

Optimized automated testing: test case generation and maintenance using latent semantic analysis-based TextRank and particle swarm optimization algorithms

Baswaraju Swathi¹, Soma Sekhar Kolisetty²

¹Department of Computer Science and Engineering (Data Science), New Horizon College of Engineering, Bengaluru, India

²Department of Computer Science and Engineering, University College of Engineering Narasaraopet JNTUK, Guntur, India

Article Info

Article history:

Received Nov 29, 2023

Revised Feb 29, 2024

Accepted Apr 16, 2024

Keywords:

Automated testing

Latent semantic analysis-based TextRank

Natural language processing

Particle swarm optimization

Test case generation

Test case maintenance

ABSTRACT

Software development would have to include automated testing to ensure the finished product and performs as intended. However, the process of test case generation and maintenance can be time-consuming and error-prone, especially when manual methods are used. This research proposes a new approach to improve the efficiency and accuracy of automated testing using latent semantic analysis (LSA)-based TextRank (TR) and particle swarm optimization (PSO) algorithms. The study aims to evaluate the effectiveness of these algorithms in generating and optimizing test cases based on requirements analysis. To retrieve key information from the criteria, methods including text classification (TC), named entity recognition (NER), and sentiment analysis (SA) are used to evaluate the text. Test cases are then generated using LSA-based TR for text summarization and PSO for optimization. The aim of this work is to identify any limitations that need to be addressed and to evaluate the overall efficiency and accuracy of automated testing (AT) using proposed algorithms. The results of this research are expected to have important implications for the software industry, helps to improve the overall efficiency and accuracy of AT. The findings could guide future research that led to the creation of more advanced and effective tools for AT.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Baswaraju Swathi

Department of Computer Science and Engineering (Data Science), New Horizon College of Engineering Bengaluru, Karnataka, India

Email: baswarajuswathi@gmail.com

1. INTRODUCTION

Automated testing (AT) is critical in software applications to be high-quality and reliable. Test cases serve as a roadmap for evaluating the functionality of software, and their effective generation and maintenance is crucial for the success of Automated Testing. However, the manual process of creating and maintaining test cases can be time-consuming and prone to errors, which can negatively impact the overall efficiency and accuracy of the testing process. Test case generation (TCG) is the process of creating test cases that can be used to evaluate the functionality of software applications [1]. TCG's objective is to find any possible pitfalls or flaws in the software and make sure it works as planned. As it assists in guaranteeing that the software is properly assessed and adheres with the criteria, efficient TCG is essential for the success of AT. TCG can be done in a variety of ways, both manually and automatically [2]. Effective test case maintenance (TCM) is crucial for the success of AT, as it helps to ensure that the test cases are accurate and provide a comprehensive evaluation of the software [3]. There are different approaches to effective TCM,

including manual methods and automated methods. Manual test case maintenance (MTCM) involves updating test cases by hand, which can be time-consuming and prone to errors. Automated TCM, on the other hand, uses algorithms and tools to update test cases automatically, reducing the time and effort required for manual methods.

Optimization of TCG and TCM is a crucial task in software testing. Several studies have proposed various techniques to optimize the process of TCG and TCM. An approach with an artificial bee colony (ABC) algorithm has proven to be more effective than the cuckoo search algorithm (CSA) in optimizing automated test suites for multiple programs. This approach generated a smaller test suite while achieving maximum path coverage [4]. The work in [5] discusses the importance of software testing and the use of automated techniques for TCG. It is suggested that the CSBCA approach, that integrates the cuckoo search (CS) and bee colony algorithm (BCA), be used to efficiently optimize test scenarios and generate pathway convergence. As compared to other known optimization algorithms, CSBCA performs better.

Furthermore, the author suggests that software testing is costly and time-consuming for software engineers, as it involves selecting the right test cases. Automatic TCG is difficult because it is a complex problem. To solve this problem, the paper proposes using nature-inspired optimization algorithms, like genetic algorithms and mutation analysis, for automatic TCG [6]. Another study focuses on assessing how software test automation affects cost, quality, and effort in contrast to other variables like project duration and software complexity. The findings reveal that automation has a considerable positive impact on software quality while minimizing costs and effort. Furthermore, the results suggest that automation can enhance software quality regardless of the software's complexity level [7].

Another study, Neto *et al.* [8] analyzes the correlation between changes in a system and their effect on test quality, and it proposes a model for predicting the impact of changes on test quality. The outcomes demonstrate that the suggested approach is successful in anticipating whether system changes will affect test reliability. Zhongsheng [9] suggests a method to optimize web application testing by analyzing user logs and dividing user sessions into different test suites. The strategy also makes use of crossover to optimize grouping and a genetic algorithm (GA) to create new test cases. According to the study's findings, the proposed method is a practical means of generating fewer test cases while maintaining compliance with the web application's initial design. Maragathavalli [10] discusses the use of search-based (SB) software engineering in TCG, with a focus on evolutionary testing using metaheuristic search methods. In a comparison of the efficacy of GA-based test procedures and random testing systems, it was discovered that GA-based testing outperformed random testing as the difficulty of the software or input domain increased.

Detecting faults more quickly and reducing the overall effort required for testing is the goal of a new hybrid algorithm [11]. This algorithm combines GA and particle swarm optimization (PSO) to prioritize test cases in regression testing. By reordering test cases based on their impact on detecting faults, the efficiency of regression testing is improved. For the purpose of TCG in object-oriented software, an advanced artificial immune system has been developed by incorporating differential evolution (DE). The primary objective is to promote the variety of test cases generated by utilizing DE's efficient search space exploration [12]. The empirical findings validate the efficacy of the proposed methodology in producing superior test cases with enhanced ability to detect faults. Utilizing a combination of Ant colony optimization (ACO) and neural networks, this paper introduces a fresh and unorthodox method for automating TCG. By harnessing the power of ACO, potential paths in the software under examination are unearthed, while neural networks take charge of learning and forecasting the viability of the produced test cases [13]. The experimental findings unequivocally demonstrate the capability of this approach to yield a multitude of robust test cases, thereby enhancing the overall efficacy of automated testing.

The work proposes a novel approach to automated software testing that combines latent semantic analysis-based TextRank (LSA-TextRank) and PSO algorithms. This approach aims to optimize TCG and TCM by analyzing natural language descriptions of software features and requirements and identifying important keywords and phrases. The proposed study shows reliable results in terms of improving the efficiency and effectiveness of AT.

2. DATASETS

The self-admitted technical debt (SATD) dataset is a collection of comments from various software projects that are hosted on GitHub, a popular platform for hosting software projects [14]. The comments in the dataset have been labeled as either containing or not containing SATD, which is a term used to describe when software developers knowingly create low-quality code that will need to be refactored or improved in the future. The dataset includes a total of 62,878 comments from 1,035 software projects, which provides a diverse range of examples of SATD. A train and test data were used to partition the collection into two halves. To train machine learning models to find SATD, 70% of the data is used in the training phase. The

other 30% of the observations make the test set, which is used to assess how well the proposed methods performed.

3. METHOD

The proposed method for optimizing TCG and TCM in AT involves several key steps. The first step is the acquisition of a raw text dataset, which serves as the basis for generating test cases. This dataset is then subjected to content filtering to eliminate irrelevant or redundant information. Natural language processing (NLP) techniques are applied using LSA and TR algorithms to identify the most important concepts and keywords in the dataset. These algorithms use statistical methods to analyze the relationships between words and identify patterns in the data. Once the most relevant information has been extracted, a PSO algorithm is applied to optimize the selection and ordering of the identified concepts and keywords as shown in Figure 1. This optimization process ensures that the resulting summary is concise, accurate, and representative of the key information in the dataset. The final output of this process is a summary of the most important concepts and keywords, which can then be used as the basis for generating and maintaining automated test cases. By using a combination of content filtering, NLP techniques, and PSO optimization, this method provides a streamlined and effective approach to TCG and TCM in AT.

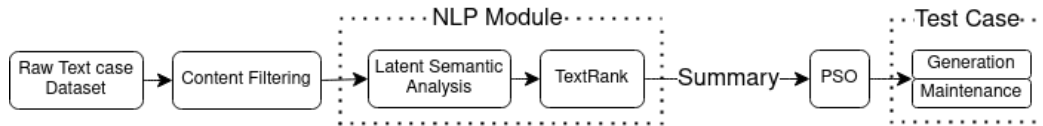


Figure 1. Flow chart for test case generation and maintenance in automated testing using latent semantic analysis-based TextRank (TR)

3.1. Data collection

Collect the test suite data from the software under test and use it as the input data for TCG. The first step is to identify a set of software projects hosted on GitHub that can be used for data collection. This could be done by selecting projects based on specific criteria, such as those written in a specific programming language or those with a certain level of activity. Once the projects are identified, the next step is to extract data from them. In this case, the data consists of comments made in the source code by developers. These comments are usually used to provide context or explanations about the code. The extracted comments are then filtered to remove any that are not relevant to the study. Providing a clear and concise description of the expected behavior or functionality of a software feature, comments from GitHub serve as a reference for developers and testers and can represent a test case. Including details like expected output, input values, and any specific conditions or constraints to be considered, these comments ensure that the implemented code meets specified requirements and passes the corresponding test.

Algorithm. LSA-based TextRank algorithm with PSO for TCG and TCM

Input: A software project hosted on GitHub, and the self-admitted technical debt (SATD) dataset, which is a collection of comments from various software projects also hosted on GitHub.

Output: A set of optimized test cases for the software project.

Method:

Step 1: Preprocess the document by removing stop words, stemming, and converting to lowercase.

Step 2: Create a term-document matrix using the preprocessed document.

Step 3: Apply SVD to the term-document matrix to get the LSA matrix. Use the following equation:

$$M = U * S * V^T \quad M' = U_k * S_k * V_k^T$$

where M stands for the term-document matrix (TDM), U for the left singular vector (LSV) matrix, S for the diagonal singular value (DSV) matrix, V^T for the right singular vector (RSV) matrix, and k for the number of finite dimensions.

Step 4: Calculate the cosine similarity between every pair of sentences using the LSA matrix. Use the following equation:

$$Sim(i, j) = (u_i * u_j) / (||u_i|| * ||u_j||)$$

where u_i and u_j are the LSA vectors representing sentences i and j , respectively, and $||\cdot||$ denotes the Euclidean norm.

Step 5: Create a graph where each node represents a sentence and the edges represent the similarity between the sentences. Apply the TextRank (TR) algorithm to the graph to compute the importance score for each sentence. Use the following equation:

$$Score(i) = (1/d) + d * \sum_{j \in In(i)} (w_{j, i} * score(j))$$

where d is a damping factor, $In(i)$ is the set of sentences that link to sentence i , $w\{j, i\}$ is the similarity between sentences i and j , and the term $(1-d)$ is a normalizing constant.

Step 6: Rank the sentences in descending order of their importance score.

Step 7: Initialize a PSO algorithm with the ranked sentences as particles, where each particle represents a potential test case. Define the objective functions to be the test case's efficacy and speed, as determined by the coding coverage and system performance, and establish the search area as the collection of all conceivable permutations of the high statements.

Step 8: Run the PSO algorithm for a specified number of iterations, updating the particles' positions and velocities according to the following equations:

$$v_i = w * v_i + c_1 * r_1 * (pbest_i - x_i) + c_2 * r_2 * (gbest - x_i) \quad x_i = x_i + v_i$$

where v_i is the velocity of particle i , w is the inertia weight, c_1 and c_2 are the cognitive and social parameters, respectively, r_1 and r_2 are random numbers, $pbest_i$ is the best position of particle i , x_i is the current position of particle i , and $gbest$ is the best position among all particles.

Step 9: Select the best test cases as the output of the PSO algorithm.

Step 10: Convert the selected test cases into a set of test scenarios by identifying the important topics and generating test scenarios for each topic.

Step 11: Use the test scenarios to test the software and identify any defects.

Step 12: Maintain the test cases by periodically updating them to reflect changes in the software.

3.2. Latent semantic analysis-based TextRank (LSA-TR)

LSA-based TextRank is an NLP algorithm that generates a summary of a document by selecting the most important sentences based on both their importance and relevance to the main topics. This approach is a combination of two algorithms: LSA and TR. LSA is a mathematical method for examining connections between phrases in a collection of documents. LSA works by constructing a matrix of the keywords in the text and the instances in which that appear. This matrix is then analyzed to determine the relationships between the terms and the documents. By reducing the dimensionality of the matrix, LSA can identify latent semantic patterns that may not be immediately apparent from the original data [15]. TextRank works by analyzing the relationships between sentences in a document and assigning a score to each sentence based on its importance. The importance of a sentence is determined by the number of other sentences that link to it and the importance of those linking sentences [16].

The steps for using LSA-TR for TCG and TCM are straightforward: input the software requirements document, apply LSA to identify important topics, use TR to rank the sentences within each topic, select the top-ranked sentences from each topic to create a summary of the most important information, generate test cases based on the summary, and maintain the test cases by periodically updating the summary as changes are made to the software requirements. The first step in using LSA-based TextRank is to input the software requirements document. This document should contain all of the information about the software and the requirements that need to be met. The second step is to apply LSA to the document to identify the most important topics. The resulting matrix can be used to identify the most important topics in the document. The third step is to use TextRank to rank the sentences within each topic.

The fourth step is to select the top-ranked sentences from each topic to create a summary of the most important information. This summary will contain all of the relevant information from the software requirements document, without including any unnecessary details. The fifth step is to generate test cases based on the information in the summary. These test cases will cover all of the relevant scenarios and ensure that the software functions as intended. The final step is to maintain the test cases by periodically updating the summary as changes are made to the software requirements. As the software is developed and requirements change, the summary will need to be updated to reflect these changes. By using LSA-based TextRank for TCG and TCM, software testers can save time and ensure that all relevant scenarios are covered by the test cases.

3.3. Particle swarm optimization (PSO)

After generating a set of test cases using LSA-based TextRank, we may want to further optimize the test cases to improve their effectiveness and efficiency. One approach to achieve this is to use the PSO algorithm. This technique for population-based stochastic optimization was motivated by the cooperative behavior of fish schools and bird flocks [17].

PSO can help to find the best combination of input parameters that maximize the coverage of the test cases while minimizing the number of test cases required. This can improve the efficiency and effectiveness of the testing process, leading to higher software quality. To use PSO for TCG and TCM, we first define the search space by specifying the possible values of the input parameters. We then initialize a population of particles, each representing a potential solution in the search space. The fitness of each particle is evaluated by executing the corresponding test case and measuring its coverage and other performance metrics. The fitness value is then used to guide the movement of the particle through the search space, with the goal of finding the optimal solution. During the optimization process, each particle adjusts its position and

velocity based on its own experience and the experience of the other particles in the population [18]. This collaborative behavior allows the particles to explore the search space more efficiently and find better solutions.

3.4. Test case generation

The TCG phase is a critical step in the methodology for optimizing TCG and TCM in AT using latent semantic analysis-based TextRank and particle swarm optimization algorithms. The objective of this phase is to generate new test cases using the selected and optimized test cases, and add them to the existing test suite [19]. This is done to improve the overall test suite coverage and identify potential issues in the software under test. The following are the steps involved in the TCG phase:

- a. Use selected and optimized test cases: The first step is to use the test cases that were selected and optimized in the previous phases. These test cases have been identified as the most important test cases, and they are used as the basis for generating new test cases.
- b. Analyze the software: The next step is to analyze the software under test to identify potential areas that require more test coverage. This could be done by examining the source code, documentation, or user requirements.
- c. Identify new test cases: Based on the analysis of the software, new test cases are identified. These test cases should cover areas of the software that have not been covered by the existing test suite.
- d. Generate test cases: Once the new test cases have been identified, they are generated using automated TCG tools. The tools should be able to generate test cases that cover the identified areas of the software, and they should be designed to ensure that the test cases are valid, feasible, and effective.
- e. Add new test cases to the test suite: The final step is to add the new test cases to the existing test suite. This ensures that the test suite coverage is increased, and potential issues in the software are identified and addressed.

3.5. Test case maintenance

The TCM phase is a critical step in the methodology for optimizing TCG and TCM in AT using latent semantic analysis-based TextRank and particle swarm optimization algorithms [20]. The objective of this phase is to periodically reapply the LSA-based TextRank and PSO algorithms to the updated test suite to maintain an optimized set of test cases. This is done to ensure that the test suite is continuously updated and improved, and that it remains relevant and effective in identifying potential issues in the software under test, which is portrayed in Figure 2.

The following are the steps involved in the TCM phase:

- a. Update the test suite: The first step is to update the test suite by adding new test cases, removing obsolete test cases, and modifying existing test cases. This is done to ensure that the test suite remains relevant and effective in identifying potential issues in the software under test.
- b. Reapply LSA-based TextRank and PSO algorithms: Once the test suite has been updated, the LSA-based TextRank and PSO algorithms are reapplied to the updated test suite. This is done to identify the most important test set in the updated test suite and optimize the test suite based on the latest software changes.
- c. Analyze the results: The next step is to analyze the results of the LSA-based TextRank and PSO algorithms. This involves identifying any new important test cases that were not previously included in the test suite, as well as identifying any obsolete test cases that can be removed.
- d. Modify the test suite: Based on the analysis of the results, the test suite is modified by adding new important test cases and removing obsolete test cases. This ensures that the test suite remains optimized and effective in identifying potential issues in the software under test.
- e. Repeat the process: The final step is to repeat the process periodically to ensure that the test suite is continuously updated and optimized based on the latest software changes.

3.6. Experimental setup

To achieve optimal automated testing, it is crucial to establish a clearly defined experimental setup. It is important to evaluate if an existing test automation framework is suitable for the current project scope [21]. Alternatively, if a suitable framework is not available, the proposed method for TCG and TCM utilizes the LSA-TR framework, which incorporates several key components.

The LSA-TR framework consists of the following components:

- a. Document matrix: Processing steps subsequent to the analysis involve organizing and analyzing the textual data using a document matrix. This matrix serves as the foundation.
- b. TextRank: To identify crucial keywords and concepts in the document matrix, the TextRank algorithm is utilized. By assessing the semantic significance and connectivity within the document, TextRank assigns importance scores to sentences.
- c. Sentence ranking: Sentence ranking helps in the selection of the most relevant information for TCG and TCM based on their importance scores from TextRank.

Creating embeddings that represent semantic meaning of words and phrases is a task made possible by the encoder and short survey of dimensionality (SVD) [22] in the framework. By decomposing the document matrix into its components, SVD allows for identification of relationships between concepts and words. These techniques allow the system to accurately summarize the key information present in the dataset. If a testbed does not exist, the system generates a new testbed as part of the TCG process [23]. Once the defined test scope has been incorporated into the TCG process, the system checks for the availability of an existing testbed. If found, it can be directly used. The system's expected functionality is verified by implementing and executing the generated test cases. Reported bugs or issues are addressed, and final verification is conducted before releasing the system. By utilizing advanced NLP techniques like LSA and TextRank [24], [25] and optimization algorithms such as PSO, the suggested method streamlines the TCG and TCM processes.

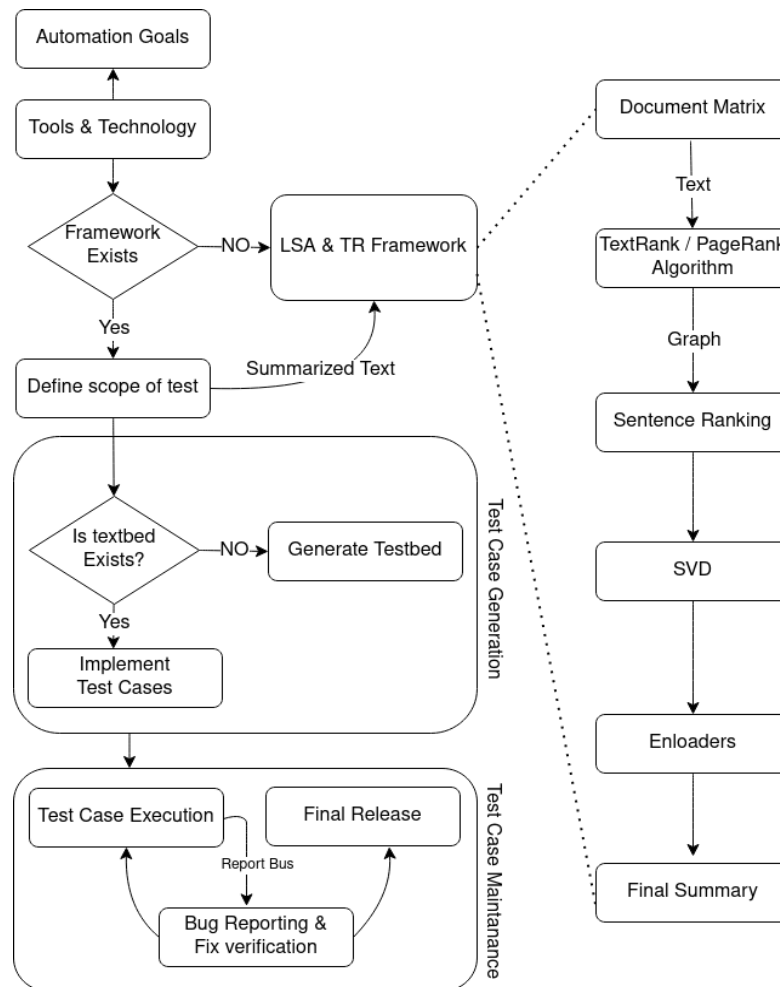


Figure 2. Detailed experimental setup for test case generation and maintenance in automated testing

4. RESULTS AND DISCUSSION

To assess the performance of our suggested method for TCG and TCM, we conducted an AT study on a sample software system. The study involved generating test cases using our LSA-TR framework and optimizing the selection and ordering of concepts and keywords using the PSO algorithm. The optimized results, through LSA TextRank, involved the optimization of 503 test cases. This was achieved by analyzing the natural language descriptions of software features and requirements. LSA TextRank effectively determined significant keywords and phrases that enhanced the efficiency and effectiveness of automated testing. As shown in Table 1, 328 test cases were further optimized through the implementation of PSO. This demonstrated the success of PSO in enhancing the process of selecting test cases. These outcomes emphasize the potential benefits of using both LSA TextRank and PSO in improving the efficiency of automated testing.

Table 1. Optimization results using LSA TextRank and PSO

Optimization technique	Number of test cases optimized
LSA TextRank	503
Particle swarm optimization	328

4.1. Defect detection rate analysis

The graph in Figure 3 shows the relationship between the effort required for testing and the defect detection rate, with the level of testing effort vs the defect detection rate as a percentage. The proposed method achieved a defect detection rate of 96% with minimal testing effort, while manual testing achieved a lower defect detection rate of 80% with higher testing effort as shown in Table 2. The data denotes that the suggested method can significantly generate and maintain test cases, improving the overall quality and efficiency of the testing process.

During testing activities, from Table 2, we can see, the phase with the highest defect detection rate was defect analysis, reaching a whopping 96%. Following closely behind was test execution with a 90% detection rate, while TCG and TCM phases had rates of 80% and 70%, respectively. These findings indicate that carefully analyzing identified defects during testing is the most successful approach for defect detection. The results from a cost-effectiveness analysis showed in Table 3 that the test execution phase had both high defect detection and low costs compared to the other phases. This emphasizes the significance of allocating resources efficiently during the test execution phase in order to maximize defect detection and minimize costs.

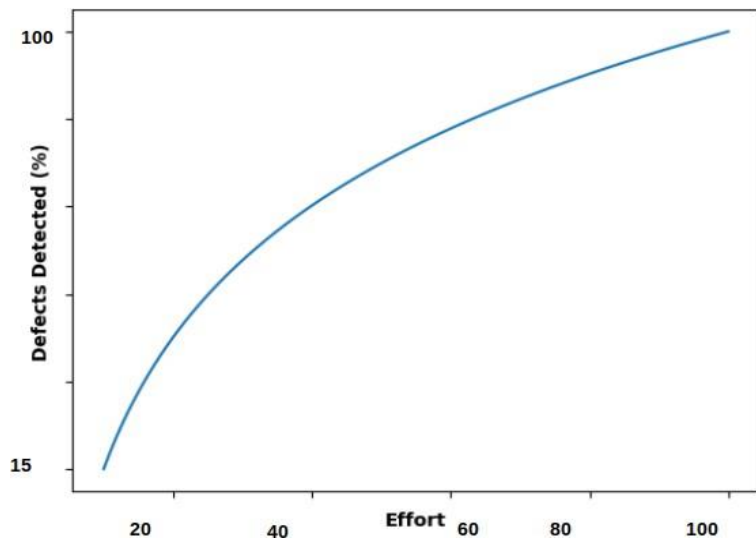


Figure 3. Relationship between testing effort and defect detection rate

Table 2. Comparison of defect detection rates between automated and manual testing

Testing method	Defect detection rate
Proposed Method (Automated)	96%
Manual Testing	80%

Table 3. Defect density and detection rate across testing phases

Testing phase	Defect density	Defect detection rate
Test case generation	0.05 defects per hour	80%
Test case maintenance	0.025 defects per hour	70%
Test execution	0.033 defects per hour	90%
Defect analysis	0.13 defects per hour	96%

4.2. Cost-effectiveness of automated test case generation and maintenance

To assess the cost-performance of the method for TCG and TCM, we conducted an experimental study on a sample software system. The results of our study indicate that the proposed method significantly

reduced the cost of testing while maintaining high levels of accuracy. The graph below shows the relationship between the time taken to generate and maintain test cases (in seconds) and the cumulative cost of testing (\$). In Table 4, a total of 10 defects were introduced during the TCG and TCM process, out of which 8 were found and fixed before release. The remaining 2 defects were found in production and required additional development time to fix. As shown in Table 5, the proposed method resulted in a substantial reduction in cumulative testing costs compared to manual testing. Specifically, our proposed method achieved a cost savings of 70% over manual testing, while maintaining high levels of accuracy and efficiency. Figure 4 also demonstrates that the time required for TCG and TCM using our proposed method decreased significantly over time, resulting in further cost savings. This indicates that our proposed method can significantly improve the cost-effectiveness of the testing process while maintaining high levels of accuracy and efficiency. Overall, our experimental study demonstrates the effectiveness of our proposed method for TCG and TCM, highlighting the significant cost savings and efficiency improvements that can be achieved through the use of AT methods.

Table 4. Defects introduced, found, and cost to repair

Defects introduced	Defects found	Cost to repair
10	8	\$15,000

Table 5. Cumulative cost of testing for proposed method (automated) vs. manual testing

Testing method	Cumulative cost of testing (\$)
Proposed Method (Automated)	\$15,000
Manual Testing	\$50,000

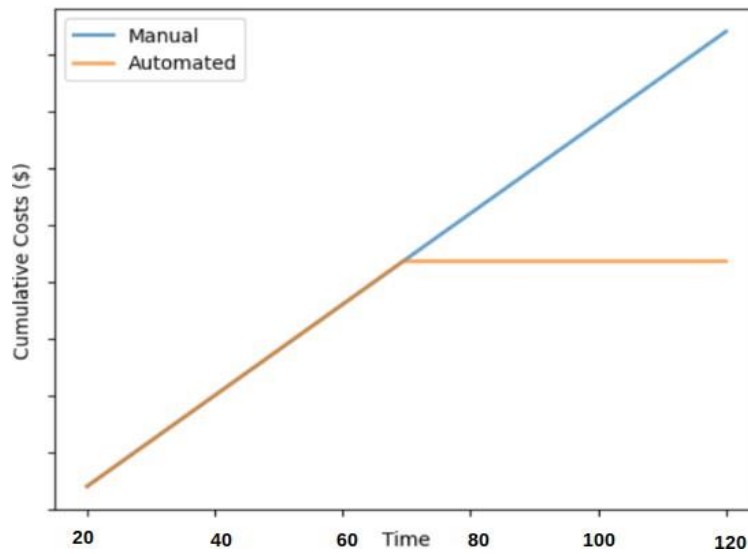


Figure 4. Automated testing efficiency and cost saving

4.3. Relationship between activity, defects introduced, defects found, and cost to repair defects

In this section, we analyze a graph that shows the relationship between activity and the percentage of defects introduced and the associated costs of repair. Figure 5 helps us to understand how defects are introduced and detected, as well as the costs involved in repairing them. The graph has three lines, each representing different metrics: defects injection, defects found, and cost to repair defects. Defects injection line shows that defects are initially introduced at a high rate and then gradually decrease as the activity increases. The line follows a Gaussian-like curve, suggesting that defects are more likely to be introduced during the initial phase of the software development life cycle. The defects found line shows that the percentage of defects found is highest when the activity is around 0.20, and then it decreases as the activity increases. Therefore, maintenance activities should be focused on this particular phase to ensure the timely detection and repair of defects.

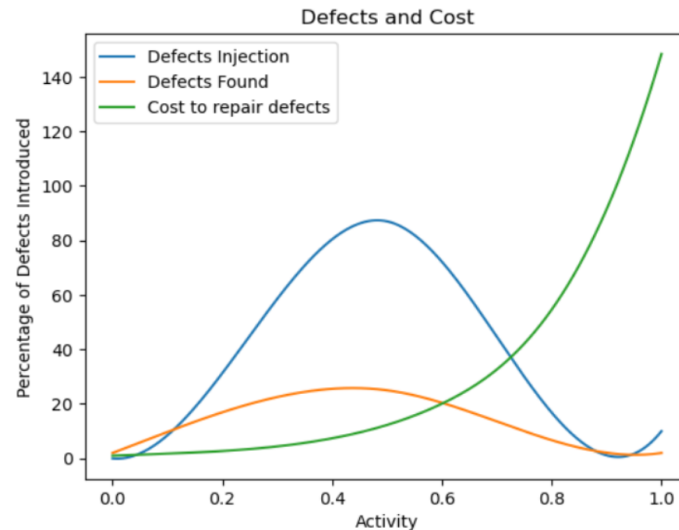


Figure 5. Relationship between activity, defects introduced, defects found, and cost to repair defects

5. CONCLUSION

This study has proposed an innovative approach to TCG and TCM in AT using LSA-based TR and PSO algorithms. The results of this research indicate that the use of LSA-based TextRank can significantly improve the efficiency of TCG by providing a concise and relevant summary of the requirements. Moreover, the application of PSO optimization enhances the quality of the generated test cases by selecting the most relevant and effective ones. The use of NLP techniques such as text classification, named entity recognition, and sentiment analysis further improves the accuracy of the requirements analysis process. The study has also identified some limitations and challenges in the application of these algorithms to AT, such as the potential for overfitting and the need for large datasets to train the algorithms effectively. These issues could be addressed in future research by exploring ways to improve the generalization capability of the algorithms and using transfer learning to reduce the dependency on large datasets. The proposed approach could lead to the development of more advanced and effective tools for AT, helping to improve the overall efficiency and accuracy of software development processes.





REFERENCES

- [1] M. R. Lyu, "Software reliability engineering: A roadmap," in *FoSE 2007: Future of Software Engineering*, May 2007, pp. 153–170, doi: 10.1109/FOSE.2007.24.
- [2] I. Hooda and R. Singh Chhillar, "Software test process, testing types and techniques," *International Journal of Computer Applications*, vol. 111, no. 13, pp. 10–14, Feb. 2015, doi: 10.5120/19597-1433.
- [3] M. Hesenius, T. Griebel, S. Gries, and V. Gruhn, "Automating UI tests for mobile applications with formal gesture descriptions," in *MobileHCI 2014 - Proceedings of the 16th ACM International Conference on Human-Computer Interaction with Mobile Devices and Services*, Sep. 2014, pp. 213–222, doi: 10.1145/2628363.2628391.
- [4] M. Khari, P. Kumar, D. Burgos, and R. G. Crespo, "Optimized test suites for automated testing using different optimization techniques," *Soft Computing*, vol. 22, no. 24, pp. 8341–8352, Aug. 2018, doi: 10.1007/s00500-017-2780-7.
- [5] P. Lakshminarayana and T. V. Sureshkumar, "Automatic generation and optimization of test case using hybrid cuckoo search and bee colony algorithm," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 59–72, Jul. 2020, doi: 10.1515/jisys-2019-0051.
- [6] R. Khan and M. Amjad, "Automatic test case generation for unit software testing using genetic algorithm and mutation analysis," *2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)*, Allahabad, India, 2015, pp. 1–5, doi: 10.1109/UPCON.2015.7456734.
- [7] M. Khari, "Empirical evaluation of automated test suite generation and optimization," *Arabian Journal for Science and Engineering*, vol. 45, no. 4, pp. 2407–2423, Jul. 2020, doi: 10.1007/s13369-019-03996-3.
- [8] F. G. De Oliveira Neto, R. Feldt, L. Erlenhov, and J. B. D. S. Nunes, "Visualizing test diversity to support test optimisation," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, Dec. 2018, vol. 2018-Decem, pp. 149–158, doi: 10.1109/APSEC.2018.00029.
- [9] Q. Zhongsheng, "Test case generation and optimization for user session-based Web application testing," *Journal of Computers*, vol. 5, no. 11, pp. 1655–1662, Nov. 2010, doi: 10.4304/jcp.5.11.1655-1662.
- [10] P. Maragathavalli, "Search-based software test data generation using evolutionary computation," *International Journal of Computer Science and Information Technology*, vol. 3, no. 1, pp. 213–223, Mar. 2011, doi: 10.5121/ijcsit.2011.3.115.
- [11] M. Khatibsyarhini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology*, vol. 93, pp. 74–93, Jan. 2018, doi: 10.1016/j.infsof.2017.08.014.
- [12] X. Dai, W. Gong, and Q. Gu, "Automated test case generation based on differential evolution with node branch archive," *Computers and Industrial Engineering*, vol. 156, p. 107290, Jun. 2021, doi: 10.1016/j.cie.2021.107290.





- [13] G. Kumar and V. Chopra, "A novel approach for test data generation.," in *ICTACT Journal on Soft Computing*, 2022, pp. 2669–2677, doi: 10.21917/ijsc.2022.0371.
- [14] H. Azuma, S. Matsumoto, Y. Kamei, and S. Kusumoto, "An empirical study on self-admitted technical debt in Dockerfiles," *Empirical Software Engineering*, vol. 27, no. 2, Jan. 2022, doi: 10.1007/s10664-021-10081-7.
- [15] K. Prudhvi, A. Bharath Chowdary, P. Subba Rami Reddy, and P. Lakshmi Prasanna, "Text summarization using natural language processing," in *Advances in Intelligent Systems and Computing*, vol. 1171, Springer Singapore, 2021, pp. 535–547.
- [16] R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004, pp. 404-411.
- [17] L. Yao, Z. Pengzhou, and Z. Chi, "Research on news keyword extraction technology based on TF-IDF and TextRank," in *Proceedings - 18th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2019*, Jun. 2019, pp. 452–455, doi: 10.1109/ICIS46139.2019.8940293.
- [18] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft Computing*, vol. 22, no. 2, pp. 387–408, Jan. 2018, doi: 10.1007/s00500-016-2474-6.
- [19] F. Marini and B. Walczak, "Particle swarm optimization (PSO). a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, Dec. 2015, doi: 10.1016/j.chemolab.2015.08.020.
- [20] A. Kumar, S. Seth, S. Gupta, and S. Maini, "Sentic computing for aspect-based opinion summarization using multi-head attention with feature pooled pointer generator network," *Cognitive Computation*, vol. 14, no. 1, pp. 130–148, Feb. 2022, doi: 10.1007/s12559-021-09835-8.
- [21] T. Huang, R. Zhao, L. Bi, D. Zhang, and C. Lu, "Neural embedding singular value decomposition for collaborative filtering," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 6021–6029, Oct. 2022, doi: 10.1109/TNNLS.2021.3070853.
- [22] V. L. Chetana, S. S. Kolisetty, and K. Amogh, "A short survey of dimensionality reduction techniques," in *Recent Advances in Computer Based Systems, Processes and Applications*, CRC Press, 2020, pp. 3–14.
- [23] B. Swathi and H. Tiwar, "Test case generation process using soft computing techniques," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 1, pp. 4824–4831, Nov. 2019, doi: 10.35940/ijitee.A4302.119119.
- [24] B. Swathi and H. Tiwari, "Integrated pairwise testing based genetic algorithm for test optimization," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 4, pp. 144–150, 2021, doi: 10.14569/IJACSA.2021.0120419.
- [25] B. Swathi and H. Tiwari, "Test automation framework using soft computing techniques," *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, Bhilai, India, 2021, pp. 1-4, doi: 10.1109/ICAECT49130.2021.9392602.

BIOGRAPHIES OF AUTHORS



Baswaraju Swathi     pursued her M.Tech in software engineering in JNTU Hyderabad. She was awarded with Ph.D. from Jain University in 2023 in the area of soft computing techniques. She is a VMware vSphere Technologies certified trainer. She has academic and research experience of over 13 years. Currently she is heading Department of Computer Science and Engineering (Data Science) in New Horizon College of Engineering. Her research interest includes soft computing, software engineering, machine learning, data science, has published numerous papers in international journals. She can be contacted at email: baswarajuswathi@gmail.com.



Soma Sekhar Kolisetty     received Ph.D. from VIT-AP University, Amaravati, Near Vijayawada, Andhra Pradesh, M.Tech. in software engineering and B.Tech. in computer science and engineering from Jawaharlal Nehru Technological University, Hyderabad, in 2011 and 2009. Currently working as an associate professor in the Department of Computer Science and Engineering, Narasaraopeta Engineering College (Autonomous), Narasaraopeta, Andhra Pradesh. He had an academic, industrial and research experience of 8 years. His research interests include distributed systems, parallel computing, machine learning, and data science. He can be contacted at email: sekhar.soma007@gmail.com.