A Systematic Literature Review on Test Case Minimization

Ritwik Dalmia[1], Soumili Chandra[2], Sudhir Kumar Mohapatra[3], Srinivas Prasad[4], Mesfin Abebe Haile[5]

[1,2,3] *Sri Sri University, Cuttack, India,*
[4] *GITAM University, Vishakhapatnam, India*
[5] *Adama Science and Technology University*

Abstract

Regression testing is an important testing technique. This testing is done after the implementation of the software. Test case minimization (TCM) or Test case reduction (TCR) is used in regression testing to minimize the test cases. The Main Purpose of TCM is to find a minimal set of test cases without compromising on the fault detection potential. This NP-hard technique does not save cost as well as time. The article gives qualitative findings for the articles that were surveyed. Furthermore, it multiplies the results with scientific impeachments and significant results from the fundamental sources of information literacy.

Keywords

SLR, Test case minimization, test case reduction, Regression testing.

1. Introduction

SLR (Systematic Literature Review) evaluates and analyses all potential connected activities to our research questions. Kitchenham and Charters [1] based their SLR regulations on the SLR requirements., SLR has been used to achieve the following goal:

To identify the chief rituals and practices by using exact technologies, procedures, tools, or processes by accumulating information from other related modules. To summarize, identify, evaluate, and interpret all available research gaps about test suite minimization. In this study, the systematic literature review is used as the review methodology.

There are some reasons why selecting a systematic literature review as a review methodology [1].

1) SLR can define search strategies; this ensures the completeness of the primary studies to be assessed in the review.
2) It has well-defined primary study inclusion and exclusion criteria which allows filtering out more relevant documents for the review,
3) Since every process of the review is documented, it is easy to understand for the readers,
4) It is the well-defined methodology that makes the result of the review less likely biased and reporting the report that supports the review question.

2. The Need for Conducting a Systematic Literature Review

The need for this systematic literature review is to extract some existing information about test suite minimization.

There are different reasons to conduct SLR among those reasons some are as follows:

- ✓ To extract information about test suites minimization,
- ✓ To extract information about the coverage information used for minimization of the test suite,
- ✓ To know how the previous studies minimize the test suite,

ACI'22: Workshop on Advances in Computation Intelligence, its Concepts &; Applications at ISIC 2022, May 17-19, Savannah, United States
EMAIL: ritwik.d2020bds@srisriuniversity.edu.in (A. 1); soumili.c2020bcs@srisriuniversity.edu.in (A. 2); 3 sudhir.mohapatra@srisriuniversity.edu.in (A. 3); sprasad@gitam.edu (A. 4); mesfinabha@gmail.com (A. 5)
ORCID: 0000-0003-3065-3881 (A. 3)

- ✓ To apply different inclusion and exclusion criteria,
- ✓ To apply different search strategies to get sufficient information.

3. Review Question

RQ1: What are the existing techniques for minimization of the test suite?

RQ2: How the existing techniques minimize the test suites?

RQ3: What are frequently used coverage information for test suite minimization?

RQ4: What metrics are used by researchers to measure the experiments in test minimization?

4. Conducting Review

This section contains two main parts which are included as follows:

4.1 Generating Search Strategies

The systematic literature review uses properly defined search keywords to explore the dominant subjects in selected search assets.

4.2 Search Keyword

The coverage of this systematic literature review includes studies related test suite minimization in software testing. To find out the primary studies related to test suite minimization the following keywords are used
{((‘regression’ AND ‘test’) AND (‘suite’ OR ‘case’) AND (minimization OR reduction) OR (algorithm OR technique OR approach OR heuristic)
OR (empirical study OR experiment OR experimental study)) <in title, abstract, and keywords>}

4.3 Search Result:

By using the above keywords 120 papers related to test suite minimization are downloaded which are published in between 2017 to 2021. Then papers are filtered and selected based on the inclusion and exclusion criteria defined. The primary studies searched for this systematic review is in the following database: IEEE Xplore, ACM Digital library, Science Direct, Springer Library, Elsevier Online Library, Google Scholar and the references found on the primary studies.

4.4 Documenting the search process

Table 1: Documenting the search process

Data Source	Documentation
Digital Library, Journals Hand Searches and Conference proceedings	Name of Database: IEEE xplore Search Strategy for the database: Using search key words Date of search: June 10, 2021 Years covered: 2017 – 2021 Name of Database: ACM Digital library Search Strategy for the database: Using search key words Date of search: June 11, 2021 Years covered: 2017 – 2021

Name of Database: Science Direct
Search Strategy for the database: Using search key words
Date of search: June 13, 2021
Years covered: 2017 – 2021

Name of Database: Springer Library Search Strategy for the database: Using search key words
Date of search: June 13, 2021
Years covered: 2017 – 2021

Name of Database: Elsevier Online Library,
Search Strategy for the database: Using search key words
Date of search: June 14, 2021
Years covered: 2017 – 2021

5. Study Selection Criteria

The inclusion and exclusion standards for the primary research of related works are described in this section..

Inclusion criteria: The primary studies evaluated are supposed to respond to the research questions, and those that meet the following criteria are included in SLR:

IC1: Between 2017 and 2021, studies must be published.
IC2: The results of the study must be published in the conference proceedings..
IC3: Studies must be published in journal issues.
IC4: Studies focused on Software Testing, Regression testing and Test optimization.
IC5: Studies focused on Test suite minimization
IC6: Studies focused Test suite reduction techniques and algorithms
IC7: Studies that can provide answers to one or more of our research or review inquiries.

Exclusion criteria: If one of the following criteria is met, the downloaded documents are excluded.

EC1: Studies unrelated to Test suite minimization
EC2: Duplicate studies with similar findings
EC3: Studies that aren't a full paper, a short paper, a master's thesis, or a doctoral (Ph.D.) dissertation.
EC4: Studies that are not written in English

6. Primary Study Selection

Table 2: Selected primary studies

Study No.	Ref no.	Publication Year	Study Focus	RQ1	RQ2	RQ3	RQ4
PS1	[2]	2019	The study focus on minimizing the size test suite and execution time using a search based technique Ant Colony Optimization and uses statement coverage as a coverage information.	✓	✓	✓	✓
PS2	[3]	2020	Using the search-based NSGA-II method, the work concentrates on a multi-objective test suite optimization that is code and mutation coverage based for developing a representative	✓	✓	✓	✓

			set of test cases capable of both identifying and locating problems.				
PS3	[4]	2019	The work focuses on fault coverage-based test suite minimization utilizing greedy "learning from mistakes," which shortens the Time of execution and decreases test suite size.	✓	✓	✓	✓
PS4	[5]	2020	The "REDUNET" network-based optimization method and integer linear programming are the main topics of the study design an integrated framework for the automatic development of an effective and efficient test suite that combines test suite generation, code coverage analysis, and test suite reduction. To decrease a test suite while preserving the same code coverage.	✓	✓	✓	✓
PS5	[6]	2018	The study focuses on creating and minimizing test cases based on maximal path cover-age using an artificial bee colony algorithm or the cuckoo search method.	✓	✓	✓	✓
PS6	[7]	2017	The study focused on test case minimization using a flower pollination technique, in an effort to shorten the time needed for a single run and the size of the finished test suite.	✓	✓	✓	✓
PS7	[8]	2019	The study focuses on combining the test requirements set, extracting test cases from each cluster, and locating as many related test cases as feasible using an enhanced K-means algorithm, as well as using the Degree of Membership Function to build a fuzzy clustering approach.	✓	✓	✓	✓
PS8	[9]	2018	The study focuses on eliminating redundancies at the test statement-level using fine-grained test case minimization while maintaining test assertions and suite coverage.	✓	✓	✓	✓
PS9	[10]	2020	The study focused on fault coverage-based test suite optimization utilizing the butterfly optimization algorithm, with fault detection as a performance measure.	✓	✓	✓	✓
PS10	[11]	2017	Based on a reduction in test suite size and execution and validation costs, the study suggests the diversity dragonfly approach for cost-aware test suite reductions.	✓	✓	✓	✓

7. Reporting the Systematic Literature Review Result

This section describes the review questions listed in the section 3.

RQ1: What are the existing techniques for minimization of the test suite?

Neha Gupta et al. [3] presented a test suite reduction method based on the NSGA-II algorithm. The proposed method sought to develop a small test suite that could discover and localize software flaws. Finally, they stated that their approach provided a basic set of test suites with a 78 % reduction in size and 95.16 % fault detection using test suites from the Defects4j repository.

Arun Prakash et al. [4] a fault coverage-based test suite optimization (FCBTSO) method was developed which reached from on Harrolds–Gupta–Soffa (HGS) and learning from mistakes approach. Their goal was to maximize fault coverage while keeping the test suite to a minimum. They stated that their technique was more effective than the Greedy method, HGS, Additional Greedy, and En-hanced HGS fewer test cases are being used.
Misael Mongiovì et al. [5] REDUNET is a test suite reduction method that combines integer linear programming with network-based optimization. With the objective of reducing the test suite while ensuring equivalent code coverage, using test suite generation, code coverage analysis, and test case reduction, they developed a comprehensive framework for the automatic creation of an effective and efficient test suite. They claimed that their method resulted in a 50% the test suite's size was decreased.
Manju Khari et al. [6] described the creation of an automated testing solution that involves the production and reduction of test suites. For lowering the test suite, they developed the artificial bee colony algorithm or the cuckoo search approach. Their main goal was to reduce the test suite while increasing path coverage.

Mohapatra et al. [2] For test suite reduction, an ant colony optimization approach was devised. Their objective was to construct a realistic collection of test cases that covered all of the requirements while keeping the original test suite's fault detection capability and executing in the shortest possible time.

AbdulRahman et al [7] for the problem of test suite reduction, a suggested Test Generator Flower Pollination Strategy based on the Flower Pollination Algorithm is presented. By removing pointless test cases, they hoped to reduce the cost of software testing. They claimed that their approach outperformed existing algorithms and that it is also simple to build, has fewer parameters, and is more flexible.

Bao-Sheng et al [8] For test suite reduction, an improved K-Means approach was developed. They stated that their technique minimizes the number of redundant test cases while simultaneously providing the largest coverage and is more effective and efficient.

Arash et al [9] proposed fine-grained test suite minimization method. Their method uses the inference of a test cases model to analyze data, allowing for automated test re-organization inside a test suite. Their technique eliminates redundancy at the test statement level while maintaining test assertions and testing suite coverage. The researchers claim that the method was able to eliminate 43 % of duplicated test cases and 20% of execution time on average.

Abhishek et al. [10] proposed an efficient self-adaptive butterfly optimization algorithm-based approach for test suite minimization. When compared to the bat search method, they found that their approach performed better in terms of fault detection.

Shounak et al. [11] developed a diversity dragonfly-based method for cost-conscious test suite reduction. The test suite's quality and cost were their main concerns. They claimed that the DDF's reduction capabilities are superior to previous approaches and that the cost is likewise inexpensive, ensuring a high-quality test suite reduction.

RQ2: How the existing techniques minimize the test suites?

Neha Gupta et al. [3] used the NSGA-II algorithm. Setting NSGA-II parameters was the first step. The first step was to set the NSGA-II parameters. They fed the algorithm a test suite in the form of a bit string as input. The chromosomal size is the same as the test suite size. The suite's number of solutions was proportional to the population size. They employed ramping half-and-half initialization strategies to populate the population. They limited the number of generations based on the amount of input. A half uniform crossover was utilized for the crossover function, and a bit-flip

with a probability of Pm 1/n was employed for the mutation function. Finally, they established two criteria for determining when to cease. The first was the number of generations, and the second was the threshold value for each input.

Arun Prakash et al. [4] created a fault coverage-based test suite optimization technique. Their algorithm's initial step is to determine the fault weight (FW) for each fault type f_i This brings the total number of tests that cover each defected i^{th} fault. Their approaches choose a test case from a related cardinality test suite which covers the most flaws (or undetected faults) in step 2. If two test cases address the same defects, the selection method can choose one at random. The overall number of issues/faults fi that each test case's coverage covered (Step 3) and the test case weight were also determined (TCW). In step 4, a loop was executed until all coverable faults (FS) or all test cases in the suite had been performed.

Misael Mongiovì et al. [5] To make use of the advantages of the control flow graph, they used integer linear programming and network-based optimization, and they identified test suite minimization as a minimal set coverage issue. The following are the steps to obtain the reduced suite: Step 1: For the program under test, the Randoop tool was used to construct a test suite. Step 2: The produced test suite was put as Java unit tests into the maven project. Step 3: The test cases are run with a service tracing code coverage tool. Step 4: The method coverage was calculated when executing the test cases. Finally, in Step 5 for test suite reduction, the test execution traces and control flow graph are provided, resulting in a smaller test suite with the same code coverage and a faster execution time.

Manju Khari et al. [6] provided a strategy for generating test data using black-box testing techniques, then optimizing the data using an Cuckoo search or artificial bee colony algorithms.

The steps in the artificial bee colony algorithm are as follows: The first step is to set up the ABC's control settings. Step 2 involved employing black-box testing methodologies to create the initial population. Step 3 involves analyzing the population and adding test cases to the result set that follow new independent routes. The fourth step is to make the cycle equal 1. Step 5 is based on the employed bee and observer bee phases and performs the following tasks: locating a different neighbor test case than ai, assessing a'i as a food supply, and replacing ai with a'i if it is superior to ai. Increase the test counter if there is no way to make the solution better.

To replace any test cases with test counters higher than the stopping criterion value, use the Scout Bee Phase to find the faulty test cases and replace them with newer, random test cases. Then the cycle is equal to cycle plus one. Finally, repeat all of the previous stages in Step 6 of the While cycle! =stopping criterion.

Steps in the Cuckoo Search Algorithm Step 1: Set up the CSA's control parameters and stopping criteria. Step 2: Using any black-box approach, start the initial population. Step 3 Analyze the sample and include test cases that take fresh, separate routes in the outcome set. In step 4, set cycle = 1. Do the following procedures in Step 5 Cuckoo Phase: c'i, obtain a random cuckoo with an egg. Check the self-reliant path the test case follows by going to a random nest.

If the new egg (test case) c'_i is superior to the old egg (test case) c_i in the nest, c'i will replace ci if a test case for that independent path already exists in the result set from the Egg Replace/Drop Phase. Otherwise, find a new nest. Cycle + 1 is the make cycle. Finally, while cycling, step 6! = If the stopping criteria are met, go to the next stage.

Mohapatra et al.[2] presented an ant colony-based approach for test suite reduction. The test cases are represented as nodes in a full graph in their method. The execution duration of each test case and linked test criteria are stored in a matrix. In a full network, a node is a location where ants begin their search. The ant uses the matrix to help choose neighbor nodes based on reducing execution time and

maximizing needs. The test suite is used as the foundation for creating a comprehensive graph, according to their core concept. The test cases represent each vertex in the graph. There are at least as many ants as there are test cases. The ant in the whole vertex will be the starting point for the solution. Each ant adds additional edges to its existing path in order to find the optimum path. The find next() method is used to create a new route. This function looks for nearby edges with the most phero-mone deposits. If there is a tie between edges, it chooses one at random. When there are no more edges remaining for the path, the process of adding edges comes to an end.

AbdulRahman et al [7] Once the input parameters, such as the parameter number p and the V set of values for each feature V = [v0..vj], have been specified is the initial stage. Let the final test case list be a collection of candidate tests, and then construct all potential interactions elements using IEL based on P and V, as well as a random pollen population. If rand is smaller than pa, Create a step vector L that complies with the Levy distribution and compute global pollination by $x_t^{i+1} = x_t^i + \Upsilon L(\lambda)(g^* - x_t^i)$ while the IEL is not empty and the loop has not achieved maximum generation. If not, choose a random seed from a uniform distribution in the range [0, 1], carry out local pollination using the equation $x_t^{i+1} = x_t^i + \in (x_t^i - x_t^k)$, and then stop. Update the population with the new solutions if they are superior, and then call it a day. G* end is the most effective solution at the moment and adds g* to the Final Test Case List as the best test case. End while, end-procedure, and remove covered interaction items from IEL.

Bao-Sheng et al [8] For test suite reduction, an improved K-Means approach was developed. They refined the K-Means methodology and created a fuzzy clustering approach by integrating the test criteria, selecting test cases from each cluster, and locating as many related test cases as possible using the Degree of Membership Function. Assume that in this method, the data sets consist of K categories. T = t1, t2, and tn is set by the software test case, and that mi is the cluster center of cluster I i[1, K]. Derive the fuzzy K-Means membership functions to achieve the optimal answer for the fuzzy K-Means algorithm.
Arash et al [9] presented a fine-grained test suite minimization approach Instead of depending on code coverage, their methodology captures the true behavior of test cases by documenting calls to production methods and their inputs, keeping track of cover-age and defect detection, and keeping track of all test assertions in a test suite. Instrumenting code and building models are the two fundamental phases. They used code instrumentation to store and study the nature and value of every referential variable so each test statement level are desired test state must be accomplished. Additionally, variables are used and defined at this stage. Additionally, determining Equivalent Test assertions and compatible states is a part of the second model generation stage. Two minimization algorithms are then put into practice. Test composition and test suite reorganization.

By rearranging the test suite, they were able to determine the quickest route to the closest test statement that was yet undiscovered. Starting from the beginning state. They repeated this technique from that node until there was no longer any way to extend the route to encompass additional equivalent test statements or assertions. This event occurs when all of the model's relevant test statements and declarations are covered.
Else, start from the beginning and repeat the method if there are any similar test statements or assertions that are still undiscovered. They employed a variation of the best-fit search technique that keeps running in an effort to find the quickest path.

Then To store the state, the Composing Minimized Test Cases method employs a bidirectional map from variable names and types to variable values. As it moves through the test statements in the rearranged test suite path, it checks to see if each test statement contains the value for each variable. If a value of this kind exists but is known by a different name in the state, it will rename the variable present inside the test expression. If the target type and the variable type are different, it is cast to the target type. If there are any names that are duplicated, they are renamed. Finally, it changes the bi-directional state map with the test statement's modified values, signaling that the x call with inputs set of test cases is now running.

Abhishek et al. [10] presented a butterfly optimization-based strategy for test suite reduction. The objective function f(s), where s=(s1, s2,..., s_{dim}), and dim is the number of dimensions, must first be defined. Si=(i=1,2,3,4,....n) is the second generation of the initial population of n butterflies. Add the input parameters a, p, and c after that.
Third, if the stopping conditions are not satisfied, the following activities are performed for each butterfly bfy: compute fragrance using $bfr_i=Sc\ (SI)^a$, where bfr_i is the magnitude of fragrance and Sc, SI, and a parameter are a modality dependent. Then pick the best bfy. For each bfy in the population, generate a random probability s ranging from 0 to 1. The fourth step is to check if s_p, and then proceed to the best bfy for global search by using $s_i^{t+1} = s_i^t + (r2 \times bg^* - s_i^t) \times bfr_i$ where sit is the solution vector s_i for i_{th} butterfly in iteration number t and bg* considering all solutions in the current stage, this shows the best-known solution for the current condition. Ith butterfly's fragrance is characterized by bfr_i and r as a random value between 0 and 1. If s>p move at random and execute a local search $s_i^{t+1} = s_i^t + (r^2 \times s_k^t - s_i^t) \times bfr_i$ where s_i^t and s_k^t are j_t and k_{th} butterflies chosen at random from solution space. After that, the newly developed solution is reviewed, and the better solution is updated. Finally, alter the value of a parameter. The best solution was found.

Shounak et al. [11] proposed diversity dragonfly algorithm-based approach for minimization of test suite. The first step is initializing the input parameters and population. The second step is updating principal parameters of the dragonfly algorithm. Thirdly, it is about computing the objective function. Then update the position. The final step is injecting of the diversity factor.

RQ3: What are frequently used the coverage information for test suite minimization?

For test suite reduction, many forms of coverage information are utilized in the literature
. **Neha Gupta et al**. [3] employed diversity conscious mutation adequacy criteria fault finding, statement coverage, and branch coverage. **Arun Prakash et al.** [4] used fault-coverage. **Misael Mongiovì et al.** [5] used statement coverage (code and method coverage). **Manju Khari et al.** [6] used path coverage. **Mohapatra et al.** [2] used statement coverage. **AbdulRahman et al** [7] used test requirement. **Bao-Sheng et al** [8] used test requirement set. **Arash et al** [9] used statement and path coverage. **Abhishek et al.** [10] used fault coverage. **Shounak et al.** [11] used branch coverage.

RQ4: What metrics are used by researchers to measure the experiments in test minimization?

In the literature different metrics are used to measure the experiments in test minimization.

Neha Gupta et al. [3] percentage of test suite size reduction, percentage of fault discovered, and fault localization score were all employed.
Arun Prakash et al. [4] made use of the optimized test suite's size, execution speed, and coverage of fault
Misael Mongiovì et al. [5] utilized the Percentage of suite size and test execution time improvement.
Manju Khari et al. [6] used path coverage and fitness values.
Mohapatra et al. [2] used Sizes of representative sets, and scalability.
AbdulRahman et al. [7] used the size of the final test suite and time for one execution. **Bao-Sheng et al** [8] The rate of simplification, the efficacy of fault detection, and the rate of fault detection loss were all employed.
Arash et al. [9] used Execution time, Code Coverage, and Fault Detection capability. **Abhishek et al.** [10] used fault detection capability.
Shounak et al. [11] utilized the size and expense of the test suites needed for execution and validation.

8. Related Work

This section cover related works on test suite minimization techniques. The related work result

described as follows.

Qing et al. [12] For wearable embedded software, a novel test-suite reduction methodology based on the subtraction operation (TSRSO) was devised. The Quine-McCluskey (Q-M) algorithm was created by Quine and McCluskey is employed to simplify a Boolean function algebraically. They performed a matrix column transformation procedure to remove redundant criteria based on interrelationships among testing requirements, and a modification to the row transformation of the matrix to reduce the test suite in their suggested model.

They compared their model's reduction capabilities to the outcomes of GRE, GE, and H with the aim of providing broad guidance to testers in choosing the best strategy for test suite building. According to their study, the minimal reduced test sets produced by TSRSO are smaller than those created by GE, GRE, or H, and the performance of H, GE, and GRE is quite similar.

Reena et al. [13] Class partitioning and a Genetic Algorithm were used to produce a model for test case reduction in a component-based system. For the construction of the test suite, fitness values were maximized in their model. To boost the performance of the genetic algorithm, they enlarged the search space and included fitness scaling. By contrasting the old genetic algorithm with the new improved genetic algorithm, they determined which genetic algorithm produced the best fitness values at constant mutation and crossover rates). They observed that when the two algorithms were compared, the modified genetic algorithm produced superior results than the standard GA method.

Shaima et al. [14] A method called Deterministic Test Suite Reduction (DTSR) was created using Hyper graph Minimal Transversal Mining. They chose the candidate test cases on the basis of hypergraph structural information. The requirement data enhanced the selected test cases by retaining a deterministic set. For achieving the purpose hypergraph was considered as a test suite by DTSR, where its hyperedges were analogous to requirements and its nodes were equivalent to tests By selecting the fewest possible number of test cases that satisfy the criteria, a subset of a hypergraph's minimal transversals was retrieved. They compared their approach with search-based algorithms and based on their report, The outcomes indicated that their algorithm was superior and the reduction rate of their approach differs from 50% up to 65% of the initial range of the set.

Shounak et al. [15] created a cost-aware test suite reduction method using the Greedy Search Algorithm with TAP measurement (GTAP). To determine the importance of test cases, TAP measure was specially developed. Two more elements were added in their approach: the quantity of test instances that might fulfil improved test requirements and the determination of the test suite's most crucial test cases. Additionally, it generated a realistic collection of test scenarios at the lowest possible cost. They used eleven subject programs available in the SIR repository to examine their algorithms. They used reduction capability and relative capability to evaluate the effectiveness of their and the existing algorithm. They stated the fact that the DIV-GA acquired 90.27% which is less than the average performance of their algorithm in all the programs is 93.07%. As a result, they achieved better outcomes.

Shilpi et al. [16] developed a similarity-based greedy approach for producing an effective number of test cases. Their approach was a combination of two regression testing activities which are minimization and prioritization. Their main strategy is to first analyze the test cases to determine the test case pairings' difference and similarity values and then to optimize the test cases using a greedy and clustering technique. They employed two algorithms Test Case Coverage Analyzer (TCCA) and Similarity-Based Greedy Algorithm (SBGA)for optimizing the test suite. They compared their experimental results to Harrold Gupta and Soffa's (HGS) prominent heuristic, considering the minimal test suite size and fault coverage testing requirements. They claimed that the result of their method was fairly successful in terms of defect detection, a reduced test suite without having a significant impact on the percentage of suite size reduction.

9. Conclusion

Our literature review showed that different researcher use different techniques to find out minimized test suite and different minimization techniques resulted in different outputs; the output may depend on the input data for the algorithm. In minimization of test suite many researchers use different metaheuristic optimization algorithm. Most of the developed tools are not adopted by the software testing industry yet according to our review.

10. References

[1] BA, Kitchenham and Charters, Stuart, "Guidelines for performing Systematic Literature Reviews in Software Engineering," vol. 2, 2007

[2] S. Mohanty, S. K. Mohapatra, and S. F. Meko, "Ant colony Optimization (ACO-Min) algorithm for test suite minimization," *Advances in Intelligent Systems and Computing*, pp. 55–63, 2020.

[3] N. Gupta, A. Sharma, and M. K. Pachariya, "Multi-objective test suite optimization for detection and localization of software faults," *Journal of King Saud University - Computer and Information Sciences*, 2020.

[4] P. Agrawal, A. Choudhary, A. Kaur, and H. M. Pandey, "Fault coverage-based test suite optimization method for regression testing: Learning from mistakes-based approach," *Neural Computing and Applications*, vol. 32, no. 12, pp. 7769–7784, 2019.

[5] M. Mongiovì, A. Fornaia, and E. Tramontana, "Redunet: Reducing test suites by integrating set cover and network-based optimization," *Applied Network Science*, vol. 5, no. 1, 2020.

[6] 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, 2017.

[7] R. A. Alsewari, H. C. Har, A. A. Homaid, A. B. Nasser, K. Z. Zamli, and N. M. Tairan, "Test cases minimization strategy based on flower pollination algorithm," *Recent Trends in Information and Communication Technology*, pp. 505–512, 2017.

[8] T. Tan, B. Wang, Y. Tang and x. Zhou, "An Improved K-means Algorithm for Test Case Optimization," 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), 2019, pp. 169-172, doi: 10.1109/CCOMS.2019.8821687.

[9] X. Wang, S. Jiang, P. Gao, x. Ju, R. Wang, and Y. Zhang, "Cost-effective testing based fault localization with distance based test-suite reduction," *Science China Information Sciences*, vol. 60, no. 9, 2017.

[10] S. Verma, A. Choudhary and S. Tiwari, "Test Case Optimization using Butterfly Optimization Algorithm," 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2020, pp. 704-709, doi: 10.1109/Confluence47617.2020.9058334.

[11] S. R. Sugave, S. H. Patil and B. E. Reddy, "DDF: Diversity Dragonfly Algorithm for cost-aware test suite minimization approach for software testing," 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), 2017, pp. 701-707, doi: 10.1109/ICCONS.2017.8250554.

[12] Q. Shen, Y. Jiang, and J. Lou, "A new test suite reduction method for wearable embedded software," *Computers & Electrical Engineering*, vol. 61, pp. 116–125, 2017.

[13] Reena and P. K. Bhatia, "Test case minimization in cots methodology using genetic algorithm: A modified approach," *Proceedings of ICETIT 2019*, pp. 219–228, 2019.

[14] S. Trabelsi, M. T. Bennani, and S. B. Yahia, "A new test suite reduction approach based on hypergraph minimal transversal mining," *Future Data and Security Engineering*, pp. 15–30, 2019.

[15] S. R. Sugave, S. H. Patil, and B. E. Reddy, "Ddf: Diversity dragonfly algorithm for cost-aware test suite minimization approach for software testing," *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2017.

[16] S. Singh and R. Shree, "A new similarity-based greedy approach for generating effective test suite," *International Journal of Intelligent Engineering and Systems*, vol. 11, no. 6, pp. 1–10, 2018.

[17] S. Verma, A. Choudhary and S. Tiwari, "Test Case Optimization using Butterfly Optimization Algorithm," 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2020, pp. 704-709, doi: 10.1109/Confluence47617.2020.9058334.

[18] S. R. Sugave, S. H. Patil and B. E. Reddy, "DDF: Diversity Dragonfly Algorithm for cost-aware test suite minimization approach for software testing," 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), 2017, pp. 701-707, doi: 10.1109/ICCONS.2017.8250554.

[19] Sugave, ShounakRushikesh, SuhasHaribhau Patil, and B. Eswara Reddy. "A Cost-Aware Test Suite Minimization Approach Using TAP Measure and Greedy Search Algorithm." International Journal of Intelligent Engineering and Systems 10.4 (2017): 60-69.
[20] Singh, Shilpi, and Raj Shree. "A new similarity-based greedy approach for generating effective test suite." International Journal of Intelligent Engineering and Systems 11.6 (2018): 1-10.