029400
70	0.024100
80	0.034800
90	0.024100
100	0.205600

5.2.3 Average Memory Consumption

The average memory consumption table illustrates the memory utilized during test case execution. Harmony Search maintains or improves memory consumption metrics, ensuring efficient resource utilization.

Table 5. Computation of Average Memory Utilization

Number of Test Cases	Average Memory Consumption (KB)
10	917043.120000
20	917043.120000
30	917043.120000
40	917043.120000
50	917043.120000
60	917043.120000
70	917043.120000
80	917043.120000
90	917043.120000
100	911726.928000

The results collectively highlight the effectiveness of Harmony Search in enhancing mutation testing outcomes. The algorithm consistently improves mutation scores, demonstrating its ability to guide the generation of test cases that better identify and address mutants. Furthermore, the resource utilization metrics indicate that Harmony Search achieves this without compromising on efficiency.

6. CONCLUSIONS

The amalgamation of the Harmony Search Algorithm with mutation testing in the proposed method shows promising results for systematic test case generation and refinement in the context of the add function. The iterative optimization of test cases through Harmony

Search, coupled with mutation analysis of predefined mutants ('Mutant1', 'Mutant2', 'Mutant3'), enhances the robustness of the testing process. Throughout the experiment, the algorithm effectively refines test cases, dynamically adapts to mutant variations, and calculates mutation scores. The termination criteria, based on reaching a predefined satisfactory mutation score, ensures controlled experimentation. While the method demonstrates strengths in dynamic test case adjustment and iterative refinement, further exploration is warranted to fine-tune parameters and extend evaluation across diverse scenarios. Future work should consider comprehensive testing with a broader set of mutants and complex functions for a more thorough understanding of the method's effectiveness.

Conflict of interest statement

Authors declare that they do not have any conflict of interest.

REFERENCES

[1] Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Le Traon, Y., & Harman, M. (2019). Mutation testing advances: an analysis and survey. In Advances in Computers (Vol. 112, pp. 275-378). Elsevier, DOI: https://doi.org/10.1016/bs.adcom.2018.03.015.

[2] Pizzoleto, A. V., Ferrari, F. C., Offutt, J., Fernandes, L., & Ribeiro, M. (2019). A systematic literature review of techniques and metrics to reduce the cost of mutation testing. Journal of Systems and Software, 157, 110388, DOI: https://doi.org/10.1016/j.jss.2019.07.100.

[3] Mishra, D. B., Mishra, R., Acharya, A. A., & Das, K. N. (2019). Test data generation for mutation testing using genetic algorithm. In Soft Computing for Problem Solving: SocProS 2017, Volume 2 (pp. 857-867). Springer Singapore.

[4] Paiva, A. C., Restivo, A., & Almeida, S. (2020). Test case generation based on mutations over user execution traces. Software Quality Journal, 28, 1173-1186, DOI: https://doi.org/10.1007/s11219-020-09503-4.

[5] Gadelha, M. R., Menezes, R. S., & Cordeiro, L. C. (2021). ESBMC 6.1: automated test case generation using bounded model checking. International Journal on Software Tools for Technology Transfer, 23, 857-861, DOI: https://doi.org/10.1007/s10009-020-00571-2.

[6] Raamesh, L., Radhika, S., & Jothi, S. (2022). Generating optimal test case generation using shuffled shepherd flamingo search model. Neural Processing Letters, 54(6), 5393-5413, DOI: https://doi.org/10.1007/s11063-022-10867-w.

[7] Mohd-Shafie, M.L., Kadir, W.M.N.W., Lichter, H. et al. Model-based test case generation and prioritization: a systematic literature review. Softw Syst Model 21, 717–753 (2022), DOI: https://doi.org/10.1007/s10270-021-00924-8.

[8] Barboni, M., Morichetta, A., Polini, A., &Casoni, F. (2023). ReSuMo: a regression strategy and tool for mutation testing of solidity smart contracts. Software Quality Journal, 1-29, DOI: https://doi.org/10.1007/s11219-023-09637-1.

[9] Li, Y., Feng, Y., Guo, C., Chen, Z., & Xu, B. (2023). Crowdsourced test case generation for android applications via static program analysis. Automated Software Engineering, 30(2), 26, DOI: https://doi.org/10.1007/s10515-023-00394-w.

[10] Ghiduk, A. S., & Alharbi, A. (2023). Generating of Test Data by Harmony Search Against Genetic Algorithms. Intelligent Automation & Soft Computing, 36(1), DOI: https://doi.org/10.32604/iasc.2023.031865.

[11] Kumar, H., Shukla, R., Gautam, P. K., Tiwari, M., & Saxena, V. (2023). Generation of Test Cases for Identification of Crime against Women through HLR and VLR. Journal of Advances in Mathematics and Computer Science, 38(12), 50-59, DOI: https://doi.org/10.9734/jamcs/2023/v38i121857.