Received May 27, 2019, accepted June 20, 2019, date of publication July 2, 2019, date of current version July 22, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2926384

A Comprehensive Investigation of Modern Test Suite Optimization Trends, Tools and Techniques

AYESHA KIRAN, WASI HAIDER BUTT, MUHAMMAD WASEEM ANWAR, FAROOQUE AZAM, AND BILAL MAQBOOL
Department of Computer & Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan

Corresponding author: Muhammad Waseem Anwar (waseemanwar@ceme.nust.edu.pk)

ABSTRACT Software testing is an important but expensive activity of software development life cycle, as it accounts for more than 52% of entire development cost. Testing requires the execution of all possible test cases in order to find the defects in the software. Therefore, different test suite optimization approaches like the genetic algorithm and the greedy algorithm, etc., are widely used to select the representative test suite without compromising the effectiveness. Test suite optimization is frequently researched to enhance its competences but there is no study published until now that analyzes the latest developments from 2016 to 2019. Hence, in this article, we systematically examine the state-of-the-art optimizations' approaches, tools, and supporting platforms. Principally, we conducted a systematic literature review (SLR) to inspect and examine 58 selected studies that are published during 2016–2019. Subsequently, the selected researches are grouped into five main categories, i.e., greedy algorithm (seven studies), meta-heuristic (28 studies), hybrid (six studies), clustering (five studies), and general (12 studies). Finally, 32 leading tools have been presented, i.e., existing tools (25 tools) and proposed/developed tools (seven tools) along 14 platform supports. Furthermore, it is noted that several approaches aim at solving the single-objective optimization problem. Therefore, researchers should focus on dealing with the multi-objective problem, as multi-objective versions outperform the single-objective ones. Moreover, less attention has been given to clustering-based techniques. Thus, we recommend exploring the machine learning and artificial intelligence-based optimization approaches in the future. A broad exploration of tools and techniques, in this article, will help researchers, practitioners, and developers to opt for adequate techniques, tools, or platforms as per requirements.

INDEX TERMS Software testing, test suite optimization, single objective optimization, multi-objective optimization.

I. INTRODUCTION

Software testing is one of the most important activities of the software development life cycle. It is done to strengthen the quality of the product before delivering it to the client [1]. However, in spite of that, software testing is high-priced. More than fifty-two percent (>52%) of entire development cost is accounted for it. Hence, it is necessary to control the cost of testing process as much as possible because of its monotonous and time-consuming nature [2]. In software testing, optimization of test data can be done to control this cost [3]. Because when new test cases are added to the existing test suite for checking enhanced functionality, [4] some of the test cases become obsolescent which simply result in a waste of time, money and resources in the testing phase. However, the purpose of software testing cannot be accomplished by simply decreasing the test data. All the probable lapses existing in product or software must be adequately uncovered by the test data. Amongst several available solutions, the procedure of finding the optimum solution is regarded as optimization [5]. Fittest dataset can be filtered out with the aid of optimization and then it can be used to test different software or product related properties.

Until now, many research efforts have been devoted to test suite optimization. As a result, various algorithms/techniques for test suite optimization have been introduced intensively. These algorithms include: ACO (Ant Colony Algorithm) [6], genetic algorithm [7] and its variants like NSGA

The associate editor coordinating the review of this manuscript and approving it for publication was Hui Liu.

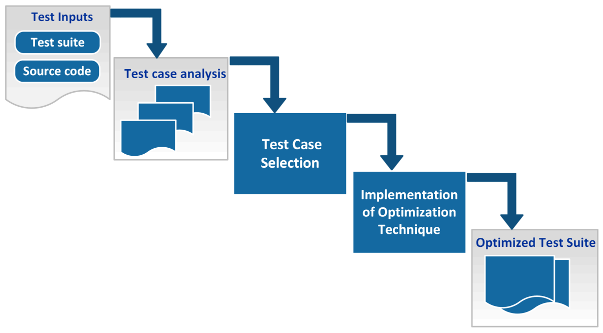

FIGURE 1. Major test suite optimization activities.

(Non-Dominated Sorting Genetic Algorithm) [8], PSO (Particle Swarm Optimization) [9], greedy algorithm [10], and clustering based optimization approaches e.g. hierarchical agglomerative clustering [11], fuzzy c mean clustering [12] etc. In literature, mostly different meta-heuristic algorithms are used to carry out the optimization of test data. Two components, namely exploitation i.e. to find out a better solution in specific region and exploration i.e. searching for a solution on a global level, are included in meta-heuristic algorithms which help to achieve a global optimum solution.

Figure 1 shows the following core activities of test suite optimization.

1. In the first step, test cases generated from software requirements specifications or source code of a software program are used as an input.
2. In the second step, an analysis is performed on these test cases in order to analyze them as per given requirements
3. On the basis of analysis performed in the second step, the initial test suite is selected from the pool of generated test cases
4. Then, an appropriate optimization algorithm, tool or technique, selected according to given requirements e.g. specific coverage criteria, fault effectiveness etc., is applied on the test suite
5. Finally, the optimized test suite is produced as an output

Test suite optimization is frequently researched topic and a recent review study has also been found [13] which partially analyzes 113 studies published up to 2016. However, the field of test suite optimization is quite intensive and there exist several studies in renowned scientific repositories (e.g. IEEE, ACM etc.) during the period of 2016 to 2019. Consequently, there is a dire need to perform a genuine and comprehensive SLR to investigate the latest developments in the area of test suite optimization. Hence, in this article, we extensively examine the 58 latest optimization studies in order to find state-of-the-art approaches, tools and supporting platforms. Following research questions are intended to be answered in this study:

RQ1: What are the primary approaches reported for improving the test suite optimization process?

RQ2: What are the primary tools employed/developed for the process of test suite optimization?

RQ3: What are the supporting platform used along optimization tools in the literature?

RQ4: How to improve modern test suite optimization approaches to accommodate the future technological advancements?

According to Kitchenham and Barbara [14] "*An approach to recognize, assess and infer all accessible studies that relate to a particular research question, or subject matter or phenomena of interest is referred to as systematic literature review*". For answering above-mentioned research questions, a Systematic Literature Review (SLR) is conducted for selection and investigation of 58 studies [15-72] published during 2016-2019. In **Figure 2**, an overview of this research work is portrayed. Following are the key contributions of this paper:

- Firstly, a comprehensive investigation of latest developments in test suite optimization is done in this study. Particularly, 58 latest research studies are considered. To the best of our knowledge, no research study is available yet that summaries and examines the state-of-the-art test suite optimization developments from 2016-2019.
- Secondly, the identification and analysis of different optimization techniques and 32 leading tools along 14 platforms for supporting and performing various test suite optimization tasks is done. Such analysis certainly benefits the researchers/practitioners to select the right technique, tool and platform as per the requirements.
- Finally, the significant research gaps where improvements are required in optimization approaches for achieving an optimal test suite according to defined criteria, are highlighted in this study

For conducting this SLR, a protocol for review (**Section 2**) is developed. Initially, we have defined six major categories (**Section 2.1**) to simplify the process of data extraction and synthesis (**Section 2.5**). Four scientific databases i.e. (Springer, ACM, Science Direct, IEEE) are used for the search process (**Section 2.3**) according to inclusion and exclusion rules (**Section 2.2**). Subsequently, 58 research studies that full complies with inclusion and exclusion criteria are identified. As shown in **Figure 2,** the selected researches are then analyzed and grouped (**Section 3**) into six major categories.

Subsequently, a comprehensive analysis of all categories is done i.e. Greedy algorithm (**Section 3.1**), Meta-Heuristic algorithm (**Section 3.2**), Hybrid algorithm (**Section 3.3**), Clustering algorithm (**Section 3.4**) and General (**Section 3.5**). Furthermore, several existing tools (**Section 3.6.1),** proposed tools (**Section 3.6.2**) and platforms support (**Section 3.6.3**) are identified and evaluated in **Section 3.6**. A comparative study is also performed to find out the strengths and weaknesses of each category (**Section 3.7**). The broad evaluation of selected studies results in providing the answers to the research questions (**Section 4**). Then, in **Section 5**the significant findings are discussed. Finally, the conclusion of this SLR is given in **Section 6**.

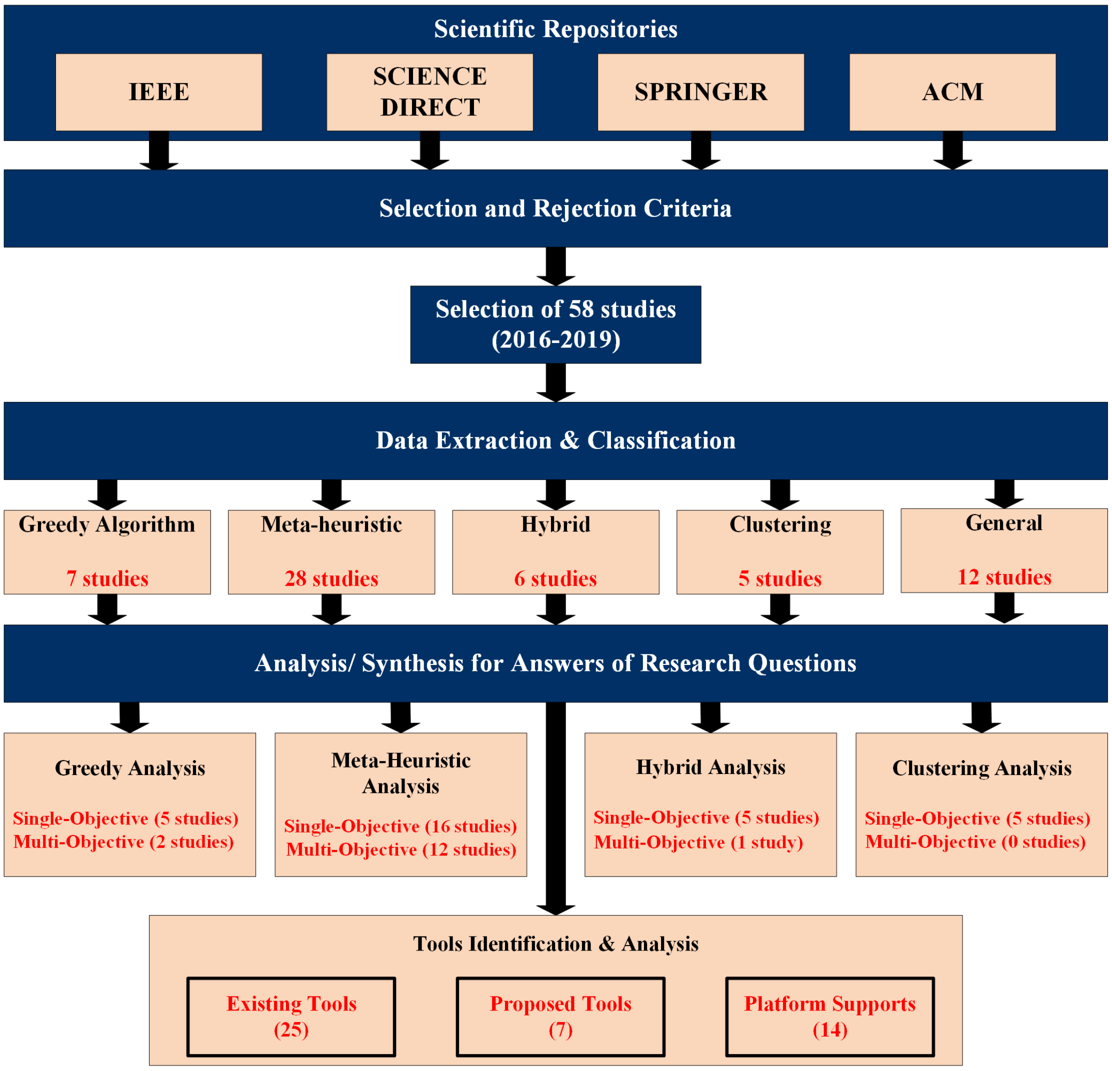
Scientific Repositories
IEEE
SCIENCE DIRECT
SPRINGER
ACM
Selection and Rejection Criteria
Selection of 58 studies (2016-2019)
Data Extraction & Classification
Greedy Algorithm 7 studies
Meta-heuristic 28 studies
Hybrid 6 studies
Clustering 5 studies
General 12 studies
Analysis/ Synthesis for Answers of Research Questions
Greedy Analysis Single-Objective (5 studies) Multi-Objective (2 studies)
Meta-Heuristic Analysis Single-Objective (16 studies) Multi-Objective (12 studies)
Hybrid Analysis Single-Objective (5 studies) Multi-Objective (1 study)
Clustering Analysis Single-Objective (5 studies) Multi-Objective (0 studies)
Tools Identification & Analysis
Existing Tools (25)
Proposed Tools (7)
Platform Supports (14)

FIGURE 2. Overview of systematic literature review.

II. REVIEW PROTOCOL

A review protocol has seven basic elements. In Introduction (Section I), the first two elements (i.e. Background and Research questions) are already presented. Thus, the details related to these two elements are not included in this section. Subsequently, the description of other five elements is given in following sections.

A. CATEGORIES DEFINITION

In order to simplify the process of data synthesis and extraction, we have defined five categories. The explanation of each category is provided below:

- **Greedy algorithm**: Greedy algorithm is one of the renowned code reduction-based heuristics. The test cases which comply with majority of disgruntled requirements are picked by it, but in case of draw condition, a random choice is made. The reduced test suite is obtained by repeatedly applying this process to each test case until every test requirement is fulfilled. From selected literature, seven test suite optimization based research studies that specifically deal with the greedy algorithm are selected and placed in this category.
- **Meta-Heuristic Algorithms**: Such researches which utilize meta-heuristic based techniques for test suite optimization are placed in this category. It is sub-categorized into Ant Colony algorithm, Genetic algorithm and Others (i.e. it incorporates different meta-heuristic techniques like Dragon-fly or Flower Pollination Algorithm (FPA), Harmony search (HS), Moth Flame Optimization (MFO) or Bat algorithm.
- **Hybrid Algorithms**: Several research studies are identified which deal with the hybridization of different approaches, such as genetic algorithm with bee colony optimization, for the generation of optimal test-suites. We have summarized all such researches in this category.

TABLE 1. Summary of search terms with results.

Sr.#	Search Terms	Operator	IEEE	ACM	Springer	Science Direct
1	Test suite optimization	N/A	265	24700	7447	15650
2	GA Test suite optimization	AND	35	2661	1483	2211
		OR	10881	39587	1987	67113
3	Hybrid algorithm Test suite optimization	AND	11	2661	1588	2029
		OR	8690	40561	2232	17480
4	Clustering Test suite optimization	AND	5	429	2554	4511
		OR	28027	27801	7999	244828
5	Heuristics algorithms Test suite optimization	AND	3	2661	1324	589
		OR	1577	39607	1631	26953

- **Clustering**: Clustering approach is used to optimize test cases and it improves the efficiency of software testing as well. Rather than checking the overall test cases, the whole program can be checked through any clustered test case. In our selected studies, there are five researches which utilize clustering algorithms/techniques for generating optimized test cases with the intention of handling the cost or time etc., of software testing process. All studies belonging to clustering-based optimization are considered in this category.
- **General**: From the pool of 58 selected studies, some researches are identified which do not specifically belong to a single approach e.g. researches that deal with the integer linear programming, integer non-linear programming, precision slicing, probability models and other approaches for the purpose of test suite optimization. All those studies are positioned in the general category.

B. INCLUSION AND EXCLUSION RULES

To accomplish the desired objectives, logical inclusion, as well as exclusion rules for conducting this SLR, are declared. The rules are outlined below:

- Firstly, the overall structure of research studies is taken into consideration i.e. state-of-the-art research studies, highly relevant to test suite optimization and it should belong to ACM**,** IEEE**,** Science Direct or Springer.
- Secondly, the content of abstract and title shall relate to research questions.
- Finally, the introduction and conclusion section of selected studies must contain the aspects of test suite optimization.

The research studies that do not conform to the inclusion criteria declared above and the following constraints, are excluded:

- Only one of those research studies is selected which have almost similar research contents.
- No peer-reviewed studies are taken into consideration
- Editorials and abstracts are excluded from the search process

C. SEARCH PROCESS

The process of finding research studies is started by searching four databases i.e. IEEE, Springer, ACM, Science Direct, according to our inclusion and exclusion criteria (**Section 2.2**). In order to perform the search process, we have also exploited several keywords. In **Table 1,** a brief description of keywords is given. For performing the search process, two operators i.e. AND, OR are utilized. As the studies obtained by using AND operator are not sufficient, so we have also employed OR operator.

Moreover, we have also used different options for the advance search that are available in selected databases e.g. "year span", "where keyword contains" etc. By using given keywords, a huge number of research studies are obtained which are not feasible to be examined entirely. The basic reason is that the test suite optimization keyword relates to different concepts. So, with the help of these advanced search options, we have improved the results e.g. specifying the subject as computer science, or machine learning etc. Furthermore, we used complete "Test Suite Optimization" keyword instead of TSO to accomplish the desired outcomes. Conclusively, 58 research studies are selected (**Figure 3**) with the help of this search process and our defined rules.

- Overall, 3601 studies are selected and 1763 of them get rejected on the basis of Title
- Subsequently, remaining 1838 research studies are taken into consideration and after reading the abstract of those studies 1456 more studies are excluded
- We thoroughly inspect the remaining 382 researches. As a result of our analysis, 324 studies get discarded and 58 research studies that entirely conform to our defined inclusion and exclusion criteria (**Section 2.2**) are finally selected.

TABLE 2. Summary of selected studies w.r.t databases and publication type.

Database	Type	No. of studies	Reference	Total
IEEE	Conference	18	[22] [23] [29] [31] [32] [35] [36] [39] [40] [43] [48] [51] [54] [55] [56] [58] [62] [69]	22
	Journal	4	[30] [44] [33] [60]	
Science Direct	Conference	1	[38]	7
	Journal	6	[18] [19] [24] [26] [47] [42]	
Springer	Conference	9	[28] [34] [45] [50] [53] [68] [70] [71] [72]	15
	Journal	6	[15] [16] [21] [27] [49] [52]	
ACM	Conference	11	[17] [20] [25] [37] [41] [46] [57] [61] [64] [65] [67]	14
	Journal	3	[59] [63] [66]	

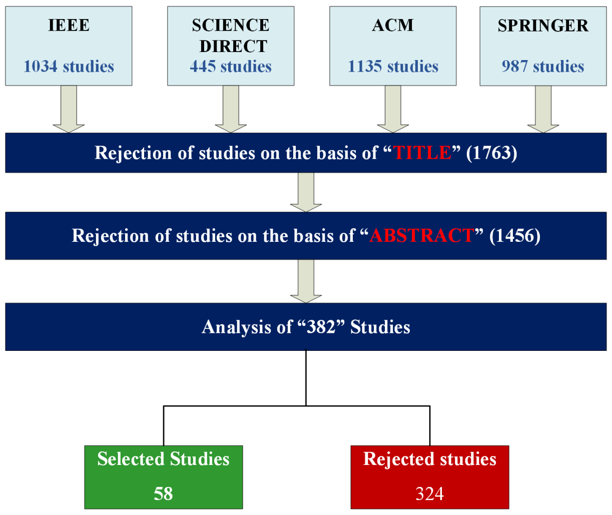
IEEE
1034 studies
SCIENCE DIRECT
445 studies
ACM
1135 studies
SPRINGER
987 studies
Rejection of studies on the basis of "TITLE" (1763)
Rejection of studies on the basis of "ABSTRACT" (1456)
Analysis of "382" Studies
Selected Studies
58
Rejected studies
324

FIGURE 3. Summary of search process.

D. QUALITY ASSESSMENT

For assuring the reliable result of this systematic review, high impact researches are selected from widely accepted and recognized databases. Twenty-two (22) researches are selected from IEEE repository, seven (7) from Science Direct, fourteen (14) from ACM and fifteen (15) from Springer, making a total of fifty-eight (58) selected studies. To the max, we intend to select researches that have a high impact as demonstrated in **Table 2**. Additionally, the aim is to analyze state-of-the-art research studies essentially as shown in **Figure 4**. Therefore, we ensure that the results of this SLR are impactful and reliable. In **Table 2**, the summary of selected repositories in accordance with the type of publication is given. **Database** signifies the name of repository from which research study is retrieved, **No. of studies** depicts the quantity of conference and journal papers selected from each database. **Type** signifies whether the selected study belongs to a conference or journal. For each study, **references** are also given. In the last column, sum of **total studies** is also provided for further exploration. From **Table 2**, it is evident that most conferences are from IEEE repository i.e. eighteen (18), while maximum journals are from science direct and springer i.e. six (6) from each repository.

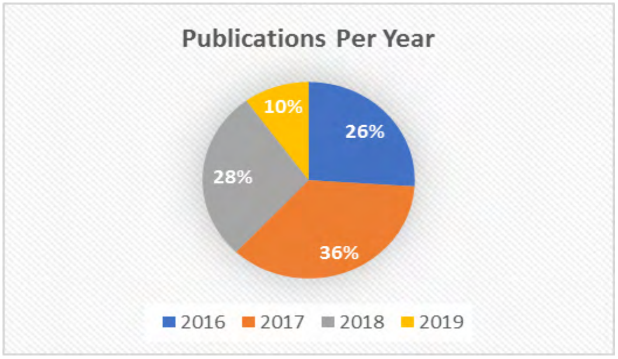

FIGURE 4. Distribution of selected studies w.r.t publication year.

E. DATA EXTRACTION AND SYNTHESIS

A complete template is developed for extraction of data as well as synthesis as shown in **Table 3**. Initially, the extraction of information related to bibliography i.e. title, authors, year of publication and research paper type is done. Next, fundamental findings like technique which is proposed, details of implementation, evaluation and experimentation process are extracted. For achieving objectives of SLR, this extracted data help to conduct a complete analysis. Techniques and tools used/proposed for implementing test suite optimization process are identified from selected research studies. Lastly, a detailed analysis is done to obtain answers about research questions defined in **Section 1**.

III. RESULTS AND ANALYSIS

As shown in **Table 4,** we have classified selected studies into five main categories. For further investigation, the references for subsequent studies are also given. From **Table 4**, it is observable that there're seven (7) studies that belong to the greedy algorithm. Twenty-eight (28) researches are associated with the meta-heuristic category and it is sub-categorized into a genetic algorithm that comprises of thirteen (13) studies, ACA algorithm which incorporates four (4) studies and other meta-heuristic algorithms/techniques embodies eleven (11) studies. Six (6) research studies are found in the hybrid category while five (5) researches are associated with clustering category. Lastly, twelve (12) studies are placed in the general category.

TABLE 3. Data extraction and synthesis template.

Sr.#	Description	Details
1	Bibliographic Information	Title, authors, year of publication and research paper type is analyzed
2	Proposed Technique	The approach followed by selected researches is observed
3	Implementation Details	Technologies that are used for implementation of proposed methodology are observed
4	Results	Results of each selected paper are analyzed thoroughly
5	Classification	Grouping of selected studies according to categories defined in **Section 3**. In **Table 4,** a summary **of** results is provided
6	Investigation of Categories	For finding the solutions of selected research questions, analysis of each category is performed. Results are summarized as follows: Greedy algorithm category (**Table 5**) Meta-heuristic algorithm category (**Table 6, Table 7, Table 8**), Hybrid algorithm category (**Table 9**), Clustering algorithm category (**Table 10**), General category (**Table 11**)
7	Tools and Platform Support	Tools used or proposed (**Table 12**, **Table 13**,) and supporting platform (**Table 14)** in each study are analyzed

TABLE 4. Classification of selected researches into five main categories.

Sr.#	Category	References of corresponding studies	Total
1.	Greedy Algorithm	[15] [16] [17] [18] [19] [20] [21]	7
2.	Meta-Heuristic Techniques:		28
	Genetic Algorithm	[22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34]	13
	Ant Colony algorithm	[35] [36] [37] [38]	4
	Other Meta-heuristic techniques	[39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49]	11
3.	Hybrid	[50] [51] [52] [53] [54] [55]	6
4.	Clustering	[56] [57] [58] [59] [60]	5
5.	General	[61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72]	12

As depicted from selected literature, a fair amount of work is done on algorithms for optimization of the test suite. Yet, researchers are making efforts to explore test suite optimization in other directions too. The process of analysis/synthesis is simplified by the categorization of studies (**Table 4**). Each category is investigated comprehensively so that precise outcome can be presented and the objective of SLR can be achieved. In later sections, the details are given for each of them. In order to clearly state we evaluate them according to certain parameters. Firstly, the name of the algorithm used for optimization in each study, is outlined. Secondly, the objective of each study is stated to depict whether the test cases are generated to reduce execution time or cost, increase fault detection ability or maximize their code coverage, path coverage, or requirement coverage etc. Thirdly, the tool used or proposed in each study is also extracted, so that researchers can select the appropriate tool as per their requirements.

The detail of each tool is further given in **Section 3.6.** Subsequently, the running environment of each study is also included to help researchers in selecting the appropriate tool or technique according to the compatibility with their systems' specification. It includes the details about Operating System (OS) version, memory (RAM) and processor required for the proper functioning of the tool. Lastly, the dataset is enlisted to represent the test data used in each study for validation of proposed techniques, which will help to figure out the suitability of the proposed technique for a specific programming language.

A. GREEDY ALGORITHM CATEGORY

A greedy algorithm is one of the well-known code reduction-based heuristics. The test cases that satisfy the majority of disgruntled requirements are picked by it, but in case of a draw condition a random choice is made. The reduced test suite is obtained by repeatedly applying this process to each test case until every test requirement is fulfilled. In **Table 5**, the evaluation of seven greedy based studies is performed according to parameters defined in Section 3.

Singh and Shree [15] presents a systematic approach using CMMIX (enhanced greedy algorithm) for the reduction of test suite size. An experiment conducted on small program code indicates that with the help of this approach, the test suite can be optimized effectively without changing its coverage value. Wang *et al.* [16] propose a multi-criteria based selection of test cases using a greedy algorithm.

TABLE 5. Summary of investigation for greedy approach.

Sr.#	Ref.	Algorithm	Objective	Tool	Running environment			Dataset
					OS	RAM	PROCESSOR	
1	[15]	CMMIX	Path coverage	N/A	N/A	N/A	N/A	Program code
2	[16]	Greedy algorithm	Fault localization Reduction rate	GCOV	Ubuntu 8.04	4 GB	Core 4 Duo	Siemens test suite, Space benchmark
3	[17]	Approximate greedy algorithm	eCoverage	Monkey EMMA	Windows 7	16 GB	Core i7	15 apps
4	[18]	Cost-aware greedy algorithm	Fault detection	N/A	Windows 7 SP1	2 GB	Core Duo	SIR , Siemens, space, gzip, ant
5	[19]	Scope-aided Greedy algorithm	Fault detection	KLEE MILU	N/A	N/A	N/A	17 variant versions from 3 C subjects
6	[20]	Greedy algorithm	Failed-Build Detection Loss	PIT	N/A	N/A	N/A	Dataset from Travis torrent
7	[21]	Greedy algorithm	Fault detection Execution time reduction	N/A	N/A	N/A	N/A	SIR, Siemens benchmark

Experimentation done on 3535,392 test cases reveals that the proposed approach performs better in terms of test suite reduction and effectiveness of fault localization as compared to the statement and vector-based approaches. Jabbarvand *et al.* [17] describe two ways of reducing the tests: an integer programming formulation and a greedy algorithm using EMMA tool and Ip-solve. Evaluation of proposed techniques on 15 apps selected from F-droid repository shows that both algorithms maintain the quality of test and a considerable reduction in the size of the test suite is achieved.

Lin *et al.* [18] focus on the Greedy-based techniques and empirically evaluates the additional Greedy and two cost-aware Greedy techniques using Siemens, gzip space, and ant programs. Execution of test cases is done on Ubuntu 10.10 running on Windows 7 SP1, having an intel core duo processor with two GB memory. According to evaluation results, lower cost of regression testing and higher efficiency of fault detection is accomplished with cost-aware methods. Miranda and Bertolino [19] employs scope-aid for boosting total and additional greedy, similarity-based and search-based prioritization; greedy additional selection; and the GE algorithm for minimization. For experimentation, 17 variant versions from grep, gzip, and sed are selected. In order to determine the in-scope entities and mutant generation, KLEE and MILU tools are also used. Empirical evaluation shows that without affecting the fault-detection ability, this technique results in significant reduction in size of the test suite. Shi *et al.* [20] consider the problem of missing faults after the reduction of test suite, which can be covered by the original test suite. Greedy algorithm and three other approaches namely: GE, GRE, and HGS are used for evaluation purpose. From 32 GitHub projects, 1478 failed builds are selected for evaluation. Results suggest that FBDL (Failed-Build Detection Loss) cannot be predicted properly using traditional test suite reduction metrics. Wang *et al.* [21] propose a distance-based test suite reduction technique for increasing the fault localization efficiency. A greedy algorithm is used for determining the optimal solution. An empirical investigation conducted on SIR benchmark and Siemens indicates that reduction in test suite size, as well as time and cost of testing based fault localization, is achieved using the proposed approach.

From **Table 5**, it is analyzed that SIR and Siemens benchmarks are mostly used as a dataset in greedy algorithm based test suite optimization studies. By evaluating the results obtained from selected studies, it is evident that the greedy algorithm mainly focuses on achieving higher efficiency in terms of fault detection.

B. META-HEURISTIC ALGORITHM CATEGORY

A meta-heuristic is an advanced algorithm framework which is independent of problem. It offers strategies which help in developing heuristic algorithms for optimization purposes. In this study, we have selected twenty-eight researches which belong to different meta-heuristic based methods, for example genetic algorithm, ACO algorithm and some other meta-heuristic techniques/algorithms. Given below are the details of each sub-category:

1) GENETIC ALGORITHM

An interesting area for researchers is the implementation of evolutionary algorithms for the generation of optimized test cases. One such type of algorithm is computational intelligence based methodology called Genetic Algorithm (GA). From our fifty-eight (58) selected research studies, we have found thirteen researches which have utilized a genetic algorithm for the purpose of test suite optimization. In **Table 6**,

TABLE 6. Summary of investigation for a genetic approach.

Sr. #	Ref.	Algorithm	Objective	Tool	Running Environment			Dataset
					OS	Processor	RAM	
1	[22]	Fitness based Genetic algorithm	Path coverage	N/A	N/A	N/A	N/A	C program
2	[23]	Genetic algorithm	Code Coverage	EMMA, JUnit	N/A	N/A	N/A	N/A
3	[24]	Genetic Strategy	Execution time reduction	N/A	Windows 7	Core i7	6GB	CA (N; 2, 7, 5)
4	[25]	Genetic algorithm	Execution cost	Randoop	N/A	N/A	N/A	Randoop Test suite
5	[26]	MORTOGA	Requirement coverage Fault detection	N/A	N/A	N/A	N/A	30,834 test cases
6	[27]	Parallelized genetic algorithm	Execution time reduction	Spark	Ubuntu	Core i5	4GB	19 benchmarks
7	[28]	NSGA-II	Code & Branch coverage, Execution time reduction	Mockito	N/A	N/A	N/A	420 tests
8	[29]	Steady-state genetic algorithm	Execution time reduction, Requirement & code coverage	N/A	N/A	N/A	N/A	1, 000 test cases
9	[30]	DynaMOSA	Statement Mutation & Branch coverage	EvoSuite	N/A	N/A	N/A	346 java classes
10	[31]	Enhanced Genetic Algorithm	Functional & Structural coverage, Execution time reduction	QuestaSim	N/A	Core i7	N/A	Case Study
11	[32]	NSGA-II	Frequency, Order, Constraint	N/A	N/A	N/A	N/A	35 SUT models
12	[33]	NSGA-II	Code coverage, requirements coverage, Execution time reduction	N/A	N/A	N/A	N/A	20 java apps
13	[34]	Real-coded genetic algorithm	Path coverage	N/A	N/A	N/A	N/A	Triangle classifier, Greatest Common Divisor program

we have summarized the algorithm name, the objective of the study, a tool used, a specific requirement of running environment and dataset extracted from the respective studies.

Khan *et al.* [22] employ a genetic algorithm in order to automate the generation of the optimized test case. A criterion for du-path adequacy is introduced with the help of fitness function. Authors validate the given method with the help of a C program. Similarly, Kothari and Rajavat [23] implement a model for software testing with the help of a genetic algorithm. They develop the application with the help of EMMA and JUnit testing tool. According to the results, the proposed model is capable of producing optimized test data, accompanied by the improvements in space as well as the time complexity of the system. Esfandyari and Rafe [24] proposes an approach for the minimal generation of test suite using GA, known as Genetic Strategy (GS). They used CA (N; 2, 7, 5) with a population of size 150 for experimentation. The approach is implemented in MATLAB on windows 7 with six GB RAM and 2.20 GHz core i7QM CPU. Experimental results reveal that GS supports higher interaction strengths as compared to other strategies. Schuler [25] attempts to reduce the time of execution and size of test data whilst maintaining their coverage. Test data is generated with the help of randoop framework and genetic algorithm is applied to it. 3 open source projects selected from the F-Droid repository are used as a case study to empirically evaluate the proposed approach.

Garousi *et al.* [26] introduce a genetic algorithm based approach called MORTOGA. An empirical study performed on this approach shows that it provides better coverage of changed requirements as well as cost and benefit analysis as compared to the manual approach. In order to deal with time-consuming nature of NP-hard problems that often results in substantial limitations for the real time application of genetic algorithms in large-scale problems, Qi *et al.* [27] propose a two-way parallelization algorithm in order to parallelize the genetic algorithm. Implementation of propose technique on 5 real-world and 14 pairwise synthetic benchmarks using Ubuntu 12.04 shows that it competes for sequential techniques in terms of reduced test data size and computation. Turner *et al.* [28] apply a well-known multi-objective approach i.e. Non-dominated Sorting Genetic Algorithm II (NSGA-II) on the test suite of Mockito 2.0. By using multi-objective optimization, they analyze the trade-off among execution time and code coverage for the selected test suite. It is obvious from the results that minor reduction in coverage of code can result in a significant decrease in execution time.

Yamuç *et al.* [29] presents a GA based approach for test case reduction along its comparison with a greedy approach using 1, 000 test cases and a sum of 10, 000 test requirements. Although, greedy approach produces better results in terms of execution time GA substantially outperforms greedy algorithm in terms of cost reduction with a factor of 26.14%. Subsequently, by considering the cost reduction factor, the overhead of processing time can be ignored. Panichella *et al.* [30] propose a novel many objective genetic algorithm DynaMOSA to redevelop the problem of test case generation as a problem of many objective optimizations. A prototype tool i.e. extension of EvoSuite framework, for test data generation, is used for implementation. Empirical assessment done on 346 java classes shows that DynaMOSA holds substantial improvements with respect to statement, branch and mutation coverage as compared to whole suite approach and its antecedent MOSA. Zachariáová *et al.* [31] employ the genetic algorithm in order to build a new approach for regression suite optimization of Application-specific Instruction-Set Processors (ASIPs). According to simulation results generated by QuestaSim tool, this optimization technique reduces the simulation runtime and the resulting statistics for coverage remains equal to actual suite coverage.

Sabbaghi and Keyvanpour [32] employ the non-dominated sorting genetic algorithm-II (NSGA-II) to deal with multi-objective optimization problem of combinatorial testing. Different criteria are taken into consideration i.e. Considering order (CO), Considering frequency (CF), Considering constraints (CC). Experimental evaluation done on 35 SUT models reveals that the proposed approach results in the generation of the prioritized and reduced test suite. In another work, Marchetto *et al.* [33] propose an NSGA-II based multi-objective test suites reduction approach named MORE+. It considers coverage of source code and application requirements along the reduction in execution cost. According to experiments performed on 20 java applications, proposed approach provides better results in terms of cost-effectiveness as compared to baseline approaches. Lastly, Mishra *et al.* [34] propose e real-coded genetic algorithm for path coverage (RCGAPC). The proposed approach is more efficient in covering the critical paths as compared to traditional genetic algorithm. Evaluation of proposed approach done on two programs i.e. triangle classifier and greatest common divisor, shows that it reduces the number of test data generation required for path testing and produce an optimized test suite that covers 100% path for specific software.

From **Table 6**, it is analyzed that the genetic algorithm is used as a significant approach for test suite optimization. In most of the selected research studies, this algorithm mainly provides support to reduce the execution time of test cases.

2) ANT COLONY ALGORITHM

Ant colony optimization (ACO) is a computational problem-solving algorithm which is based on probability. It generates a solution by traversing a graph that consists of several states of the system. In software testing, ACO is used for generating test sequences. It maximizes their code coverage, path coverage, or requirement coverage etc. In **Table 7**, The evaluation of ACO based studies is performed according to parameters defined in **Section 3**. The parameter Running environment is not included in this sub-category, as none of these studies contains the details about OS, processor and RAM used in their experimentation process.

Zhang *et al.* [35] put forward a method for reduction of test suite based on modified quantum ant colony algorithm. They utilize seven programs written in C from the Siemens test suite and run it on a Linux platform. GCOV is used for calculating the statement coverage of this program. Results of simulation experiments show that the modified quantum ant colony system can effectively solve problems as compared to other algorithms. Kumar *et al.* [36] propose a solution for test cases optimization in large search space using modified ant colony optimization. A dataset consisting of 10 test cases is taken into consideration in order to validate the technique. Results show that the modified algorithm helps to reduce both effort and cost by selecting the test cases which take less time to find maximum faults. Han *et al.* [37] present a solution for discrete multimodal optimization by introducing a new niching algorithm named NACS. A test suite comprising ten MMO-TSP instances is also introduced for testing the performance of this algorithm. The result shows that during optimization, NACS exhibit great ability in terms of exploration by locating and maintaining multiple distinct optima. Ansaria *et al.* [38] employ the ant colony optimization technique to automate the optimization of test cases list. Five test cases are taken as input along their disclosed faults and execution time. For the assessment of the proposed technique, APFD metric is utilized.

From **Table 7,** it is observable that different optimization approaches are introduced based on ant colony algorithm. However, these algorithms are not practically more useful because only one of these researches has used a standard dataset for validation of the proposed technique and no adequate tool support is mentioned. Only the partial implementation has been focused in these researches, thus more work is needed in this direction for practical achievements.

3) OTHER META-HEURISTIC ALGORITHMS

We have evaluated eleven studies where some random meta-heuristic algorithms, such as harmony search, PSO, flower

TABLE 7. Summary of investigation for ant colony algorithm.

Sr.#	Ref	Algorithm	Objective	Tool	Dataset
1	[35]	Modified Quantum Ant Colony	Statement coverage	GCOV	Siemens
2	[36]	Modified ant colony optimization	Fault coverage rate, Execution time reduction, Code coverage	N/A	10 test cases
3	[37]	Niching Ant Colony System	Path coverage	N/A	A test suite with 10 MMO-TSP instances
4	[38]	Ant Colony Optimization	Fault coverage, Execution time reduction	N/A	Five test cases

TABLE 8. Summary of investigation for different meta-heuristic techniques.

Sr .#	Ref	Algorithm	Objective	Tool	Running Environment			Dataset
					OS	Processor	RAM	
1	[39]	Modified Flower Pollination	Execution time reduction	N/A	Windows 7	core i7	4GB	2 experiments
2	[40]	Moth Flame Optimization	Branch coverage	N/A	N/A	N/A	N/A	5 benchmark
3	[41]	Multi-Objective Particle Swarm Optimization	Branch coverage	MOTestGen	Windows 8	Core i3	4 GB	Triangle classifier SUT
4	[42]	Adaptive Teaching Learning-based Optimization	Execution time reduction	N/A	Windows 10	Core i5	16GB	CA, VCAs
5	[43]	Diversity Dragonfly	Execution time reduction	N/A	Windows 7	N/A	2GB	5 SIR subjects
6	[44]	DIVersity-based BAT	Requirement coverage	N/A	N/A	N/A	N/A	8 SIR subjects
7	[45]	Test Generator Flower Pollination	Execution time reduction	N/A	Windows 8.1	Core i5	4GB	3 benchmarks
8	[46]	Harmony search	Fault coverage, Execution time reduction	N/A	N/A	N/A	N/A	5 SIR benchmarks
9	[47]	Multi-objective evolutionary algorithm (MOEA/D)	Cost, statement, branch, and modified condition /decision coverage	GCOV	N/A	N/A	N/A	6 SIR subject, VoidAuth program
10	[48]	Many-objective optimization	Branch & statement coverage	N/A	N/A	N/A	N/A	4 SIR subjects
11	[49]	Harrolds–Gupta–Soffa (HGS) method	Fault coverage, Execution time reduction	N/A	N/A	N/A	N/A	12 SIR versions

pollination, dragon-fly and bat algorithms are used for the purpose of test suite optimization. **Table 8** represents the data extracted from these studies. As different existing or proposed meta-heuristic approaches are used for optimization purposes in this sub-category, the names of different algorithms/techniquesare included in **Table 8**. Subsequently, the objective of each study along the tool name, running environment and dataset is also listed in the table.

Kabir *et al.* [39] introduce an optimization technique namely Modified Flower Pollination Algorithm (MPFA) and implements it in java under Windows 7 OS and Net Beans environment. Results of experimentation done on two programs reveal that MPFA algorithm provides better convergence rate as compared to other techniques. Metwally *et al.* [40] propose an approach that is based on MFO (Moth Flame Optimization) algorithm. It attempts to obtain a reduced test suite that achieves maximum coverage. For the evaluation of proposed technique, experiments are performed on five benchmark methods using MATLAB. The outcome shows that the MFO technique is better up to two orders of magnitude in comparison to the random generator. In four benchmark methods, MFO produces better results than Genetic approach. Gopi *et al.* [41] propose MOPSO (Multi-Objective Particle Swarm Optimization) algorithm for search-based test data generation. Two objective functions namely production of optimal data and maximum branch coverage are introduced in it. For the production of optimal test data and extraction of their coverage and convergence performance, a tool named MOTestGen is also developed. Results reveal that when the population size increase then coverage becomes maximum.

Zamli *et al.* [42] present an approach named ATLBO (Adaptive Teaching Learning-based Optimization), which is centred on the mamdani-type fuzzy inference system strategy for exploitation and exploration. The experimental environment consists of a PC running on Windows 10 with 2.9 GHz core i5 processor and 16 GB RAM along 512 MB flash hard-disk drive. According to results, test data is minimized systematically with the help of ATLBO technique based on given interaction strength. Sugave *et al.* [43] present a novel algorithm i.e. DDF(Diversity Dragonfly Algorithm) for determining the best suite based on the hunting procedure of the dragonfly that uses a minimum objective function. The experiment is carried out using five subject programs taken from SIR using Net-Beans IDE 7.3. It is found that cost of the proposed DDF is low and the reduction capability of the DDF is better than existing methods. In another work, Sugave *et al.* [44] develop two techniques for reducing test suite. In the first technique, ATAP measure is developed while in second one DIVersity-based BAT algorithm is devised. For assessment of propose techniques, eight programs written in C language are utilized from SIR repository. Results prove that DIV-TBAT outclasses the entire present approaches in test suite reduction. Alsewari *et al.* [45] propose a new technique based on flower pollination algorithm called Test Generator Flower Pollination Strategy (TGFP) for minimization of test cases. Results of two experiments show that with different interaction strengths TGFP performs better as compared to existing strategies of combinatorial testing with respect to execution time.

A Pareto based harmony search algorithm for test case selection in regression testing is taken into consideration by Choudhary *et al.* [46]. An empirical evaluation of its performance with bat and cuckoo search algorithm is also done in this study. It is evident from the results that in most of the cases propose approach performs better than the other two approaches. Zheng *et al.* [47] adapt a multi-objective evolutionary algorithm based on decomposition (MOEA/D) for regression testing. For the purpose of experimentation, authors employed VoidAuth program and six programs retrieved from SIR. Comparison of proposed algorithm with four approaches shows that for the purpose of multi objective optimization, MOEA/D with tuning produce most effective results in all of the experiments. In another work, Wei *et al.* [48] develop a many-objective optimization approach for regression test suite minimization based on mutation testing. Evaluation perform on six many-objective evolutionary algorithms depicts that with no change in the capability of fault detection, a considerable reduction in the cost of testing is achieved using this technique. Agarwal *et al.* [49] presents a novel method referred as fault coverage-based test suite optimization (FCBTSO) for regression test suite optimization. Proposed approach is based on Harrolds–Gupta–Soffa (HGS) test suite reduction method. The computational experiments performed on 12 versions of benchmarked programs retrieved from SIR indicates that FCBTSO outperforms other approaches with respect to the execution time.

From **Table 8**, it is analyzed that different meta-heuristic approaches are proposed to generate an optimized test suite. Each of these meta-heuristic approaches focus on different performance objectives e.g. MFO provides branch coverage, two objective functions i.e. production of optimal data and maximum branch coverage are introduced in MOPSO, ATLBO focuses on size of test suite, in Pareto based harmony search algorithm fault coverage is utilized as performance measures, algorithm execution time is taken into consideration by TGFP as well as Pareto based harmony search approach, MPFA algorithm provides better convergence rate, and DDF and DIV-TBAT exhibits low cost and higher reduction capability.

C. HYBRID ALGORITHM CATEGORY

Some studies aim to combine the advantages of two or more algorithms to form a hybrid algorithm, while simultaneously trying to minimize any significant disadvantage in order to reduce test cases, such as the use of genetic and particle swarm optimization algorithm together. From the pool of 58 selected studies, 6 researches which deal with the hybrid approach are placed in this category. In **Table 9**, the summary of the data extracted from the research studies that incorporate a hybrid approach is presented. Each study is evaluated with the help of parameters defined in Section 3.

Nasser *et al.* [50] proposes a hybrid algorithm employs integration student phase of Teaching Learning Based Optimization (TLBO), named learning cuckoo search strategy. Experimental study shows that the proposed algorithm performs better as compared to original cuckoo search and other existing strategies. Singhal *et al.* [51] empirically analyze the hybrid GA and BCO technique, known as MHBG_TCS, according to time constraint. For all values of TC, MHBG_TCS tool is practically evaluated on seventeen open source programs. It is found that proposed technique is a near optimal and maximum reduction in terms of size is achieved.

Anwar *et al.* [52] present a novel technique for optimization of test suites and named it as hybrid adaptive neuro-fuzzy interface system, which is tuned with GA and PSO. Benchmark test suites are used for evaluation and it shows that higher requirement coverage in terms of test suite reduction without affecting the rate of fault detection is attained. The basic aim of this research study by Khan *et al.* [53] is to customize the cost and time for the testing process after the automatic test case generation. According to performed analysis, these two optimization techniques namely cuckoo search and genetic algorithm produce better result together as compared to a single one. Saraswat and Singhal [54] propose a hybrid GA_PSO algorithm and implement it in java. A case study of e-learning website is taken into

TABLE 9. Summary of investigation for hybrid technique.

Sr.#	Ref	Algorithm	Objective	Tool	Running Environment			Dataset
					OS	Processor	RAM	
1	[50]	Learning Cuckoo Search	Convergence rate	N/A	N/A	N/A	N/A	N/A
2	[51]	Genetic algorithm Bee Colony Optimization	Execution time reduction	MHBG_TCS	N/A	N/A	N/A	17 programs
3	[52]	Genetic algorithm PSO	Requirement coverage, Fault detection	N/A	Linux	Core i7	8GB	Benchmark test suites
4	[53]	Cuckoo Search Genetic algorithm	Execution time reduction	N/A	N/A	N/A	N/A	Randomly generated data
5	[54]	Hybrid GA_PSO	Fault detection	N/A	Linux	N/A	N/A	Case study
6	[55]	Cluster-based genetic algorithm	Execution time reduction	jMetal	Mac	Core i7	16GB	3 optimization problems

consideration for validation of proposed approach. Test suite prioritization is also performed with this algorithm and the final outcome shows that the proposed hybrid method is capable of producing optimized test suite. To solve the multi-objective test optimization problem, Pradhan *et al.* [55] introduces a cluster-based genetic algorithm with the elitist strategy (CBGA-ES). The results show that CBGA-ES significantly improves the quality of the solutions and it outclasses other algorithms with respect to achieved objectives for the selected test optimization problem.

From **Table 9** it is obvious that although useful, existing work has not sufficiently dealt with hybrid algorithms as the strength for test suite optimization. Out of 58 selected studies, only 6 of them deal with hybrid approaches for optimization purposes. Three (3) of them focus on reducing the execution time of test cases [51], [53], [55], other two deals with fault detection [52], [54] while in [52] requirement coverage is also considered as well. Finally, one study [50] deals with the convergence rate.

D. CLUSTERING ALGORITHM

In designing of test cases, many redundant test cases may appear. An increase in cost and time of software testing can be caused due to these redundancies. By use of clustering techniques, test cases can be optimized and the efficiency of testing software can be improved. Rather than checking the overall test cases, the whole program can be checked through any clustered test case.

According to pre-defined parameters in **Section 3**, we have summarized the data extracted from the studies that lie in Clustering category (**Table 10**). The details about running environment i.e. OS, memory and RAM, is not given any of cluster-based optimization study. Therefore, this specific parameter is not included in **Table 10**.

Liu *et al.* [56] employ the k-mediod clustering algorithm for test suite optimization. They have also utilized the greedy algorithm for ensuring that test requirements are covered properly with generated test cases. In this study, Eclemma is used for calculating coverage and complexity of the test case generated by CodeproAnalytix and experiments are conducted on eclipse 4.2. According to experimental results, this technique features low complexity with higher coverage rate under the same quantity of streamlined test suite. Coviello *et al.* [57] propose a prototype tool named CUTER i.e. ClUstering-based TEst suite Reduction, for inefficient reduction of test cased. This tool implements the clustering technique and instances from its fundamental procedure. On 19 different versions of 4 Java programs, CUTER is applied as an eclipse plug-in. In another work, Coviello *et al.* [58] use hierarchical agglomerative clustering for inadequate test suite reduction. They consider 19 experimental objects of four java based software systems reduction of the test suite. For the implementation of four selected approaches, RAISE tool is used and their coverage information is gathered using JaCoCo. Results suggest that the proposed technique gives better test suite size reduction rate at the cost of the minor deficit in fault-detection ability.

Reichstaller *et al.* [59] evaluate two clustering techniques i.e. dissimilarity-based sparse subset selection and affinity propagation, for reduction of test cases. An evaluation case and a running example are introduced for the assessment of proposed techniques. It is evident from the results that drastic reduction in test suite along sound mutation score is achieved with these methods. Rosero *et al.* [60] utilize a combination of DB schema, random values and unit tests with unsupervised clustering for determining new features which are added to software products linked with databases or the test cases which relates to changes. Two DB applications are taken into consideration for examining different metrics like recall, precision, fault detection capability, test suite reduction and the F-measure. DBUnit and JUnit are also used for designing and implementation of test cases. Experimental results imply that effective clusters of test cases are achieved with the proposed method.

TABLE 10. Summary of investigation for clustering technique.

Sr.#	Ref.	Algorithm	Objective	Tool	Dataset
1	[56]	K-medoids	Code coverage	Codepro Analytix, Eclemma	Test cases generated by Codepro Analytix
2	[57]	Hierarchical agglomerative clustering	Statement Coverage	CUTER, JaCoCo1, JUnit2	19 versions of 4 Java programs
3	[58]	Hierarchical agglomerative clustering	Statement Coverage	RAISE, JaCoCo	SIR objects
4	[59]	Affinity Propagation, Dissimilarity-based Sparse Subset Selection	Mutation score	N/A	Case Study
5	[60]	Expectation-Maximization	Fault detection	DBUnit, JUnit	Two DB applications

From **Table 10**, it is analyzed that out of 5 clustering based research studies, 2 of them focus on statement coverage [57], [58], other three studies deal with fault detection [60], mutation score [59], and code coverage [56] respectively.

E. GENERAL CATEGORY

In General Category, we have placed twelve (12) research studies which do not belong to a specific technique or algorithm. They have provided extensive work on test suite optimization using different approaches such as probability model, integer linear or non-liner programming. All such studies are put under this category and analyzed with respect to major parameters defined in Section 3. **Table 11** is developed to determine the prominence of this category.

Choi *et al.* [61] introduce an approach based on SwiftHand and Random algorithm for reducing large test data which is generated by an automated Android GUI testing tool. A prototype tool named DetReduce is used for implementation of the algorithm. By testing DetReduce on 18 apps, it is found that test suites effectively reduce with the help of proposed technique. Marijan *et al.* [62] propose a test suite optimization technology for increasing the time-and cost-efficiency of testing highly configurable software, called TITAN. This technology visualizes several attributes of test suites like redundancy and coverage so that engineers can perform an evaluation of quality and status at various phases. Liu *et al.* [63] propose a test suite reduction approach based on the probability model. It is different from traditional methods of reduction which are centred on coverage. Grep system with 50 versions is taken into account for validation of proposed technique. The result shows that it is not essential for the proposed approach to cover each test requirement. However, it still depicts the equivalent ability of fault detection as the original test suite.

Lin *et al.* [64] propose a framework called Nemo for the formulation of multi-criteria test-suite minimization problem as integer non-linear programming problem. From a publicly available dataset, five open-source C projects i.e. make, grep, sed, flex, and gzip are selected. Results of the experiment reveal that with modern solvers we can use Nemo to competently find an optimum solution for the problem of multi-criteria test-suite minimization. Palomo-Lozano *et al.* [65] focus on testing WS-BPEL compositions and present a search-based technique for test suite minimization whilst preserving the mutation coverage. For reducing the testing effort along good results, integer-liner programming technique is used. C.R Panigrahi and Panigrahi and Mall [66] proposes a technique based on improved precision slices for selection of regression test cases aimed at the reduction of test suite size. Seven java programs are used to test the effectiveness of this technique. The result indicates that size of the regression test suite is reduced effectively by by using this approach, devoid of worsening fault detection efficiency. Vahabzadeh *et al.* [67] propose a model to facilitate analysis of test code at statement level and present a technique along with a tool for minimization of substantial redundancies in the statement of test cases, named as Tesler. Empirical evaluation of the technique is done on fifteen subjects. Tesler reduces the partially surplus test cases in less time and preserves the coverage of the original test suite as well as production method call behaviour. Carlsson *et al.* [68] present a novel approach for constraint optimization that takes into account a sophisticated search heuristic along GlobalCardinality and NValue constraints. Experimental evaluation is done on standard benchmarks and arbitrarily generated instances with the help of Flower/C tool. For the process of constraint optimization, this tool is compared with current tools and results depict the competitiveness of the proposed approach with tools on benchmarks and random instances. Fu *et al.* [69] introduce a similarity centred TSR approach to improve localization of faults based on the spectrum. GCOV is used for collecting execution traces of different test cases, Siemens and UNIX utility programs are used to conduct experiments. It is evident from the outcomes that the effectiveness of fault localization can be considerably improved with propose technique.

TABLE 11. Summary of investigation for general category.

Sr. #	Ref	Algorithm/ Technique	Objective	Tool	Running Environment			Dataset
					OS	Processor	RAM	
1.	[61]	DetReduce	Execution time reduction	DetReduce	N/A	N/A	N/A	18 apps
2.	[62]	TITAN	Requirement coverage Fault detection	TITAN	N/A	N/A	N/A	15 test suites
3.	[63]	Probability models	Requirement coverage	N/A	N/A	N/A	N/A	40,450 test cases
4.	[64]	Integer nonlinear programming	Statement coverage Fault detection	Nemo	N/A	N/A	N/A	C projects Grep, Flex, Sed, Make, and Gzip
5.	[65]	Integer linear programming	Mutation coverage	N/A	laptop	Core i5	8 GiB DDR3L	N/A
6.	[66]	Improved slicing	Fault detection	IReTEST	Windows XP	N/A	N/A	7 Java programs
7.	[67]	Tesler	Statement & Branch coverage, Fault detection	Testler	N/A	N/A	N/A	15 projects
8.	[68]	Constraint optimization	Feature coverage	Flower/C.	Ubuntu Linux	Quad core 2.8GHz	8MB cache per core	9 random datasets
9.	[69]	Similarity centered approach	Fault localization	GCOV	Ubuntu	Core i2	4 GB	Siemens, Unix utility
10.	[70]	Multi-objective approach	Statement & Branch coverage, Fault detection, Execution time reduction	Selenium	N/A	N/A	N/A	Web apps
11.	[71]	Dynamic Programming	Fault detection	MILU	N/A	N/A	N/A	TACS program from SIR. Triangle problem
12.	[72]	ReduceDomains algorithm	Path coverage, Execution time reduction	N/A	Windows 7	Core i3-2310	2 GB	55,566 test cases

Sivaji *et al.* [70] propose a multi-objective technique for efficiently and effectively detecting faults in regression testing. The validation of proposed approach done on several web applications shows that it can select test cases accurately with reduction in execution time and increase in Average Percentage of Faults Detected (APFD). Jatana *et al.* [71] introduce a dynamic programming based approach for test suite reduction. This approach is conceptually similar to that used by Floyd–Warshall's algorithm. Evaluation performed on TCAS code and triangle problem depicts that proposed approach runs in polynomial time and it is effective in finding a minimized test suite that can detect faults in the program under test. The proposed method by Singh and Singh [72] is based on the parallel execution to cover all independent paths. According to the results, the proposed method achieves lower execution time using parallelism as well as reduction in test cases.

From **Table 11**, it is analyzed that out of twelve (12) studies there are five (5) researches in which tools are developed or proposed i.e. DetReduce, Nemo, TITAN, I-ReTest and Tesler. There are two studies that deals with fault localization [69] [71]. Finally, mutation coverage [65], requirement coverage [63] and feature coverage [68]. Multi-objectives [70], [72] are taken into consideration by remaining studies.

F. TEST SUITE OPTIMIZATION TOOLS

From 58 selected papers, we have overall identified 32 tools in which 25 tools are existing and 7 tools are newly developed or customized.

1) EXISTING TOOLS

In **Table 12**, existing tools which are used in the selected research studies are listed. The following parameters are summarized: 1) **Name** signifies the name of tool that is

TABLE 12. Identified tools for test suite optimization.

Sr. No	Tool Name	Reference	Purpose	Language support	Open source
1	GCOV	[16] [35] [69]	Statement coverage	C, FORTRAN	✓
2	EMMA	[17] [23]	Fitness testing/ Statement coverage	Java	✓
3	Monkey	[17]	Test case generation	N/A	✓
4	KLEE	[19]	Determine set of in-scope entities	N/A	✓
5	MILU	[19] [71]	Mutated versions generation	C	✓
6	PIT	[20]	Line and mutation coverage	Java	✓
7	JUnit	[23] [57] [60]	Fitness testing/ Branch coverage/ Unit testing	Java	✓
8	Randoop	[25]	Test suite generation	Java	✓
9	Spark	[27]	PGAS implementation	N/A	✓
10	Mockito	[28]	Mock objects creation	Java	✓
11	EvoSuite	[30]	Algorithm implementation	N/A	✓
12	QuestaSim	[31]	Execution of UVM-based verification	N/A	✓
13	Callgrind	[47]	It generates a log that encompasses data related to the execution of system under test	Linux programs	✓
14	MHBG_TCS	[51]	Optimal test suite	C++, Java ,C#	✗
15	jMetal	[55]	CBGA-ES implementation	Java	✓
16	Codepro Analytix	[56]	Test case generation	Java	✓
17	Eclemma	[56]	Coverage, complexity calculation	Java	✓
18	JaCoCo	[57] [58]	Branch coverage	Java	✓
19	RAISE	[58]	Implementation	Java	✓
20	DBUnit	[60]	Unit testing of code and DB	Java	✓
21	Graphviz	[66]	Graphical visualization	DOT language	✓
22	ANTLR	[66]	I-ReTEST component development	Java	✓
23	MuClipse	[66]	Fault injection	Java	✓
24	Flower/C	[68]	Implementation of constraint optimization models	N/A	✗
25	Selenium	[70]	Testing of software application	C#, Java, Perl, PHP, Python and Ruby	✓

used to achieve the optimized test suite, 2) **Reference** of each corresponding study is presented for further details, and 3) **Purpose** represents the basic aim for the usage of tool e.g. statement coverage, test case generation, branch coverage, mutation coverage, complexity calculation etc. 4) **Language support**signifies the programming languages which are supported in the corresponding tool. 5) **Open**source shows that whether the tool is freely available for users or not.

From **Table 12**, we have scrutinized four tools that are used in the greedy algorithm based category (i.e. Android Monkey, KLEE, MILU, PIT). Seven tools (i.e., EvoSuite, EMMA, Randoop, QuestaSim, Spark, Callgrind and Mockito) are used in the meta-heuristic category. We have identified two tools (i.e. MHBG_TCS tool, jMetal) in the hybrid category. Five tools have been extracted from clustering category (i.e. Codepro Analytix, CUTER, RAISE, JaCoCo, DBUnit). In the general category, Flower/C, ANTLR, Graphviz, Selenium and MuClipse are used. JUnit is used in both genetic and clustering category while GCOV is used in greedy, meta-heuristic as well as general category.

2) PROPOSED TOOLS

In **Table 13**, seven tools are listed which are either developed or customized to attain optimization of test suites which helps in the testing of software products. We have summarized the **name** of the tool, **reference** of study from which tool is identified, the specific **purpose** of the tool and its availability as **open source,** in the table above.

TABLE 13. Proposed tools for test suite optimization.

Sr. No	Tool Name	Ref.	Purpose	Open Source
1	MOTestGen	[41]	Convergence, coverage performance extraction	✕
2	CUTER	[57]	Test-suite reduction	✕
3	DetReduce	[61]	Test-suite reduction	✓
4	TITAN	[62]	Test suite optimization	✕
5	Nemo	[64]	Optimal solution generation	✓
6	I-ReTEST	[66]	Effectiveness measurement /Regression test case selector	✕
7	Testler	[67]	Minimization of substantial redundancies in test cases statement	✕

In selected research studies, seven tools are modified/ developed to accomplish optimal test suite. Gopi *et al.* [41] develop MOTestGen tool for the production of optimal test data and extraction of their coverage and convergence performance. Coviello *et al.* [57] propose a prototype tool as an Eclipse plug-in named CUTER i.e. ClUstering-based TEst suite Reduction, for inefficient test suite reduction. Choi *et al.* [61] introduce a prototype tool named DetReduce for implementing test reduction algorithm. It can be used for GUI supporting platforms but mainly it works for android applications. An average factor of 14.7× in running time and 16.9× in size during 14.6 hours is noticed during test suite reduction by DetReduce. D Marijan *et al.* [62] use TITAN tool for timely detection of faults, reduction in redundancies and quality as well as status visualization of the testing process.

Lin *et al.* [64] implement a prototype tool called Nemo for finding optimal solution of multi-criteria test-suite minimization. Panigrahi and Mall [66], develop I-ReTest (Improved-Regression TEST case selector) for implementation of I-RTS methodology. Vahabzadeh *et al.* [67] employ Testler for evaluation of their approach. Results depict that 43% of test statement redundancies are removed which mainly result in a 52% decrease in partially redundant tests. It also reduces the execution time up to 37% without affecting the actual statement coverage, test assertions, branch coverage, and fault detection capability.

3) PLATFORM SUPPORTS

It is noted that different platform supports and frameworks are being used along with test suite optimization tools in selected research studies. From the 58 selected studies, we have found fourteen platform supports. In **Table 14**, we have summarized the name of supporting platform which is used for the implementation of optimization algorithms, 2) **References** of studies from which platform support is identified, 3) **Purpose** of each platform support and 4) **Total**number of studies from which these platform supports are extracted.

From **Table 14**, it is obvious that several frameworks, software and platforms are utilized in selected literature for performing different tasks of test suite optimization process. MATLAB is used in four studies for the computation task. Xposed framework, Dev C++ IDE, DbUtils, GNU compiler and visual studio are used in literature for performing tasks related to test suite optimization. NetBeans IDE is used in six studies while Eclipse is used in four studies for application development. JRE is exploited in one research whereas JDK is used in four studies exploited performing compilation. Spring and Apache CXF are used for development of TITAN tool while Java swing is used for development of user interface for I-ReTEST tool. Lastly, in one study Hadoop is used for processing of data.

In most of the studies i.e. six (6) researches, NetBeans IDE is exploited as it provides support for the development of Java application.

G. COMPARISON OF CATEGORIES

The comparison among different optimization algorithms is based on performance metrics that are widely adopted in the literature. In the context of test case generation, these metrics give a reasonable estimation of the effectiveness and efficiency of the test case optimization techniques. In this SLR, we intend to perform a comparison of 58 selected studies on the basis of all journals. But, only one journal paper lie is clustering as well as hybrid category. Therefore, after performing a careful analysis, eighteen high-quality studies are considered that provide comprehensive details about their test suite optimization approach. So, in order to determine the strengths and weaknesses of defined categories, we have conducted a comparative analysis on eighteen (18) research studies i.e. three (3) studies from each category/sub-category.

As shown in **Table 15**, we evaluate: 1) Objective type i.e. whether the study deals with multi-objective or single-objective optimization problem, 2) Coverage criteria of the corresponding study i.e. path coverage, branch coverage, statement coverage, 3) Fault detection effectiveness of

TABLE 14. The platform supports for test suite optimization.

Sr. No	Platform Support	Purpose	Reference	Total
1.	MATLAB	Computation	[24] [26] [40] [49]	4
2.	Xposed framework	Customization	[17]	1
3.	NetBeans IDE	Application development	[39] [41] [43] [42] [45] [50]	6
4.	DbUtils	Collect info about fault associated to failures	[33]	1
5.	Dev C++ IDE	Program compilation	[34]	1
6.	JRE 8.0	Run a compiled Java program	[45]	1
7.	Eclipse	Application development	[56] [66] [70] [72]	4
8.	Spring	TITAN implementation	[62]	1
9.	Apache CXF	Services development	[62]	1
10.	JDK	Programs creation and compilation	[44] [45] [27] [43]	4
11.	Hadoop 2.4	Data processing	[27]	1
12.	Java Swing	Development of I-ReTEST UI	[66]	1
13.	GNU Compiler	Collection of statement coverage information	[47]	1
14.	Visual Studio	For developing computer programs, as well as websites, web apps, web services and mobile apps.	[47]	1

proposed methodology i.e. whether the corresponding study is effective in terms of fault detection or not, 4) Cost/benefit analysis of proposed techniques and, 5) Execution time i.e. whether the proposed approach results in less time taken for execution. If the corresponding study satisfies the comparison parameter, it is marked with ✓.

As evident from the literature, researchers have proposed approaches that consider more than one kind of test requirement. However, most of these existing methods are static and single-objective. In single objective optimization, selected test suite may skip many important test cases and resultant test suite is not optimized, suitable and safe for use [73]. It focuses either on effectiveness (i.e., to obtain minimal RS size) or cost (i.e., to improve the fault detection capability loss), but not both. However, as required by real-world problems researchers need to incorporate multi-objective optimization by taking into account multiple alternatives to find an optimal tradeoff between cost and effectiveness as focused by multi-objective optimization [74]. Another major concern of single-criterion minimization is that minimizing a test suite could severely compromise its ability to reveal faults [64]. On the other hand, multi-objective optimization is often a set of solutions that do not dominate each other, thereby forming the trade-offs between constraints. The insight into the trade-offs may provide additional information that is hard to obtain otherwise [75]. Therefore, results suggest that the two-objective versions outperform the single-objective ones.

Because of simplicity and ease of implementation, most research studies utilize a greedy algorithm for testing purposes. Greedy Algorithm is an implementation of the 'next best' search philosophy. It works on the principle that the element with the maximum weight is taken first, followed by the element with the second highest weight and so on, until a complete, but possibly sub–optimal solution has been constructed [76]. Many studies reported that essential expectations are satisfied by the greedy approach. In spite of this, a greedy algorithm carries a crucial weakness: it stuck into local search space and produce sub-optimal solutions [29]. It can work effectively for the nearest solutions but for global solutions, it is not successful. The basic reason behind this limitation is that the greedy approach greedily picks the best test case one at a time regarding the defined objective until the termination conditions are satisfied. Sometimes, the greedy approach assigns equal weights to each objective those results in the conversion of multi-objective problem into a single objective one. This result in a loss of optimal solutions that hold the same quality [55]. The greedy algorithm attains a higher percentage of killed mutants as compared to the IP approach but reduction ability of greedy is less than IP. It can be defined by the fact that more number of test cases are executed in a greedy approach that results in a higher number of killed mutants [17]. As depicted in Table 15, greedy algorithm based studies have mostly focused on increasing the fault detection effectiveness. Greedy based techniques also

TABLE 15. Comparative analysis of selected studies.

Ref. #	Objective Type	Coverage				Fault detection	Cost/Benefit	Execution time
		Path	Req.	Branch	Statement			
				GREEDY ALGORITHM				
15.	Single-objective	✗	✗	✗	✓	✓	✗	✗
18	Single-objective	✗	✗	✗	✗	✓	✓	✗
21	Multi-objective	✗	✗	✗	✗	✓	✓	✓
				META-HEURISTIC				
				Genetic Algorithm				
25	Multi-objective	✗	✗	✗	✗	✗	✗	✓
26	Multi-objective	✗	✓	✗	✗	✓	✓	✓
27	Single-objective	✗	✗	✗	✗	✗	✗	✓
				Ant Colony Algorithm				
35	Single-objective	✗	✓	✗	✗	✗	✓	✗
36	Multi-objective	✗	✗	✗	✗	✓	✓	✓
37	Single-objective	✓	✗	✗	✗	✗	✗	✗
				Other Meta-Heuristic Techniques				
38	Multi-objective	✓	✗	✗	✗	✓	✓	✓
40	Single-objective	✗	✗	✓	✗	✗	✗	✓
46	Multi-objective	✗	✗	✗	✗	✓	✗	✓
				HYBRID				
51	Single-objective	✓	✗	✗	✗	✗	✗	✓
52	Multi-objective	✗	✓	✗	✗	✓	✓	✗
53	Single-objective	✓	✗	✗	✗	✗	✗	✗
				CLUSTERING				
56	Single-objective	✗	✗	✓	✗	✗	✗	✗
57	Single-objective	✗	✗	✗	✓	✗	✗	✗
58	Single-objective	✗	✗	✗	✓	✗	✗	✗

result in good cost reduction capability. However, the competency of fault detection gets worse in general and the time taken for performing reduction task monotonically increases for these techniques as soon as the complexity of test suite increase [18].

Metaheuristics has been developed to tackle nonlinear, complex optimization problems for which exact optimization techniques fail to offer satisfactory results. It is implemented through an iterative generation process that guides subordinate heuristics in exploration and exploitation of the search space to efficiently find near-optimal solutions. Metaheuristic algorithms are not problem and domain specific, and they are capable of locating good quality solutions in a relatively shorter time, compared to traditional optimization techniques [77]. It is also evident from the comparative analysis (Table 15) that meta-heuristics based approaches primarily aim to reduce the execution time.

Different researchers use an evolutionary search-based approach named Genetic Algorithm (GA) for the purpose of software testing. Genetic algorithm is an evolutionary approach to computing, which has the ability to determine appropriate approx. solutions to optimization problems. It uses the principles of selection and evolution to produce solution for various complex problems. Therefore, it must be noted that genetic algorithms are not always the best choice in random scenarios. Sometimes they might take quite a while to run and are therefore not always feasible for real time use. They are, however, one of the most powerful methods with which high quality solutions are created quickly to a problem. Genetic algorithms are a very general algorithm and so they will work well in any search space [78].

The genetic algorithm provides a good result in terms of reduced execution time. But, it also has some shortcomings like other methods. The genetic strategy attempts to support higher interaction strengths but for small strengths, it exhibits lower efficiency as compared to other techniques [24]. For getting a better result in benchmarks, parameter settings play a crucial role. But, choosing a good parameter setting for the genetic algorithm is a big issue. Another limitation of the genetic algorithm is its intrinsic randomness i.e. a stable result cannot be obtained by a single run [27], [30]. So as to alleviate that threat and getting the best outcome, executions related to genetic algorithm based experiments must be executed for a large number of times and default parameter settings should be used [26], [30].

Ant Colony Optimization (ACO) is also a meta-heuristic approach, which tries to find the smallest path from all test cases, but it does not cover all the test cases, which are required [36]. Like a greedy algorithm, ACO also gets trapped in local optimum and its convergence rate is slow to some extent [35], [36]. This problem of convergence can be solved with the help of improved quantum ant colony algorithm and modified ant colony optimization. Another problem with ACO algorithms is its biased elite solutions, which result in the diversity loss [37]. To enhance the performance of these approaches, we can adopt some strategies e.g. for updating parameters at every iteration. However, these strategies are also limited [39]. Global optimum is achieved in a meta-heuristic approach through exploration and exploitation.

Exploitation attempts to get closer to the best solution(s) iteratively but this can result in a local optimum problem. Exploration can be used to negotiate this problem by exploring different areas in the search space that may contain the global optimum. But, this can divert the search away from the current best solution. However, the combination of exploitation and exploration can make a negotiation in order to reach an appropriate solution without being stuck at a local optimum. Another Meta-heuristic technique named Moth Flame Optimization (MFO) can achieve full branch coverage, but for the sake of accuracy of the statistical results every experiment has to be holding many times and it does not put any restrictions on array size [40].

Many researchers utilize k-means clustering algorithm for test suite optimization but this algorithm is unstable and seldom considers the coverage rate of such test cases. Therefore, the k-medoids clustering algorithm characterized by cyclomatic complexity and code coverage rate is preferred. A greedy algorithm can also be utilized in this process to streamline test suite while guaranteeing the cases coverage rate and the error detection rate. This method features a higher coverage rate with lower complexity under the streamlined test suite of the same quantity [56]. Two existing clustering techniques, Affinity Propagation and Dissimilarity-based Sparse Subset Selection can also be used to drastically reduce the original test suite while retaining a good mutation score [59]. In several applications, hierarchical clustering is preferred because it produces deterministic

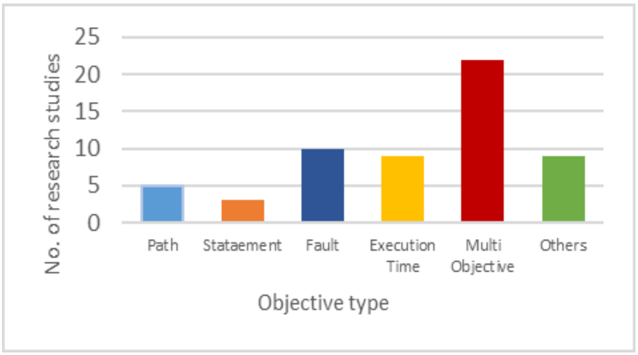

FIGURE 5. Distribution of selected studies w.r.t objective type.

output and leads to a greater reduction in test suite size at the expense of a small loss in fault-detection capability. It produces results in the form of a dendrogram. The cuts of dendrogram can be customized according to requirements e.g. for obtaining a small-sized test suite the value of the cut level on dendrogram must be lower while if the objective is to get a low fault-detection loss, the value for the cut level must be higher. But, in the case of supervised clustering, it is sometimes required to have added information along with the data, such as the sizes and number of the clusters to be formed, which makes the process costly and puts a limitation on the procedure as well [60].

As evident from this comparative analysis, every approach has its strengths and weaknesses. There is no single approach that is superior to all test suite optimization problems. Like, the genetic algorithm provides a good result in terms of reduced execution time but it must be executed for a large number of times in order to obtain stable results. Greedy based techniques result in good cost reduction capability and they increase fault detection rate as well. But the competency of fault detection gets worse in general, and the time complexity monotonically increases for it as soon as the complexity of test suite increase. Therefore, it entirely depends on the given test suite optimization problem that which approach should be taken into consideration for achieving optimization of the test suite.

From 58 selected studies, it is analyzed that researchers have used different type of objectives for the purpose of test suite optimization e.g. path coverage, statement coverage, fault coverage, executions time reduction. In **Figure 5**, the distribution of all studies with respect to objective type is presented. The percentages of algorithms' performances in terms of size reduction, execution time and fault detection extracted from selected studies is given in **Figure 6**, **Figure 7** and **Figure 8**, respectively.

The results presented in [15] shows that it can effectively optimize the test suite with 100% path coverage along 50% reduction in test suite size. But, the basic reason behind this higher coverage rate is that the size of test data is quite small. Therefore, it is essential to perform the experimentation on larger dataset in order to validate the wider applicability of these approaches. Jabbarvand *et al.* [17] employed eCoverage

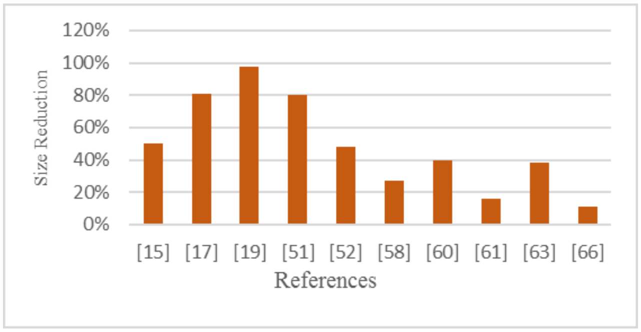

FIGURE 6. Distribution of selected studies w.r.t size reduction %.

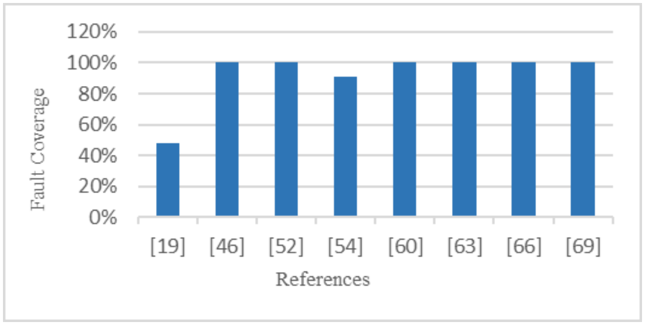

FIGURE 7. Distribution of selected studies w.r.t fault coverage %.

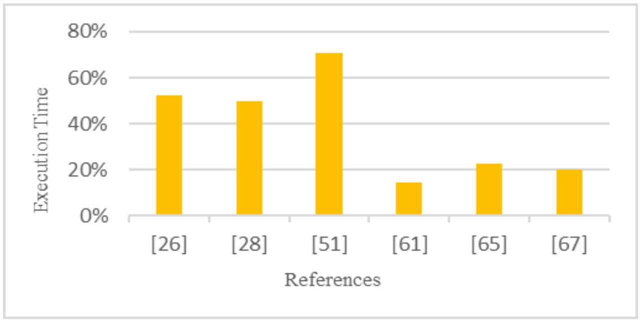

FIGURE 8. Distribution of selected studies w.r.t time reduction %.

for test suite reduction. Results indicate that 81% reduction in size with 67% eCoverage is achieved using the proposed approach. Miranda and Bertolino [19] used three datasets namely grep, gzip and sed for experimentation. For grep, they obtained 88.7% reduction in test suite size with 30.72% impact on Fault Detection Capability (FDC). In the case of gzip and sed, 97.3% and 97.4% reduction in terms of size and 36.16% and 47.62% fault detection capability is attained respectively. Coviello *et al.* [58] employed hierarchical agglomerative clustering algorithm for reducing the size of test suite on the basis of statement coverage. Results depict that 27.06% reduction in test suite size with 95% statement coverage is achieved using the proposed approach. The proposed approach by Rosero *et al.* [60] exhibits an improvement in the effectiveness of a regression test with respect to a complete regression test. In general, on average, this approach reduced the number of test cases to 14% in product 1 (Estafeta) and 40% in product 2 (Silabo), both of them with a fault detection capacity of 100%. The proposed approach by Choi *et al.* [61] named DetReduce reduces a test-suite by an average factor of 16.9× in size and 14.7× in running time. In [63], Liu *et al.* presents a test suite reduction approach based on probability models in regression testing. The average test suite reduction ratio of this approach reaches 38.27% (on average) with 100% FDR. The proposed technique by Panigrahi and Mall [66] reduces the regression test suite size by 11.25 % as compared to a related approach, without degrading the fault revealing effectiveness. The proposed based harmony search algorithm by Choudhary [46] is able to detect all the faults with respect to different test suite sizes. The approach presented by Anwar *et al.* [52] can reduce the size of test suite up to 48% with no loss in FDR. Saraswat and Singhal [54] detected 89.78% faults for case study1 using the proposed approach while for the second case study 90.85% faults are identified. In [69], Fu *et al.* presents an approach to improve fault localization and the results suggest that it can detect 100% faults.

Several studies have used execution time reduction as the main objective for the purpose of optimizing the test suites. Like, Garousi *et al.* [26] presented an approach that can achieve 100% requirement coverage with 52.1% reduction in execution time. Similarly, Turner *et al.* [28] employed execution time reduction as the optimization objective. The proposed approach results in ∼50% reduction in terms of execution time. The MHBG_TCS technique proposed by Singhal *et al.* [51] can achieve almost 71% reduction in execution time with more than 80% reduction in size of test suite. The optimization technique given by Palomo-Lozano *et al.* [65] results in 22.59% reduction in execution time. Lastly, the analysis of approach given by Vahabzadeh *et al.* [67] shows that it can reduce 52% partly redundant test cases with a reduction of 20% in execution time.

Experimentation plays a vital role in providing insights about the quality of a research study. Kitchenham *et al.* [14] emphasized on reporting guidelines, which are useful to all types of empirical studies. However, most of the researchers have not used any standard to systematically conduct and report the outcomes of their experiments related to test suite optimization and thus may pose validity threats related to their results. The current literature lacks information on exact percentages of results achieved in terms of reduced size, fault detection and execution time reduction etc.

IV. ANSWERS OF RESEARCH QUESTIONS

RQ1: For improving the test suite optimization process, what are the primary approaches reported so far?

Answer: 58 significant researches published from 2016 to 2019 have been characterized as per inclusion and exclusion criterion (**Section 2.2.1**). These researches are grouped into six subsequent categories. The details are as follows:

- Seven research studies (12.06%) have been found in the Greedy Algorithm Category (**Section 3.1**)
- Twenty-eight researches (48.27%) have been found and enumerated in the Meta-Heuristic Category (**Section 3.2**) and it is sub-categorized into genetic algorithm that comprises of 13 studies (**Section 3.2.1**), ant colony algorithm which incorporates 4 studies (**Section 3.2.2**) and other meta-heuristic algorithms i.e. Bat, Dragon-fly, Moth flame optimization and Flower algorithm etc. embodies 11 studies (**Section 3.2.3**)
- Six research studies (10.34%) have been found and listed in the Hybrid Category (**Section 3.3**)
- Five research studies (8.62%) have been found and listed in the Clustering Category (**Section 3.4**)
- Twelve research studies have been found in the General category (**Section 3.5**)

It is evident that most of the research studies are associated with different meta-heuristic algorithms in order to achieve optimization of test suites. On the other hand, clustering based test suite reduction has obtained less consideration by researchers, as out of 58 researches selected from widely known databases only 8.62% studies fall in this category.

RQ2: What are the primary tools employed/developed for the process of test suite optimization?

Answer: By performing the systematic review of the literature, we have identified 25 existing tools that support optimization process as given in **Table 12**. Evaluation of these tools is also done by considering significant tool parameters and their classification is done according to defined categories (**Section 2.1**). From **Table 12**, we have scrutinized four tools that are used in the greedy algorithm based category (i.e. Android Monkey, KLEE, MILU, PIT). Seven tools (i.e., EvoSuite, EMMA, Randoop, QuestaSim, Callgrind, Spark, and Mockito) are used in the meta-heuristic category. We have identified two tools (i.e. MHBG_TCS tool, jMetal) in the hybrid category. Five tools have been extracted from clustering category (i.e. Codepro Analytix, CUTER, RAISE, JaCoCo, DBUnit). In the general category, Flower/C, ANTLR, Graphviz, Selenium and MuClipse are used. JUnit is used in both genetic and clustering category while GCOV is used in greedy, meta-heuristic as well as general category.

In selected research studies, seven tools are proposed/ developed (**Table 13**) attain an optimal test suite. Gopi *et al.* [41], developed MOTestGen tool for the production of optimal test data and extraction of their coverage and convergence performance. Coviello *et al.* [57], proposed a prototype tool as an Eclipse plug-in named CUTER i.e. ClUstering-based TEst suite Reduction, for inefficient reduction of the test suite. Choi *et al.* [61], introduced a prototype tool named DetReduce for implementing test reduction algorithm. Marijan *et al.* [62], used TITAN tool for timely detection of faults, reduction in redundancies and quality as well as status visualization of the testing process. Jun-Wei Lin *et al.* [64], implemented a prototype tool Nemo for finding the optimal solution of multi-criteria test-suite minimization. Panigrahi and Mall [66], developed I-ReTest i.e. Improved-Regression TEST case selector, for implementation of I-RTS methodology. Vahabzadeh *et al.* [67], used Testler for evaluation of their approach.

RQ3: What are the supporting platforms for optimization tools used in literature?

Answer: During this SLR, we performed a comprehensive analysis of test suite optimization tools and identified 14 supporting platforms (**Table 14**) that have been used along identified tools. These supporting platforms include Java 1.7, JDK 1.7, Netbeans, Visual studio, DButils, Dev C++ IDE, GNU compiler, Eclipse, Spring, Hadoop, Matlab, Xposed framework and Apache CXF. Xposed framework is used for customization on one research study. NetBeans IDE is used in six studies while Eclipse is used in four studies for application development. JRE is exploited in one research whereas JDK is employed in four studies for performing compilation task. Similarly, Matlab is exploited in four studies for the purpose of computation. Spring and Apache CXF are used for development of TITAN tool while Java swing is used for development of user interface for I-ReTEST tool. GNU compiler is used in one study for the purpose of collecting statement coverage information. Dev C++ IDE and visual studio are also used in one study for performing compilation tasks. In one study, DButils is used for collecting information about a fault associated to certain failures. Lastly, in one study Hadoop is used for processing of data. It is evident that in most of the studies i.e. six (6) researches, NetBeans IDE is exploited as it provides support for the implementation of Java based test suite optimization approaches.

RQ4: How to improve modern test suite optimization approaches to accommodate the future technological advancements?

Software industry is rapidly growing with the passage of time. To facilitate the testing of software, different optimization techniques have been proposed by researchers. As seen from **Table 6**, **Table 7**, and **Table 8**, most studies have incorporated different meta-heuristic approaches for optimizing the test suites. Similarly, greedy algorithm based reduction techniques (**Table 5**) are also an active area of research in software industry. However, less attention is given to hybrid (**Table 9**) and clustering based approaches (**Table 10**).

Although these approaches have achieved significant results in terms of optimization but there are several future directions to cope with the rapid advancements in technology. Basically, regression testing focus on generation of test input and monitoring output for anticipated results and failures. Current AI (Artificial Intelligence) methods such as classification and clustering algorithms rely on just this type of primarily repetitive data to train models to forecast future outcomes accurately. These methods show the greatest promise for automating software testing today as they are capable to recognize complex patterns and make intelligent decisions based on training data. Similarly, standard machine learning and more specifically deep learning methods can be trained with data generated by cycles of user input, combined with the corresponding output of the system under test. For example, reinforcement learning is a technique that is well-tuned to design an adaptive method capable to learn from its experience of the execution environment [79]. Likewise, the neural network is shown to be a promising method of testing a software as it is also capable of learning new versions of evolving software [80]. The value of training deep learning models to forecast user input and system output is incremental, and grows increasingly accurate as data accumulates in each test cycle. Therefore, in future, different AI and machine learning approaches can be integrated with optimization algorithms to increase the effectiveness of software testing.

V. DISCUSSION AND LIMITATIONS

The main goal of this literature review is the classification of test suite optimization researches and their evaluation on the basis of different parameters in order to provide the details of the latest approaches and tools that support optimization. Grouping of optimization approaches into four categories i.e. genetic, meta-heuristic, hybrid and clustering is done. The main difference in these categories is the use of different algorithms. Greedy based approaches typically employ different types of greedy algorithms, meta-heuristic based category use various meta-heuristic algorithms such as ant-colony and genetic algorithm, and hybrid approaches exploit two or more algorithms in order to give improved and effective results by utilizing the strengths of different methods. Similarly, clustering based category make use of different type of clustering algorithms and techniques to achieve optimal test suite.

Classification of test suite optimization approaches is shown in **Table 4**. Amongst fifty-two (58) selected research studies, seven (7) fall into greedy algorithm based category, meta-heuristic based category includes twenty-eight (28) studies from selected literature and six (6) researches utilize hybrid approach for test suite optimization. However, less attention has been given to clustering based approaches i.e. five (5) studies are found in this category. Subsequently, 32 leading tools that support different optimization activities have been presented. Besides tools evaluation, this research study also examines the utilization of different supporting platforms for the development of optimization approaches. Consequently, a comprehensive comparative analysis is also performed on the identified optimization approaches by considering various important optimization parameters, such as the type of objective i.e. single or multi-objective problem, coverage criteria, analysis of time/cost benefit and execution time.

It is obvious that each approach has some potential benefits as well as shortcomings. Like, genetic algorithm results in reduced execution time, while greedy based approaches provide better fault coverage in addition to reduced cost. But, a large number of execution is required for genetic algorithm whereas time complexity increase in case of greedy approach. These approaches are utilized according to a given optimization problem. However, the extension can be done to incorporate different fault localization techniques, e.g. control flow based and information flow based fault localization along multi-criteria optimization algorithms. For accessing the performance of the proposed approach, different effectiveness measures, such as size of the reduced test suite, cost and coverage are used by researchers. But, it is essential to examine the performance of proposed approaches with cost as well as effective measures for broader acceptance. Subsequently, in most of the selected studies, publically available datasets e.g. Siemens and Space Programs taken from Software Infrastructure Repository (SIR), are used. But, the implementation of proposed approaches on large scale industrial projects is essential to measure their effectiveness. In addition, most of the selected research studies do not incorporate the proper details about the experimental setup required for the implementation of their proposed approach. In this context, different test suite optimization approaches, platform supports and important tools used for the development of these approaches are publicized under this single research which is rarely available to the best of our knowledge.

Although we entirely abide by the strategies of SLR [14] and strictly obeyed the developed review protocol for incorporating current research studies as much as possible, we're not able to find considerable amount of research studies that are published during 2016-2019. The basic reason is that we have chosen four prominent scientific repositories for this systematic review i.e. ACM, Springer, IEEE and Science Direct. Subsequently, the search process is performed in only these four repositories from 2016-2019. Studies related to testing suite optimization are published in several other databases as well, but the reliability of those researches is uncertain. Therefore, any research study which is published in other scientific databases is not taken into account. We consider that the elimination of such research studies does not conspicuously affect the results of this SLR.

VI. CONCLUSION

This research study presents state-of-the-art approaches, tools as well as platforms that support test suite optimization published during 2016-2019. In order to achieve this goal, a systematic review of the literature has been performed and 58 research studies are identified. Because of different optimization approaches, selected studies are then grouped into five different categories. Consequently, a comprehensive comparative analysis is performed on the identified optimization approaches by considering various important optimization parameters, such as the type of objective i.e. single or multi-objective problem, coverage criteria, analysis of time/cost benefit and execution time. Subsequently, 32 leading tools that support different optimization activities have been presented. Besides tools evaluation, this research study also examines the utilization of 14 different supporting platforms for the development of optimization approaches. Thus, test suite optimization approaches, platform supports and important tools for the development of these approaches are publicized under single research which is rarely available to the best of our knowledge. This research will definitely facilitate researchers, practitioners and developers to select appropriate optimization approaches and tools along platforms according to their requirements. Furthermore, it is noted that several optimization studies aim at solving the problem of single-objective optimization. Therefore, researchers should focus on test suite optimization based on solving the multi-objective problem, as multi-objective versions outperform the single-objective ones. Moreover, less attention has been given to clustering based approaches. In future, we recommend to explore machine learning and artificial intelligence based approaches for test suite optimization as they are capable to make intelligent decisions based on training data.

REFERENCES

[1] I. Sommerville, *Software Engineering* (International Computer Science Series). Reading, MA, USA: Addison-Wesley, 2004.

[2] X. Zhang, H. Shan, and J. Qian, "Resource-aware test suite optimization," in *Proc. 9th Int. Conf. Qual. Softw.*, 2009, pp. 341–346.

[3] R. Mall, *Fundamentals of Software Engineering*. New Delhi, India: PHI Learning, 2014.

[4] R. Gupta and M. L. Soffa, "Employing static information in the generation of test cases," *Softw. Test., Verification Rel.*, vol. 3, no. 1, pp. 29–48, 1993.

[5] P. McMinn, "Search-based software test data generation: A survey," *Softw. Test., Verification Rel.*, vol. 14, no. 2, pp. 105–156, 2004.

[6] M. Dorigo and G. Di Caro, "Ant colony optimization: A new meta-heuristic," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 2, 1999, pp. 1470–1477.

[7] X.-Y. Ma, Z.-F. He, B.-K. Sheng, and C.-Q. Ye, "A genetic algorithm for test-suite reduction," in *Proc. IEEE ICSMS*, Oct. 2005, pp. 133–139.

[8] Z. Anwar and A. Ahsan, "Comparative analysis of MOGA, NSGA-II and MOPSO for regression test suite optimization," *Int. J. Softw. Eng.*, vol. 1, no. 7, pp. 41–56, 2014.

[9] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.

[10] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," *ACM SIGSOFT Softw. Eng. Notes*, vol. 31, no. 1, pp. 35–42, 2006.

[11] M. L. Zepeda-Mendoza and O. Resendis-Antonio, "Hierarchical agglomerative clustering," in *Encyclopedia of Systems Biology*. New York, NY, USA: Springer, 2013, pp. 886–887.

[12] G. Kumar and P. K. Bhatia, "Software testing optimization through test suite reduction using fuzzy clustering," *CSI Trans. ICT*, vol. 1, no. 3, pp. 253–260, 2013.

[13] S. U. R. Khan, S. P. Lee, N. Javaid, and W. Abdul, "A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines," *IEEE Access*, vol. 6, pp. 11816–11841, 2018.

[14] B. Kitchenham, "Procedures for performing systematic reviews," Keele Univ., Keele, U.K., Tech. Rep. EBSE-2007-01, Version 2.3, 2004, pp. 1–26, vol. 33.

[15] S. Singh and R. Shree, "A combined approach to optimize the test suite size in regression testing," *CSI Trans. ICT*, vol. 4, nos. 2–4, pp. 73–78, 2016.

[16] K.-C. Wang, T.-T. Wang, and X.-H. Su, "Test case selection using multi-criteria optimization for effective fault localization," *Computing*, vol. 100, no. 8, pp. 787–808, 2018.

[17] R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek, "Energy-aware test-suite minimization for Android apps," in *Proc. 25th Int. Symp. Softw. Test. Anal.*, 2016, pp. 425–436.

[18] C.-T. Lin, K.-W. Tang, J.-S. Wang, and G. M. Kapfhammer, "Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity," *Sci. Comput. Program.*, vol. 150, pp. 1–25, Dec. 2017.

[19] B. Miranda and A. Bertolino, "Scope-aided test prioritization, selection and minimization for software reuse," *J. Syst. Softw.*, vol. 131, pp. 528–549, Sep. 2017.

[20] A. Shi, A. Gyori, S. Mahmood, P. Zhao, and D. Marinov, "Evaluating test-suite reduction in real software evolution," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, 2018, pp. 84–94.

[21] X. Wang, S. Jiang, Gao, X. Ju, R. Wang, and Y. Zhang, "Cost-effective testing based fault localization with distance based test-suite reduction," *Sci. China Inf. Sci.*, vol. 60, no. 9, 2017, Art. no. 092112.

[22] R. Khan, M. Amjad, and A. K. Srivastava, "Optimization of automatic generated test cases for path testing using genetic algorithm," in *Proc. 2nd Int. Conf. Comput. Intell. Commun. Technol. (CICT)*, 2016, pp. 32–36.

[23] S. Kothari and A. Rajavat, "Minimizing the size of test suite using genetic algorithm for object oriented program," in *Proc. Int. Conf. ICT Bus. Ind. Government (ICTBIG)*, 2016, pp. 1–5.

[24] S. Esfandyari and V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," *Inf. Softw. Technol.*, vol. 94, pp. 165–185, Feb. 2018.

[25] A. Schuler, "Application of search-based software engineering methodologies for test suite optimization and evolution in mission critical mobile application development," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 1034–1037.

[26] V. Garousi, R. Özkan, and A. Betin-Can, "Multi-objective regression test selection in practice: An empirical study in the defense software industry," *Inf. Softw. Technol.*, vol. 103, pp. 40–54, Nov. 2018.

[27] R.-Z. Qi, Z.-J. Wang, and S.-Y. Li, "A parallel genetic algorithm based on spark for pairwise test suite generation," *J. Comput. Sci. Technol.*, vol. 31, no. 2, pp. 417–427, Mar. 2016.

[28] A. J. Turner, D. R. White, and J. H. Drake, "Multi-objective regression test suite minimisation for mockito," in *Proc. Int. Symp. Search Based Softw. Eng.* Cham, Switzerland: Springer, 2016, pp. 244–249.

[29] A. Yamuç, M. Ö Cingiz, G. Biricik, and O. Kalıpsız, "Solving test suite reduction problem using greedy and genetic algorithms," in *Proc. 9th Int. Conf. Electron., Comput. Artif. Intell. (ECAI)*, 2017, pp. 1–5.

[30] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 122–158, Feb. 2017.

[31] M. Zachariáová, M. Kekelyová-Beleová, and Z. Kotásek, "Regression test suites optimization for application-specific instruction-set processors and their use for dependability analysis," in *Proc. Eur. Conf. Digit. Syst. Design (DSD)*, 2016, pp. 380–387.

[32] A. Sabbaghi and M. R. Keyvanpour, "A novel approach for combinatorial test case generation using multi objective optimization," in *Proc. 7th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, 2017, pp. 411–418.

[33] A. Marchetto, G. Scanniello, and A. Susi, "Combining code and requirements coverage with execution cost for test suite reduction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 4, pp. 363–390, Apr. 2019.

[34] D. B. Mishra, R. Mishra, K. N. Das, and A. A. Acharya, "Test case generation and optimization for critical path testing using genetic algorithm," in *Soft Computing for Problem Solving*. Singapore: Springer, 2019, pp. 67–80.

[35] Y.-N. Zhang, H. Yang, Z.-K. Lin, Q. Dai, and Y.-F. Li, "A test suite reduction method based on novel quantum ant colony algorithm," in *Proc. 4th Int. Conf. Inf. Sci. Control Eng. (ICISCE)*, 2017, pp. 825–829.

[36] S. Kumar, P. Ranjan, and R. Rajesh, "Modified ACO to maintain diversity in regression test optimization," in *Proc. 3rd Int. Conf. Recent Adv. Inf. Technol. (RAIT)*, 2016, pp. 619–625.

[37] X.-C. Han, H.-W. Ke, Y.-J. Gong, Y. Lin, W.-L. Liu, and J. Zhang, "Multimodal optimization of traveling salesman problem: A niching ant colony system," in *Proc. Genetic Evol. Comput. Conf. Companion*, 2018, pp. 87–88.

[38] A. Ansari, A. Khan, A. Khan, and K. Mukadam, "Optimized regression test using test case prioritization," *Procedia Comput. Sci.*, vol. 79, pp. 152–160, Jan. 2016.

[39] M. N. Kabir, J. Ali, A. A. Alsewari, and K. Z. Zamli, "An adaptive flower pollination algorithm for software test suite minimization," in *Proc. 3rd Int. Conf. Elect. Inf. Commun. Technol. (EICT)*, 2017, pp. 1–5.

[40] A. S. Metwally, E. Hosam, M. M. Hassan, and S. M. Rashad, "WAP: A novel automatic test generation technique based on moth flame optimization," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2016, pp. 59–64.

[41] P. Gopi, M. Ramalingam, and C. Arumugam, "Search based test data generation: A multi objective approach using MOPSO evolutionary algorithm," in *Proc. 9th Annu. ACM India Conf.*, 2016, pp. 137–140.

[42] K. Z. Zamli, F. Din, S. Baharom, and B. S. Ahmed, "Fuzzy adaptive teaching learning-based optimization strategy for the problem of generating mixed strength t-way test suites," *Eng. Appl. Artif. Intell.*, vol. 59, pp. 35–50, Mar. 2017.

[43] S. R. Sugave, S. H. Patil, and B. E. Reddy, "DDF: Diversity dragonfly algorithm for cost-aware test suite minimization approach for software testing," in *Proc. Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, 2017, pp. 701–707.
[44] S. R. Sugave, S. H. Patil, and B. E. Reddy, "DIV-TBAT algorithm for test suite reduction in software testing," *IET Softw.*, vol. 12, no. 3, pp. 271–279, 2018.
[45] A. A. Alsewari, H. C. Har, A. A. B. Homaid, A. B. Nasser, K. Z. Zamli, and N. M. Tairan, "Test cases minimization strategy based on flower pollination algorithm," in *Proc. Int. Conf. Rel. Inf. Commun. Technol.* Cham, Switzerland: Springer, 2017, pp. 505–512.
[46] A. Choudhary, A. P. Agrawal, and A. Kaur, "An effective approach for regression test case selection using Pareto based multi-objective harmony search," in *Proc. 11th Int. Workshop Search-Based Softw. Test.*, 2018, pp. 13–20.
[47] w. Zheng, R. M. Hierons, M. Li, X. Liu, and V. Vinciotti, "Multi-objective optimisation for regression testing," *Inf. Sci.*, vol. 334, pp. 1–16, Mar. 2016.
[48] Z. Wei, W. Xiaoxue, Y. Xibing, C. Shichao, L. Wenxin, and L. Jun, "Test suite minimization with mutation testing-based many-objective evolutionary optimization," in *Proc. Int. Conf. Softw. Anal., Test. Evol. (SATE)*, 2017, pp. 30–36.
[49] A. 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 Comput. Appl.*, vol. 31, pp. 1–16, Feb. 2019.
[50] A. B. Nasser, A. Alsewari, and K. Z. Zamli, "Learning cuckoo search strategy for *t*-way test generation," in *Proc. Int. Conf. Comput., Anal. Netw.* Singapore: Springer, 2017, pp. 97–110.
[51] S. Singhal, B. Suri, and S. Misra, "An empirical study of regression test suite reduction using MHBG_TCS tool," in *Proc. Int. Conf. Comput. Netw. Inform. (ICCNI)*, 2017, pp. 1–5.
[52] Z. Anwar, "A hybrid-adaptive neuro-fuzzy inference system for multi-objective regression test suites optimization," *Neural Comput. Appl.*, vol. 29, pp. 1–15, Jun. 2018.
[53] R. Khan, M. Amjad, and A. K. Srivastava, "Optimization of automatic test case generation with cuckoo search and genetic algorithm approaches," in *Advances in Computer and Computational Sciences*. Singapore: Springer, 2018, pp. 413–423.
[54] P. Saraswat and A. Singhal, "A hybrid approach for test case prioritization and optimization using meta-heuristics techniques," in *Proc. 1st India Int. Conf. Inf. Process. (IICIP)*, 2016, pp. 1–6.
[55] D. Pradhan, S. Wang, S. Ali, T. Yue, and M. Liaaen, "CBGA-ES: A cluster-based genetic algorithm with elitist selection for supporting multi-objective test optimization," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, 2017, pp. 367–378.
[56] F. Liu, J. Zhang, and E.-Z. Zhu, "Test-suite reduction based on K-Medoids clustering algorithm," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery (CyberC)*, 2017, pp. 186–192.
[57] C. Coviello, S. Romano, and G. Scanniello, "Poster: CUTER: ClUstering-based TEst suite reduction," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May/Jun. 2018, pp. 306–307.
[58] C. Coviello, S. Romano, G. Scanniello, A. Marchetto, G. Antoniol, and A. Corazza, "Clustering support for inadequate test suite reduction," in *Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2018, pp. 95–105.
[59] A. Reichstaller, B. Eberhardinger, H. Ponsar, A. Knapp, and W. Reif, "Test suite reduction for self-organizing systems: A mutation-based approach," in *Proc. 13th Int. Workshop Automat. Softw. Test*, 2018, pp. 64–70.
[60] R. H. Rosero, O. S. Gómez, and G. Rodríguez, "Regression testing of database applications under an incremental software development setting," *IEEE Access*, vol. 5, pp. 18419–18428, 2017.
[61] W. Choi, K. Sen, G. Necula, and W. Wang, "DetReduce: Minimizing Android GUI test suites for regression testing," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 445–455.
[62] D. Marijan, M. Liaaen, A. Gotlieb, S. Sen, and C. Ieva, "TITAN: Test suite optimization for highly configurable software," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, Mar. 2017, pp. 524–531.
[63] P. Liu, J. Ai, and Z. Xu, "Probability model-based test suite reduction," *ACM SIGSOFT Softw. Eng. Notes*, vol. 42, no. 3, pp. 1–6, 2017.
[64] J.-W. Lin, R. Jabbarvand, J. Garcia, and S. Malek, "Nemo: Multi-criteria test-suite minimization with integer nonlinear programming," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng.*, May/Jun. 2018, pp. 1039–1049.
[65] F. Palomo-Lozano, A. Estero-Botaro, I. Medina-Bulo, and M. Núñez, "Test suite minimization for mutation testing of WS-BPEL compositions," in *Proc. Genetic Evol. Comput. Conf.*, 2018, pp. 1427–1434.
[66] C. R. Panigrahi and R. Mall, and S. Engineering, "Regression test size reduction using improved precision slices," *Innov. Syst. Softw. Eng.*, vol. 12, no. 2, pp. 153–159, 2016.
[67] A. Vahabzadeh, A. Stocco, and A. Mesbah, "Fine-grained test minimization," in *Proc. 40th Int. Conf. Softw.*, 2018, pp. 210–221.
[68] M. Carlsson, A. Gotlieb, and D. Marijan, "Software product line test suite reduction with constraint optimization," in *Proc. Int. Conf. Softw. Technol.* Cham, Switzerland: Springer, 2016, pp. 68–87.
[69] W. Fu, H. Yu, G. Fan, X. Ji, and X. Pei, "A test suite reduction approach to improving the effectiveness of fault localization," in *Proc. Int. Conf. Softw. Anal., Test. Evol. (SATE)*, 2017, pp. 10–19.
[70] U. Sivaji, A. Shraban, V. Varalaxmi, M. Ashok, and L. Laxmi, "Optimizing regression test suite reduction," in *Proc. 1st Int. Conf. Artif. Intell. Cogn. Comput.* Singapore: Springer, 2019, pp. 187–192.
[71] N. Jatana, B. Suri, and P. Kumar, "Mutation testing-based test suite reduction inspired from Warshall's algorithm," in *Software Engineering*. Singapore: Springer, 2019, pp. 357–364.
[72] L. Singh and S. N. Singh, "A path coverage-based reduction of test cases and execution time using parallel execution," in *Software Engineering*. Singapore: Springer, 2019, pp. 623–630.
[73] Z. Anwar and A. Ahsan, "Multi-objective regression test suite optimization with fuzzy logic," in *Proc. INMIC*, Lahore, Pakistan, 2013, pp. 95–100.
[74] N. Mansour and K. El-Fakih, "Simulated annealing and genetic algorithms for optimal regression testing," *J. Softw. Main., Res. Prac.*, vol. 11, no. 1, pp. 19–34, 1999.
[75] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw., Test. Verification Rel.*, vol. 22, no. 2, pp. 67–120, 2012.
[76] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
[77] N. Xiong, D. Molina, M. L. Ortiz, and F. Herrera, "A walk into metaheuristics for engineering optimization: Principles, methods and recent trends," *Int. J. Comput. Intell. Syst.*, vol. 8, no. 4, pp. 606–636, 2015.
[78] A. Sharma, P. Rishon, and A. Aggarwal, "Software testing using genetic algorithms," *Int. J. Comput. Sci. Eng. Surv.*, vol. 7, no. 2, pp. 21–33, 2016.
[79] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proc. 26th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2017, pp. 12–22.
[80] M. Vanmali, M. Last, and A. Kandel, "Using a neural network in the software testing process," *Int. J. Intell. Syst.*, vol. 17, no. 1, pp. 45–62, 2002.

AYESHA KIRAN received the B.S. degree in software engineering from the Government College University, Faisalabad, Pakistan. She is currently pursuing the M.S. degree in software engineering with the Computer and Software Engineering Department, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. Her research interest includes test suite optimization.

WASI HAIDER BUTT is currently an Assistant Professor with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. His research interests include model driven software engineering, web development, and requirement engineering.

MUHAMMAD WASEEM ANWAR is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. He is a Senior Researcher and an Industry Practitioner in the field of model-based system engineering (MBSE) for embedded and control systems. His major research interest includes MBSE for complex and large systems.

FAROOQUE AZAM is currently an Adjunct Faculty with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. He is teaching various software engineering courses, since 2007. His areas of interests include model driven software engineering, business modeling for web applications, and business process reengineering.

BILAL MAQBOOL received the M.S. degree in software engineering from the Computer and Software Engineering Department, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Pakistan, in 2018. From 2017 to 2018, he was a Research Assistant with the NUST, where he is currently a Senior Researcher in the field of software engineering. His research interests include business process automation through model driven software engineering (MDSE) and natural language processing (NLP).