Received January 22, 2018, accepted February 20, 2018, date of publication February 27, 2018, date of current version March 16, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2809600

A Systematic Review on Test Suite Reduction: Approaches, Experiment's Quality Evaluation, and Guidelines

SAIF UR REHMAN KHAN[1], SAI PECK LEE[1], (Member, IEEE), NADEEM JAVAID[2], (Senior Member, IEEE), AND WADOOD ABDUL[3], (Member, IEEE)

[1]Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia
[2]COMSATS Institute of Information Technology, Islamabad 44000, Pakistan
[3]Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

Corresponding author: Saif Ur Rehman Khan (saif_rehman@siswa.um.edu.my)

This work was supported in part by the Ministry of Higher Education, Malaysia, under Grant RG114-121CT and in part by the Twasol Research Excellence Program (TRE Program), King Saud University, Saudi Arabia. The work of S. U. Rehman Khan was supported by the Bright Sparks Program at the University of Malaya under Grant BSP-151(3)11).

ABSTRACT Regression testing aims at testing a system under test (SUT) in the presence of changes. As a SUT changes, the number of test cases increases to handle the modifications, and ultimately, it becomes practically impossible to execute all of them within limited testing budget. Test suite reduction (TSR) approaches are widely used to improve the regression testing costs by selecting representative test suite without compromising effectiveness, such as fault-detection capability, within allowed time budget. The aim of this systematic review is to identify state-of-the-art TSR approaches categories, assess the quality of experiments reported on this subject, and provide a set of guidelines for conducting future experiments in this area of research. After applying a two-facet study selection procedure, we finalized 113 most relevant studies from an initial pool of 4230 papers published in the field of TSR between 1993 and 2016. The TSR approaches are broadly classified into four main categories based on the literature including greedy, clustering, search, and hybrid approaches. It is noted that majority of the experiments in TSR do not follow any specific guidelines for planning, conducting, and reporting the experiments, which may pose validity threats related to their results. Thus, we recommend conducting experiments that are better designed for the future. In this direction, an initial set of recommendations is provided that are useful for performing well-designed experiments in the field of TSR. Furthermore, we provide a number of future research directions based on current trends in this field of research.

INDEX TERMS Software testing, regression testing, test suite reduction, experiments, guidelines.

I. INTRODUCTION

Software testing is the most prominent quality assurance activity that ensures the delivery of a quality software system by exposing maximum number of defects [1]. Software system continuously evolves due to several changes such as system functionality, business demands, and technology, which may have adversely affected the existing components of a System under Test (SUT) [2], [3]. Consequently, it is necessary to *retest* the SUT by executing newly generated test cases along with previously developed test cases. As a result, the size of test suite significantly increases by accommodating all changes in a software system, which may not be executed within allocated testing budget.

Regression testing is intended for checking the altered behavior of a SUT and ensuring that changes have not adversely affected the SUT quality [4]. Similarly, regression testing is meant to reduce testing effort by avoiding retesting of the entire SUT. Motivated by this, researchers have proposed three main test optimization techniques, which aim to provide better cost-effective solution for regression testing. Yoo and Harman [5] broadly discussed these test optimization techniques: (i) *test case selection*: identifies a subset of existing test cases that can be effective to test modified portions of the SUT; (ii) *test case prioritization*: determines an ordering of test cases to enhance certain objectives such as fault detection rate [6]; and (iii) *test suite reduction*:

identifies and permanently eliminates redundant test cases based on a certain criteria, such as maximum coverage of features or source code of the SUT, to produce minimal representative suite. Both test selection and reduction techniques target to minimize the size of test suite. However, test case selection techniques focus only on selecting such test cases, which covers the modified portions of the SUT rather than eliminating them from original test suite [5].

To ensure the quality of SUT, three key activities related to each test case need to be performed [7]: (i) creating appropriate test data, (ii) determining the expected output, and (iii) evaluating the test execution results. Ultimately, it helps the tester in exposing maximum possible defects from SUT, which leads to ensure the quality of the SUT. However, above-mentioned test cases related activities significantly increase the testing effort and cost especially if a specialized test execution environment (also referred as test-bed) is required for rigorous and transparent testing of a software system. Test Suite Reduction (TSR) techniques significantly reduce the testing time since they effectively decrease various test case related costs, such as test execution, test data management, and test storage, by producing minimal-size representative suite. The representative suite is essential to reduce the testing cost in the case when a single test case requires high human effort and data preparation. Consequently, TSR approaches improve the cost-effectiveness of regression testing and have a profound impact on the required testing effort. Moreover, researchers have mentioned that removal of redundant test cases does not greatly affect the overall SUT quality [8], [9].

In the literature, two systematic reviews have been conducted in the area of regression testing: (i) test case selection [10] and (ii) test case prioritization [3]. However, literature has not seen a systematic review that provides an up-to-date view of state of the research in the field of TSR. Therefore, it is timely to conduct a systematic review on TSR due to growing body of knowledge focusing on optimal TSR problem. Furthermore, to the best of our knowledge, no published work has focused on finding various classes of TSR approaches, and assessing the quality of experiments conducted in the area of TSR. Thus, we have conducted a systematic review to select relevant TSR approaches for classification based on the types of employed algorithms and underlying common viewpoints for TSR. Since majority of the work on TSR have been evaluated empirically, we have assessed the experiment quality based on the defined quality assessment criteria. Moreover, we have suggested a set of recommendations useful for systematically conducting and evaluating the future experiments on TSR.

The main objective of this organized review is to classify proposed TSR approaches and improve the body of credible evidence by assessing the quality of experiments conducted in the field of TSR. To achieve this objective, 4,230 potential papers were analyzed. Finally, 113 studies published in peer reviewed journals and conferences were selected for this methodical review. Four main classes of TSR approaches were identified including greedy-based, clustering-based, search-based, and hybrid approaches according to the types of algorithms employed. The results of this review indicate that for the most part, research on TSR has concentrated on greedy-based approaches (69%, 65/95) for solving the single-objective optimization problem, such as determining a reduced suite having maximum SUT coverage, using one or more forms of greedy algorithms. Notice that 95 represent the total number of selected TSR approaches, while 65 represent the number of greedy-based approaches. Correspondingly, it is noted that search-based TSR approaches are becoming of greater interest (20%, 19/95) in recent years owing to several successful results reported in the literature [11]–[14]. Search-based approaches principally focus on solving single-objective optimization problems by generally targeting a global optimal solution. Furthermore, it appears that the clustering-based approaches (3%, 3/95) targeted at solving single-objective optimization problems using different clustering and sampling algorithms. Likewise, it is observed that hybrid approaches (8%, 8/95) focus on providing better cost-effective solutions by integrating the strengths of other TSR approaches using one or more algorithm types. Nonetheless, it is also evident that most reported experiments in TSR are deficient in following any specific guidelines for planning, conducting, and reporting experiments; thus posing validity threats associated with their outcome. Finally, a set of recommendations are suggested, which would be useful for conducting well-designed experiments pertaining to TSR.

This paper is organized as follows. Section II describes background and related work. Section III provides the research method applied to perform this systematic review, while TSR approaches including similarities and differences among various classifications are presented in Section IV. Section V reports on the evaluation of experimental quality related to TSR as well as guidelines for conducting a well-designed experiment. Section VI discusses the overall results of this methodical review and Section VII outlines the validity threats. Section VIII provides conclusion and suggest potential research directions in the field of TSR.

II. BACKGROUND AND RELATED WORK

This section provides an overview of regression testing and optimal TSR problem. It also discusses the related work conducted in the field of regression testing with a special emphasis on TSR techniques.

A. REGRESSION TESTING AND OPTIMAL TSR PROBLEM

Regression testing is extensively conducted by the tester to detect maximum possible defects as resulted by the modifications in a SUT. The main objective of regression testing is to ensure that modifications have not adversely affected the behavior of existing component(s) of SUT [15].

In the literature, significant numbers of approaches have been proposed to reduce high cost of regression testing. The approaches are broadly categorized into three main techniques including test suite reduction, test case selection, and test suite prioritization. Since our focus is on TSR, therefore, an interested reader may consult the work provided in [5] for more information about test case selection and prioritization techniques. TSR approaches focus on finding the smallest representative suite by retaining fault-detection capability of the original test suite [16], [17]. The TSR problem is defined by Harrold *et al.* [18] as follows:

Given: *A universal test suite* $T = \{tc_1, tc_2 \ldots, tc_m\}$.
A set of test requirements $R = \{R_1, R_2, \ldots, R_n\}$ *that must be covered to provide the desired test coverage of the program under test.*
Subsets of $T = \{T_1, T_2 \ldots, T_n\}$, *where each test set* (T_i) *is associated with each test requirement* (R_i), *such that any one of the test case(s)* tc_j *of* T_i *can satisfy* R_i.

Objective: *Find the reduced suite (representative suite) containing minimal number of test cases from* T *that satisfies each* R_i *at least once.*

To determine an optimal TSR, researchers recommended computing the smallest possible *Reduced Suite* (RS) along with preserving their fault-detection capability similar to the original test suite [19]. Generally speaking, optimal TSR can only be achieved if the RS contains minimal number of test cases. The optimal TSR problem is known to be NP-hard problem [20] and is equivalent to minimum set cover [21]. Notice that there is an exponential time relationship between computing a minimum size RS and TSR problem size [20]. Current TSR strategies grounded on various types of heuristics to produce approximate solutions within practical computational time [18], [22]–[24].

B. RELATED WORK

There are many systematic reviews, mapping studies, and surveys have been performed in the area of regression testing. Engström *et al.* [10] conducted a systematic literature review on Regression Test Selection (RTS) techniques. They evaluated 28 RTS techniques and reported that no technique is superior as it is based on different varying factors. In contrast, Catal and Mishra [3] performed a systematic mapping study on Test Case Prioritization (TCP) techniques. The authors systematically found 120 papers on TCP approaches. They reported that coverage-based TCP approach dominating the field of TCP, and significant proportion of selected papers (64%, 77/120) used industrial projects based dataset. In contrast, Qiu *et al.* [15] conducted a thorogh systematic mapping study on regression testing. However, the authors only focused on web services rather than traditional software systems. In total, they selected 30 papers focusing on web service regression testing. They reported that TSR techniques have attained less attention of researchers compared to RTS and TCP techniques in the area of web service regression testing as only two studies have focused on web service based TSR.

Prior work [5], [25] has considered different viewpoints related to TSR compared to our systematic review. Elberzhager *et al.* [25] performed a mapping study to determine different areas that are useful in reducing the testing effort. They selected 144 papers and classified into five diverse areas including *defect estimation approaches*, *test input reduction approaches*, *test automation*, *test strategy*, and *quality assurance techniques before testing*, which are based on different mechanisms to reduce testing effort. However, majority of test input reduction approaches mentioned were RTS and TCP approaches. In contrast, Yoo and Harman [5] conducted an extensive survey that formally described and broadly reported regression testing techniques including RTS, TCP, and TSR. In addition, the authors also analyzed trends and issues related to regression testing techniques. However, the survey lacks in categorizing underlying TSR strategies and methods, which could be beneficial in improving the body of knowledge in the field of TSR. Furthermore, the survey did not critically evaluate the quality of experiments for TSR. Our work differs from Yoo and Harman's [5] study in three ways: (i) Yoo and Harman's [5] conducted a traditional survey, while our paper is a systematic review, (ii) Yoo and Harman's [5] considered papers published until 2009, whereas we have investigated papers published between 1993 to 2016, and (iii) Yoo and Harman's [5] considered 35 papers related to TSR, whereas we have systematically selected a comprehensive set of 113 papers. Our systematic review thoroughly covers shared viewpoints of proposed TSR strategies, and also critically evaluates the quality of conducted experiments. We also highlight potential future research directions for further study.

III. RESEARCH METHOD

This section explains the research method applied in the current systematic review. The research questions are presented in Section A, the study selection process is discussed in Section B, and data extraction and synthesis process are reported in Section C.

A. RESEARCH QUESTIONS

The main purpose of this systematic review is to summarize and investigate state-of-the-art work in TSR from various aspects, including proposed approaches and reported experiments. With this goal in mind, we have formulated two main research questions.

RQ1: What type of TSR approaches have been proposed in existing literature and how they can be classified?

This research question intends to categorize investigations undertaken so far regarding TSR approaches. The following sub-research questions should distinguish the most important characteristics of TSR approaches:

1) *RQ1.1: What are the various classifications of TSR approaches?*
2) *RQ1.2: What are the commonalities and differences among approaches for each classification?*

RQ2: What is the overall quality of reported TSR experiments in the domain of regression testing?

This research question is meant to measure the quality of experiments carried out on TSR by synthesizing the results obtained in the reported experiments according to the defined quality assessment criteria.

The present systematic review is conducted by following the guidelines suggested in [26]. After exhaustively applying the study selection strategy, 113 studies were finalized to answer the formulated research questions.

B. STUDY SELECTION STRATEGY

To warrant the completeness of relevant study selection, three main steps were taken: (i) selecting the sources of information, (ii) identifying the search keywords, and (iii) including and excluding studies according to the defined inclusion and exclusion criteria.

1) SOURCES OF INFORMATION

To guarantee this systematic review is comprehensive, the following electronic source repositories were used to collect the potential studies:

- IEEE Xplore (<ieeexplore.ieee.org>)
- Web of Science (<www.isiknowledge.com>)
- Springer link (<www.link.springer.com>)
- Compendex (<www.engineeringvillage2.org>)
- ACM Digital Library (<www.dl.acm.org>)
- ScienceDirect (<www.sciencedirect.com>)
- Inspec (<www.theiet.org/publishing/inspec>)

The used source repositories contain nearly all important workshops, symposiums, journals, and conference proceedings within the software engineering domain, as confirmed by [27]. We performed manual filtering to exclude duplicates among potential studies.

2) SEARCH KEYWORDS

The study selection process was initiated by executing the devised search string on the identified source repositories. To systematically define the search string, three steps were taken: (i) identifying the main search keywords according to the formulated research questions, (ii) identifying the synonyms and alternative words to the main search keywords, and (iii) creating the search string by integrating the identified keywords, synonyms, and alternative words using Boolean AND, OR operators.

Based on the main work objectives of classifying the TSR approaches and evaluating the quality of experiments on TSR, the following major search keywords were used: *regression testing*, *test suite reduction*, and *experiment*.

Upon having attempted some strings in search query formulation, it appears that *"regression testing"* would return numerous studies with no relevance. For this reason, *"regression"* was linked with *"test"* via a Boolean AND operator to obtain a relevant set of studies. To ensure that no relevant studies were omitted, the word *"reduction"* was applied along with substitutes like *"filtering"* and *"minimisation."* Subsequently, in searching the domain, the word "*approach*" was connected with alternatives like *"algorithm," "heuristic,"* and *"technique"* via the Boolean OR operator. Similarly, the term "*experiment*" was utilized with its alternatives "*empirical study*" and "*experimental study,*" and these were joined with the main expression "*regression test suite reduction*" using the Boolean OR operator. Finally, the devised string was searched in all titles, abstracts, and keywords in each selected source repository. The formulated search string is given as:

{((‘regression’ AND ‘test’) AND (‘*case’* OR ‘*suite’*) AND (*reduction* OR *minimisation*OR *filtering*) OR (*approach*OR *heuristic* OR*algorithm* OR *technique*) OR (*experiment* OR *empirical study* OR *experimental study*)) < in title, abstract, and keywords >}

To ascertain that the most relevant studies were selected for review, the start year was set to 1990 and the final year to 2016. However, the earliest study chosen for this review was published in 1993 [18]. Furthermore, only papers written in English were selected.

3) STUDY SELECTION ACCORDING TO INCLUSION AND EXCLUSION CRITERIA

Since this systematic review is geared toward TSR, the inclusion and exclusion criteria for choosing the appropriate studies are based on various TSR related aspects, i.e., proposed TSR approaches and reported experiments aimed at determining the efficacy of proposed TSR approaches. In this section, the devised inclusion and exclusion criteria are discussed.

First, the defined search string was executed on all selected source repositories and 4,230 potential studies were found. After manually removing the duplicates from different repositories, 2510 studies were left. To select the most relevant studies, a two stage study selection procedure was applied carried out by three participating researchers. Initially, a random set of studies were given to the participating researchers to establish a common understanding. In the first selection stage, the studies found were randomly separated into three equal sets and handed to the three participating researchers who checked titles and abstracts. If a researcher was not confident about a given study, it was retained for the second study selection phase. In the first selection stage, studies were excluded based on titles and abstracts by applying the given exclusion criteria:

- Titles or abstracts that do not include *‘test suite reduction’* or other related keywords such as *‘test suite minimisation’* or *‘test suite filtering’*.
- Titles or abstracts that do not include targeted aspects related to test suite reduction (i.e., proposed TSR approaches and reported experiments on TSR).

Subsequent to applying the inclusion and exclusion criteria in the first stage, 156 studies were located. In the second phase, the selected studies were once again split into three sets and randomly assigned to the three researchers who checked the contents of the allocated studies to ensure that studies reporting the targeted aspects related to TSR. If a researcher was unsure whether to include or exclude a study, consultation with other researcher(s) followed. In the second

TABLE 1. Type of data collected for the formulated research questions.

Research Questions		Type of Data Collected
RQ1	RQ1.1	test suite reduction approach, purpose, algorithm type
	RQ1.2	coverage source, coverage type, optimization problem type, technique type; clustering algorithm name and type, sampling algorithm name and type, subclass; search algorithm, optimization problem type, solution type; hybrid approach name and type, approach combination
RQ2		goal definition, context, null hypothesis, variables (dependent and independent), cost measures, effectiveness measures, scalability measures, experiment design, comparison baseline, selection of subjects, standard design, instrumentation, descriptive statistics, hypothesis testing, statistical significance test, practical significance test, scalability analysis, validity threats (conclusion, internal, external, construct)

selection stage, studies were excluded by checking the contents of each study according to the following exclusion criteria:

- Studies that do not report test suite reduction.
- Focus was only directed towards studies that were published in peer-reviewed conferences, workshops, symposium, or journals. The posters and extended abstracts were excluded due to lack of technical details.

4) DEMOGRAPHIC DATA AND OVERVIEW

Finally, we selected 113 studies, of which 95 reported TSR approaches and 18 referred to experiments conducted in the field of TSR. The earliest selected study was published in 1993, and the latest study was published in 2016 (see Appendix).

C. DATA EXTRACTION AND SYNTHESIS

To collect the data from the final selected studies, a Microsoft Excel sheet specifically designed for data extraction purposes was utilized. Two sets of information were gathered from each study: (i) standard information including study's title, authors' names, complete reference, a brief summary, characterization of TSR aspects (i.e., TSR approach and experiment), publication type (e.g., conference, workshop, symposium, or journal), researcher's name and comments; and (ii) summarized information according to the formulated research questions (see Table 1).

Next, the final chosen studies were equally distributed among three participating researchers. Each of the researchers read through the assigned set, extracted the relevant data, and indicated this information on the designed Microsoft Excel sheet. In case of any ambiguity, the participating researchers made decisions through mutual consensus by calling a wrap-up meeting.

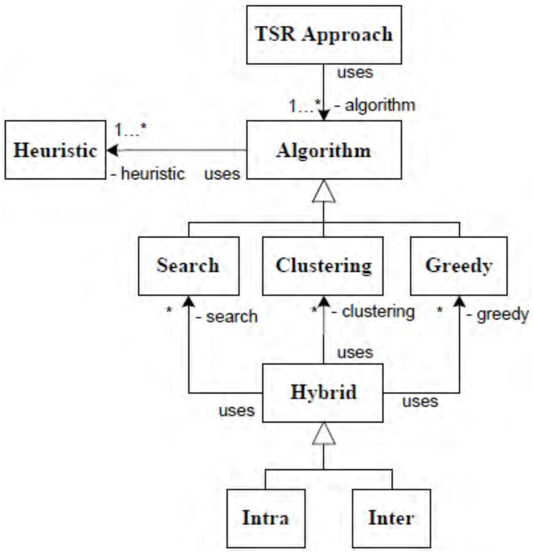


FIGURE 1. Conceptual diagram of Test Suite Reduction (TSR) approaches. A TSR approach determines representative suite by employing one or more algorithms from one or more *Algorithm* class using one or more heuristics from *Heuristic*.

IV. TYPES OF TEST SUITE REDUCTION (TSR) APPROACHES (RQ1)

This section presents current state-of-the-art TSR approaches along with similarities and differences among various classifications, which is based on different underlying concepts and algorithms.

A. CLASSIFICATION OF TSR APPROACHES (RQ1.1)

Based on this systematic review, we classified TSR approaches into four main categories based on the type of employed algorithm(s) (see Fig. 1): *Greedy*, *Clustering*, *Search*, and *Hybrid*.

The following definitions served to define the classification of TSR approaches:

$TS = \{tc_1, tc_2, \ldots, tc_{nts}\}$ is an original test suite that is to be reduced and *'nts'* is the number of test cases in *TS*.

$RS = \{tc_1, tc_2, \ldots, tc_{nrs}\}$ is a subset of *TS* and *'nrs'* is the number of representative test cases in *RS*, where $nrs < nts$.

$Heuristic = \{h_1, h_2, \ldots, h_{nh}\}$ is a set of heuristics used to guide test suite reduction and *'nh'* is the number of heuristics in *Heuristic*.

$Cost = \{cost_1, cost_2, \ldots, cost_{ncost}\}$ is a set of cost measures and *'ncost'* is the number of cost measures in *Cost*, such as cost of a TSR algorithm and execution time of the RS.

$Effect = \{effect_1, effect_2, \ldots, effect_{neffect}\}$ is a set of effectiveness measures and *'neffect'* is the number of effectiveness measures in *Effect*, e.g., test time reduction percentage, test suite reduction percentage, and test coverage percentage.

$AlgorithmClass = \{class_1, class_2, \ldots, class_{nac}\}$ is a set of classifying algorithms employed for TSR and *'nac'* is the number of classifying algorithms in *AlgorithmClass*.

$CostF\ (TestSuite, cost_i)$ is a function that returns the cost of a test suite (TestSuite) based on a cost measure ($cost_i$) from *Cost*.

$EffectF(TestSuite, effect_i)$ is a function that returns the effectiveness of a test suite (*TestSuite*) based on an effectiveness measure ($effect_i$) from *Effect*.

1) TSR APPROACHES

Generally, a TSR approach obtains Reduced Suite (RS) by employing one or more algorithms from one or more *AlgorithmClass* using one or more heuristics from *Heuristic* (Fig. 1) with the aim of achieving (1) and (IV-A.1).

$$\sum_{i=1}^{n\cos t} CostF(RS, \cos t_i) \leq \sum_{i=1}^{n\cos t} CostF(TS, \cos t_i), \tag{1}$$

$$where \quad (n\cos t \geq 1) \wedge (RS \wedge TS \neq 0)$$

$$\sum_{i=1}^{neffect} EffectF(RS, effect_i) \leq \sum_{i=1}^{neffect} EffectF(TS, effect_i), \tag{2}$$

$$where \quad (neffect \geq 1) \wedge (RS \wedge TS \neq 0)$$

The following sub-sections present the classification of TSR approaches.

a: GREEDY-BASED APPROACH

$Coverage = \{coverage_1, coverage_2, \ldots, coverage_{ncoverage}\}$ is a set of coverage criteria, where *'ncoverage'* is the number of coverage criteria. Notice that *Coverage* is a subset of the *Effect* set.

Coverage(TS, Criterion) is a function that returns the coverage percentage based on a criterion (*Criterion*) from the *Coverage* set as achieved by *TS*.

A test case can satisfy one or more elements of *Criterion* (e.g., in the case of control flow-based coverage, the criterion element can be a statement, branch, or path), a binary relation matrix *Satisfy(TestSuite, Criterion)* for the set of *Criterion* c, and *TestSuite* TS can be defined as:

$$Satisfy(TS, Criterion) = \{(tc, c) | tc\ satisfyc, tc \in TS \wedge c \in Criterion\} \begin{bmatrix} 1, & if\ tc\ satisfy\ c \\ 0, & otherwise \end{bmatrix} \tag{3}$$

where rows represent the test cases (*tc*) and columns signify the *Criterion* (*c*) of a binary relation matrix.

A greedy-based approach employs an algorithm from $class_i$ from *AlgorithmClass* that uses one or more heuristics from *Heuristic*, which are greedy-based heuristics, to obtain RS such that conditions (4) and (5) hold [18]:

$$nrs < nts \tag{4}$$

$$\sum_{i=1}^{ncoverage} Coverage(RS, coverage_i) \leq \sum_{i=1}^{ncoverage} Coverage(TS, coverage_i), \quad where\ n\ coverage \geq 1 \tag{5}$$

Greedy-based approaches usually employ a greedy algorithm (or variants of greedy algorithm) to find a solution. A greedy algorithm runs the following steps to obtain RS [28]:

Step 1:

$$\forall tc_i \cdot tc_i \in TS \wedge i \in \mathrm{N}^+, \quad i \leq nts : Selection(tc_i) = Satisfy(TS, Criterion) \tag{6}$$

where the function *Selection(tc$_i$)* returns the best candidate test case (tc$_i$) based on the maximum covered entries of a binary matrix *Satisfy(TS, Criterion)*. In the case of a tie between test cases (i.e., covering same number of entries of a binary matrix), apply random test selection strategy.

Step 2: Mark the corresponding covered entries of the test case (tc_i) in *Satisfy (TS, Criterion)* as 0 (i.e., satisfied), and add the test case (tc_i) in the *RS* and removes tc_i from *TS* as denoted in (7).

$$RS = (RS \cup tc_i) \text{ and } TS = (tc_i \backslash TS) \tag{7}$$

Step 3: Check whether all the entries in *Satisfy(TS, Criterion)* are marked as 0, $TS = \phi$, and $nrs < nts$. If no, move to Step 1, otherwise return the *RS*.

Greedy algorithm provides a local optimal solution by following the *Current Best* strategy to determine the RS without considering the global optimal solution. In all iterations, a greedy algorithm selects a test case that has high SUT coverage until the desired coverage is attained [18], [29]. However, it has been observed that the greedy algorithm determines a minimal-cardinality suite at the cost of some or marginal compromise in fault-detection capability [30].

b: CLUSTERING-BASED APPROACH

$SubsetTS = \{subset_1, subset_2, \ldots, subset_{nsubset}\}$ is a set of *TS* subsets, where *'nsubset'* is the number of *TS* subsets.

$SimilarityMeasure = \{sm_1, sm_2, \ldots, sm_{nsm}\}$ is a set of similarity measures, where *'nsm'* is the number of similarity measures.

Similarity (SimilarityMeasure, tc_1, tc_2) is a function that takes a *SimilarityMeasure* and two test cases (tc_1 and tc_2) as input and returns a value representing the similarity of the given test cases. It should be noted that the similarity between test cases is calculated based on cost and/or effectiveness measures.

$DissimilarityMeasure = \{dsm_1, dsm_2, \ldots, dsm_{ndsm}\}$ is a set of dissimilarity measures, where *'ndsm'* is the number of dissimilarity measures.

Dis similarity (DissimilarityMeasure, $subset_1, subset_2$) is a function that takes a *DissimilarityMeasure* and two subsets ($subset_1$ and $subset_2$) as input and returns a value representing the dissimilarity between the given subsets of the test cases. The dissimilarity between test case subsets is calculated based on cost and/or effectiveness measures.

A clustering-based approach functions in the following way to obtain the RS [31]:

Step 1: Employ an algorithm from $class_i$ from *AlgorithmClass*, where $class_i$ is a class of cluster-based algorithms to obtain *SubsetTS* such that for each $subset_k$ in *SubsetTS* (8):

$$\forall i, j \wedge i \neq j : Similarity(sm_1, tc_i, tc_j) \geq SimThreshold \quad (8)$$

where sm_1 is the 1st similarity measure and *i, j ranges from 1* to k, while *SimThreshold* is a value below which two test cases are considered significantly different.

Moreover, for each $subset_k$ in *SubsetTS* as defined in (9):

$$\forall i, j \wedge i \neq j : Dissimilarity(dsm_1, subset_i, subset_j) \quad (9)$$

where dsm_l is the 1st dissimilarity measure and *i, j ranges from 1* to *nsubset*, while *DisThreshold* is a value below which two test cases are considered significantly similar.

Step 2: Obtain *RS* by sampling x test cases from each $subset_k$ by employing sampling algorithms and mark subset_k as satisfied such that $nrs < nts$.

Step 3: Check whether *SubsetTS* is marked as satisfied. If no, move to Step 2, otherwise return *RS*.

Suppose there is a subset $TS = \{S_1, S_2, \ldots, S_k\}$, where each S_i is a subset of *TS* resulting from cluster analysis, each test case in S_i is similar to each other, and similarity is measured based on similarity metrics such as Jaccard index [32] or Levenshtein [33]. Each S_i differs from other S_j, where $i \neq j$. The difference is measured based on dissimilarity metrics such as Euclidean distance [31]. Finally, the RS is obtained by sampling test cases from each S_i, for instance one-per-cluster sampling [34].

c: SEARCH-BASED APPROACH

For *TS* there are set of potential solutions $S = \{S_1, S_2, S_3, \ldots, S_k\}$, where k is the total number of potential solutions and can be measured as $2^K - 1$. The size of each S_i (i.e., the number of test cases) ranges from *1* to n.

Fitness(solution) is a function that returns the fitness of a solution, where fitness is calculated based on cost and/or effectiveness measures.

A search-based approach finds the *RS* from *S* (where $S \in TS$) using a search algorithm such that following conditions hold as mentioned in (10) to (13):

$$\forall_i : Fitness(RS) > Fitness(s_i), i \text{ ranges from 1 to } k \quad (10)$$

$$CostF(RS, cost_j) \leq CostF(TS, cost_j) \quad (11)$$

$$EffectF(RS, effect_j) \leq EffectF(TS, effect_j) \quad (12)$$

$$nrs < nts \quad (13)$$

To obtain *RS*, a search-based approach operates as follows [35]:

Step 1: Initialize *RS* as empty (i.e., $RS = \phi$)

Step 2: $\forall s_i$, *TS*, apply the fitness function based on cost and/or effectiveness measures as mentioned in (14) and (15):

$$\begin{aligned} CostF(RS, \cos t_i) &= \sum\nolimits_{i=1}^{k} \sum\nolimits_{i=1}^{n} CostF(s_i, \cos t_i) \\ &\leq \sum\nolimits_{i=1}^{n \cos t} CostF(TS, \cos t_i) \quad (14) \end{aligned}$$

$$\begin{aligned} EffectF(RS, effect_i) &= \sum\nolimits_{i=1}^{k} \sum\nolimits_{i=1}^{n} EffectF(s_i, effect_i) \\ &\leq \sum\nolimits_{i=1}^{neffectt} EffectF(TS, effect_i) \quad (15) \end{aligned}$$

Step 3: Compare the attained cost ($cost_i$) of *RS* and *TS* using the cost measure (i.e., *CostF*) mentioned in (16):

$$(CostF(RS, \cos t_i) \leq CostF(TS, \cos t_i)) \quad (16)$$

and/or the obtained effectiveness ($effect_i$) of *RS* and *TS* using the effectiveness measure (i.e., *EffectF*) defined in (17)

$$(EffectF(RS, effect_i) \leq EffectF(TS, effect_i)) \quad (17)$$

Step 4: Obtain *RS* by checking (18) and (19).

$$\sum_{i=1}^{k} (Fitness(RS) > Fitness(s_i)) \quad (18)$$

$$(nrs < nts) \quad (19)$$

In the search-based approach there is a set of potential solutions, where each solution must be capable of being represented such that it is usable in the search algorithm. Then, RS is determined by applying the devised fitness function based on the cost and/or effectiveness measures [36], [37]. Notice that similar to clustering-based approaches, search-based approaches also employ *SimilarityMeasure* to determine the similarity score between pairs of test cases. Consequently, search-based approaches generate a RS having diverse set of test cases based on the defined cost and/or effectiveness objective(s) and similarity score.

d: HYBRID APPROACH

As shown in Fig. 1, a hybrid approach employs one or more types of TSR algorithm. Hybrid approach is further classified into Intra and Inter hybrid techniques:

1) *Intra Hybrid Techniques*: employ more than one algorithm from $class_i$ from *AlgorithmClass* for test suite reduction.
2) *Inter Hybrid Techniques*: employ more than one *class* of *AlgorithmClass*.

e: CONCLUDING REMARKS

Concerning the discussion given about RQ1.1, it can be concluded that a TSR approach mainly differs from other approaches (i.e., greedy-based, clustering-based, search-based, and hybrid) in terms of the type of algorithms applied to obtain a reduced test suite with some or marginal compromise in testing effectiveness, more importantly fault-detection capability. In this review, four main algorithm classes have been identified: (i) greedy-based algorithms, (ii) deterministic algorithms (i.e., Integer Linear Programming), (iii) clustering-based algorithms (clustering and sampling), and (iv) search-based algorithms. It has also been observed that a number of approaches (i.e., hybrid) are utilized in an attempt to combine various algorithms for TSR.

2) COMMONALITIES AND DIFFERENCES AMONG TSR APPROACHES (RQ1.2)

The following sub-sections present the main perspectives that help to understand the commonalities and differences for each classification of TSR approaches.

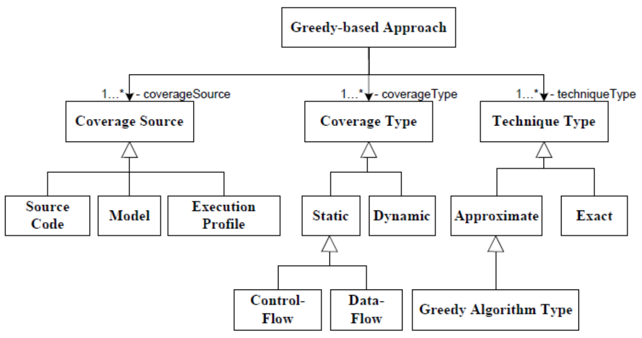


FIGURE 2. Conceptual diagram of greedy-based approach.

TABLE 2. Variants of greedy-based heuristics.

Coverage Source	Optimization Problem Type	Coverage Type	Relevant Studies
source code-based	single-objective	static	(S1; S4; S10; S19; S20; S22; S27; S30; S37; S46; S49; S54; S58; S60; S62; S68; S82; S87; S97; S99; S100; S101; S106)
	multi-objective	static	(S15; S31; S32; S35; S36; S78; S80; S85; S86; S98; S105)
model-based	single-objective	static	(S40; S48; S104)
execution profile-based	single-objective	dynamic	(S2; S3; S9; S12; S13; S17; S21; S24; S25; S29; S33; S34; S45; S50; S72; S77; S88; S92; S93; S107; S108; S109; S111)

a: GREEDY-BASED APPROACH

Greedy-based approach can be categorized from three perspectives: (i) Coverage Source, (ii) Coverage Type, and (iii) Technique Type employed for TSR. The classification is depicted in Fig. 2.

1) *Coverage Source*: Three sources of coverage identified are: (i) Source Code-based, (ii) Model-based (e.g., graph theory and model checker), and (iii) Execution Profile-based. Notice that execution profile-based coverage is obtained by executing the entire test suite. Table 2 provides a list of various heuristics that used these coverage sources.
2) *Coverage Type*: Two types of coverage recognized are: (i) static and (ii) dynamic. Notice that dynamic coverage is achieved from SUT execution. The source code and model-based TSR heuristics use static coverage criteria, while execution profile-based TSR heuristics use dynamic coverage criteria. However, two source code-based TSR approaches (S40; S48) have been found, which employ dynamic coverage criteria to obtain the RS. Table 3 lists a range of coverage criteria utilized in greedy-based approaches.
3) *Technique Type:* Two types of techniques identified are: (i) approximate and (ii) exact. Approximate techniques exploit different variants of greedy algorithms in order to perform approximation, which is closer to an optimal solution. In contrast, exact techniques (e.g., Integer Linear Programming) determine a global optimal solution at the cost of long execution time compared to approximate techniques [17]. Table 4 provides a list of these techniques employed by greedy-based approaches.

TABLE 3. Coverage type and criteria used by greedy-based heuristics.

Coverage Type	Subclass	Coverage Criteria	Relevant Studies
static	data-flow	computation-use	(S24)
		predicate-use	(S24)
		definition-use	(S1; S24; S31; S32)
		definition-use pairs	(S69; S102)
	control-flow	statements	(S10; S49; S82; S85; S87; S98)
		block (consecutive set of statements)	(S2; S17; S80)
		state (variable domain)	(S30)
		loop	(S85)
		branch	(S10; S17; S24; S32; S45; S78; S85; S102)
		transition	(S29; S30; S50; S72)
		transition-pairs	(S93)
		path	(S21)
		unit	(S22; S37; S46; S99)
		mutation score	(S5; S80; S97)
		event	(S80)
		semantic logic (MC/DC)	(S27; S30; S35; S85)
		syntactic logic (MUMCUT)	(S62)
Dynamic		dynamic program invariants	(S23)
		test-for-diagnosis (TfD)	(S48)
		grammar-rule	(S40)
		call stack	(S38)
		graphical user interface based call stack	(S56)
		object-oriented programs based call stack	(S51)
		computation-use	(S24)
		predicate-use	(S24)

TABLE 4. Types of technique used in greedy-based approaches.

Technique Type	Algorithm/ Method used	Optimization problem type	Relevant Studies
Approximate	greedy	single-objective	(S1; S2; S3; S4; S9; S10; S12; S13; S17; S19; S20; S21; S22; S23; S24; S25; S27; S29; S30; S33; S34; S37; S38; S45; S46; S49; S50; S51; S54; S56; S58; S60; S62; S68; S72; S77; S83; S87; S88; S92; S93; S97; S99; S100; S101; S106; S107; S108; S109)
		multi-objective	(S15; S32; S35; S36; S40; S48; S78; S80; S85; S86; S98; S104; S105)
Exact	integer linear programming	multi-objective	(S31)
	bi-criteria		(S82)

Source code greedy-based heuristics (39%, 37/95) have remained an intense research topic over the past few decades (Table 2). Researchers have proposed a number of greedy-based TSR approaches that primarily consider source code-level structural coverage. Most of the work on code greedy-based TSR heuristics (24%, 23/95) is aimed at solving single-objective optimization problems using different variants of greedy algorithm, such as additional greedy algorithm (S10) and delayed greedy algorithm (S32). Single-objective optimization 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. For instance, a test engineer might employ a multi-objective optimization based TSR approach to find the minimum RS size that achieves maximal SUT coverage with the least test execution cost (S70).

Some model-based heuristics (S2; S17; S45) incorporate the *subsumption* concept for TSR, meaning that if any basic concept (e.g., statement) in a super concept (e.g., path) is covered by a test suite, then all basic concepts in that super concept must also be covered by the same test suite. Ultimately, *subsumption* concept significantly reduces the size of test suite without compromising on their fault detection ability. In contrast, some work has focused on proposing execution profile-based approaches (S23; S38; S51; S56; S83), which are based on dynamic coverage criteria. However, all proposed execution profile-based approaches focused on single-objective optimization problem.

Greedy-based heuristics (S15; S19; S54; S78; S97) use a *binary matrix* as an input to perform TSR. For example, a source code-coverage heuristic for single-objective optimization uses a component-path matrix (S19) and test-requirements matrix (S54). Alternatively, in the source code-coverage heuristic for multi-objective optimization a purpose-test incidence matrix is utilized (S15). Furthermore, the greedy algorithm-based approach for single-objective optimization calls for a test-mutants matrix (S97) and the greedy algorithm-based approach for multi-objective optimization uses a test-requirements matrix (S78).

Greedy-based approach greatly depends on the coverage criteria (Table 3), which can be generally classified into: (i) static and (ii) dynamic criteria. Both coverage criteria act as test redundancy metrics to assist the greedy-based approach in determining redundancy among test cases. The static coverage criteria can be further categorized into two sub-criteria: (i) data-flow based and (ii) control-flow based. Researchers are presently focusing on different dynamic coverage criteria due to diversity in applications. Although dynamic criteria require automatic compilation; however, they never guarantee about complete coverage of the SUT.

A greedy-based approach employs two types of techniques including, approximate and exact, to find a minimal-cardinality suite (Table 4). Approximate techniques employ various types of greedy algorithms (22%, 21/95) to determine minimal size reduced suite. However, approximate techniques never guarantee about computing global optimal solution. In contrast, exact (deterministic) techniques ensure global optimal solution for TSR problem [17]. To the best of our knowledge, only two exact techniques (S31; S82) have been proposed in greedy-based class, which are based on Integer Linear Programming (ILP) and a bi-criteria model to ensure minimal-cardinality suite at the cost of long execution time. Consequently, a tradeoff between test suite size and algorithm runtime is required to provide more cost-effective solution for optimal TSR problem [17].

According to the above discussion, it can be deduced that most greedy-based approaches attempt to achieve one or more coverage criteria to determine a minimal set of test cases by employing approximate techniques using variants of greedy algorithms or exact techniques. However, the proposed approaches differ in relation to the coverage source, coverage type, and technique type applied to obtain a reduced test suite.

b: CLUSTERING-BASED APPROACH

Clustering-based approach is hereby grouped in accordance with two criteria: (i) clustering algorithm type and (ii) sampling algorithm type (Fig. 3).

Table 5 lists which techniques employ what types of clustering algorithm, whereas Table 6 names various kinds of sampling algorithm used by state-of-the-art clustering-based approach.

It is evident that existing clustering-based TSR approaches employ supervised clustering algorithms to group test cases

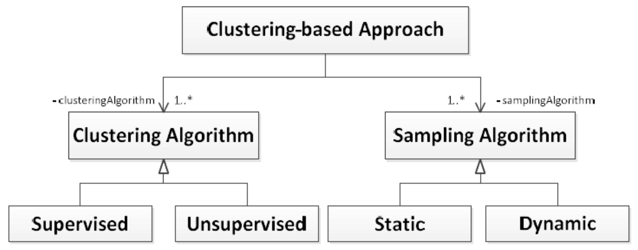


FIGURE 3. Conceptual diagram of clustering-based approach.

TABLE 5. Types of clustering algorithms used by clustering-based approaches.

Clustering Algorithm	Subclass	Technique Name	Related Studies
supervised	hierarchical clustering	user sessions based on concept lattice's hierarchical clustering	(S41)
		user sessions clustering based on agglutinate hierarchical clustering algorithm	(S95)
	machine learning	MachinE Learning based refinement of BlAck-box test specification (MELBA)	(S64)

TABLE 6. Types of sampling algorithms used by clustering-based approaches.

Sampling Algorithm	Technique Name	Relevant Studies
dynamic	adaptive sampling	(S64)
	longest common subsequence output	(S95)

based on the determined similarity/dissimilarity [38], where the majority of approaches (S41; S95) use hierarchical clustering algorithms (Table 5). Afterward, clustering-based approaches employ different dynamic sampling algorithms (S64; S95) to select the representative test cases (Table 6).

In conclusion, all clustering-based approaches work in a similar way (as mentioned in Section IV(A)); however, the approaches mainly differ from each other in terms of algorithm used for clustering and sampling.

c: SEARCH-BASED APPROACH

Search-based approach can be classified based on single-objective or multi-objective optimization algorithms, as shown in Fig. 4.

Table 7 tabulates different variants of search-based algorithms.

It is observed that majority of search algorithms (79%, 15/19) concentrate on solving single-objective optimization problems (Table 7). In contrast, other search algorithms (S70; S90; S103; S110) are meant to solve

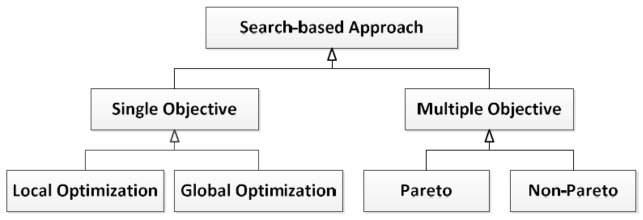


FIGURE 4. Conceptual diagram of search-based approach.

TABLE 7. Variants of search-based algorithms.

Optimization Problem Type	Type of Solution	Relevant Studies
single-objective	local optimization	(S67)
	global optimization	(S7; S8; S28; S42; S44; S69; S71; S73; S74; S75; S76; S89; S91; S112)
multi-objective	Pareto-based	(S90; S103; S110)
	Non-Pareto based	(S70)

multi-objective TSR problems by establishing an acceptable tradeoff between the mentioned cost and effectiveness measures. Moreover, single-objective search algorithms provide either local or global types of solutions. Local optimization presents an optimal solution with respect to the neighboring set of solutions, while global optimization determines an optimal solution by considering all possible solutions. Global optimization is more effective but requires greater computational effort than local optimization [35]. Therefore, the type of solution (i.e., local vs. global) depends on the available computational resources. In contrast, multi-objective search algorithms can be grouped into two main classes: (i) Pareto optimization and (ii) Non-Pareto optimization. Pareto optimization gives a set of conflicting optimal solutions by replacing an individual dominating candidate solution with an individual population [39]. In contrast, Non-Pareto optimization provides a set of non-conflicting optimal solutions from all possible solutions [13]. Interested readers may refer to [40] for more details on multi-objective optimization-based classes.

In conclusion, all search-based approaches for TSR function similarly, as conferred in Section IV(A), with the principal differences lying in the search objectives (i.e., single objective vs. multi objective) as well as type of solution (i.e., local vs. global).

d: HYBRID-BASED APPROACH

Table 8 lists various hybrid techniques reported in the literature. It is clear that researchers have mostly used a greedy-based approach (5%, 5/95) to propose hybrid techniques, whereby greedy algorithm-based heuristics have been widely employed to determine the RS. Moreover, researchers have concentrated on all possible combinations of main TSR approach categories (i.e., search, greedy, and clustering) to propose hybrid techniques.

TABLE 8. Variants of hybrid-based techniques.

Type	Approach Combination	Subclass	Relevant Studies
intra-hybrid	greedy vs. greedy	static slicing and delta debugging	(S53)
	greedy vs. greedy	greedy heuristic (HGS) and greedy heuristic G (GRE)	(S61)
	greedy vs. greedy	greedy heuristic G (GRE) and selective redundancy	(S81)
inter-hybrid	search vs. search	simulated annealing and genetic algorithm	(S18)
	greedy vs. clustering	greedy heuristic and clustering algorithm	(S65; S102)
	greedy vs. search	greedy heuristic and genetic algorithm	(S84)
	search vs. clustering	genetic algorithm and clustering algorithm	(S96)

e: DISCUSSION

In line with the discussion on various classifications in prior sub-sections, it can be deduced that four main TSR approaches (i.e., greedy, search, clustering, and hybrid) have been proposed during the past twenty-four years.

Focus has mainly been directed toward proposing greedy-based TSR approaches (69%, 65/95). This is mainly due to the common belief that achieving 100% SUT coverage via different coverage requirement(s) can expose a maximum number of defects [7]. Conversely, search-based TSR algorithms are now gaining more attention (20%, 19/95) following the introduction of the *Search-based Software Engineering (SBSE)* notion by Harman and Jones [36]. A possible reason is that search algorithms have shown significant benefits [34], [41], [42] and researchers are interested in studying the applicability of search algorithms in the context of TSR [35]. Moreover, the remaining TSR approach categories including hybrid (8%, 8/95) and clustering (3%, 3/95) have received less consideration with regards to performing TSR.

V. QUALITY OF REPORTED EXPERIMENTS IN TSR (RQ2)

Experimentation plays a vital role in providing insights about the quality of a software product. Kitchenham *et al.* [43] emphasized on reporting guidelines, which are useful to all types of empirical studies. Motivated by this, researchers have proposed several reporting guidelines for experiments [43]–[45]. However, to the best of our knowledge, there is no ''de-facto standard'' or ''unified guidelines'', which can be used to systematically conduct and report the outcome of an experiment related to TSR.

A number of experiments have been conducted to determine the cost-effectiveness of TSR approaches (S14; S26; S39; S43; S47; S52; S55; S57; S59; S63; S79; S94) or to judge the impact of TSR on fault-detection capability loss (S3; S6; S11; S16; S30; S66). Thus, it is important to evaluate the quality of experiments reported in the literature.

TABLE 9. Overview of quality assessment criteria.

Identifier	Experiment phase	Criteria	Description
QAC1	Scoping	goal definition	clear statement of the goals using goal/question/ metric.
QAC2	Planning	context	clear statement of context such as industrial or academic setting.
QAC3		null hypothesis	formal statement of the hypothesis.
QAC4		variables	identification of independent and dependent variables including, cost, effectiveness, and scalability measures.
QAC5		experiment design	selection of an appropriate standard design types such as one factor with two treatments.
QAC6	Analysis and Inter-pretation	comparison baseline	selection of a suitable benchmark(s).
QAC7		descriptive statistics	appropriate reporting of descriptive statistics, such as measure of central tendency, dispersion, and dependency, for selected cost, effectiveness, and scalability measures.
QAC8		hypothesis testing	decision about acceptance or rejection of hypothesis.
QAC9		scalability analysis	consideration of scalability issues.
QAC10		validity threats	consideration of possible risks associated with selected study such as internal, external, conclusion, and construct threats.

Moreover, the ultimate objective is to derive an initial set of recommendations useful for conducting and evaluating the future experiments on TSR. Therefore, we have adopted a systematic methodology to achieve these objectives. First of all, we have identified a set of quality assessment criteria for evaluating experiments. Next, we have investigated the quality of selected studies based on the defined quality assessment criteria. After that, we have provided overall trends of TSR experiments. Finally, we have presented a set of recommendation that can help in improving the quality of TSR related future experiments.

A. QUALITY ASSESSMENT CRITERIA

To assess the quality of published experiments on TSR, a set of criteria focusing three main experiment phases including, experiment scoping, planning, and results analysis and interpretation, is identified according to the principles of a well-designed software engineering experiment provided by Wohlin *et al.* [46]. Table 9 provides an overview of quality

TABLE 10. Quality assessment score attained by the selected studies.

Study	QAC1	QAC2	QAC3	QAC4	QAC5	QAC6	QAC7	QAC8	QAC9	QAC10
S3	N	Y	P	N	N	N	Y	Y	Y	N
S6	N	Y	P	N	N	N	N	N	N	N
S11	N	Y	P	Y	Y	N	Y	Y	Y	Y
S14	N	P	N	N	N	Y	N	N	Y	N
S16	N	Y	Y	N	Y	Y	Y	Y	N	N
S26	N	P	N	N	N	Y	N	N	Y	N
S30	N	P	P	N	N	Y	N	N	N	Y
S39	N	P	N	N	N	Y	N	N	N	N
S43	N	P	Y	Y	Y	Y	Y	Y	Y	Y
S47	N	P	P	Y	N	N	Y	N	Y	N
S52	N	Y	N	N	N	Y	N	N	Y	N
S55	N	Y	N	N	N	Y	N	N	Y	Y
S57	N	P	N	N	N	Y	Y	N	Y	N
S59	N	Y	N	Y	N	N	N	N	Y	Y
S63	N	P	N	N	N	N	N	N	Y	N
S66	N	Y	P	Y	N	N	Y	Y	Y	Y
S79	N	P	N	N	N	Y	N	N	Y	N
S94	N	Y	P	Y	N	N	Y	Y	N	Y

Legends: Y= yes; N = no; P = partly

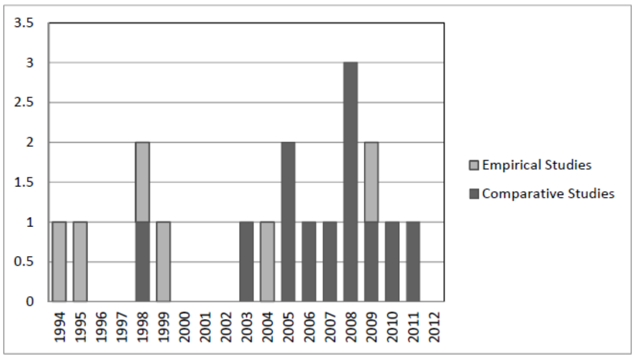


FIGURE 5. Number of studies conducted per year.

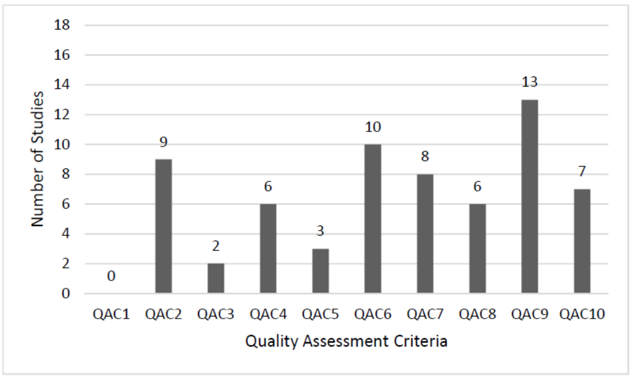


FIGURE 6. Number of studies satisfying quality assessment criteria. The number on bar represent the number of performed experiments (studies) that satisfy the defined quality assessment criteria (QAC). For example, 2 studies out of 18 fulfill the QAC3.

assement criteria, which ultimately helps in evaluating the quality of experiments reported in the literature related to TSR.

B. QUALITY ASSESSMENT SCORE

Based on the defined quality assessment criteria (Table 9), we have evaluated the quality of reported TSR experiments. Table 10 presents the quality assessment results for the selected studies.

C. TRENDS OF TSR EXPERIMENTS

Researchers have conducted a significant number of studies in the field of TSR. In total, 18 studies were selected, of which 12 focused on determining the cost-effectiveness of TSR approaches (S14; S26; S39; S43; S47; S52; S55; S57; S59; S63; S79; S94) and six aimed at judging the impact of TSR on fault-detection capability loss (S3; S6; S11; S16; S30; S66). Fig. 5 shows the number of studies conducted on a yearly basis starting from 1994, when Wong *et al.* [47] performed the first ever empirical study after the introduction of two TSR heuristics (S1; S2) in order to measure the fault-detection loss. The publication trends on TSR experiments indicate that researchers have mainly focused on conducting comparative studies. It is mainly due to the fact that significant numbers of TSR approaches have been proposed. Consequently, researchers were interested to determine best TSR approach from other compared approaches.

The following sub-sections present the quality assessment results of reported TSR experiments according to the defined quality assessment criteria (Section V(A)), which covers three main phases of an experiment: (i) project scoping, (ii) planning, and (iii) analysis and interpretation.

1) EXPERIMENT SCOPING

In current state-of-the-art TSR experiments, none of the selected study used Goal/Question/Metric to explicitly define the *scope* (QAC1) of the experiment (see Fig. 6). Wohlin *et al.* [46] termed experiment scope as the ‘‘definition of the experiment’’, which should be explicitly defined to provide clear and unambiguous definition regarding the experiment objective. In contrast, nine studies (S3; S6; S11; S16; S52; S55; S59; S66; S94) reported the *context* (QAC2). Moreover, six studies (33%, 6/18) implicitly (partly) discussed the online context of the experiments using two benchmark programs including Siemens programs [48] and Space programs [49]. Four studies applied Siemens programs (S11; S16; S55; S66) and three used Space programs (S3; S6; S59) for evaluation purposes.

The selected studies implicitly address the specific context of experiments. For example, 12 studies (S14; S26; S39; S43; S47; S52; S55; S57; S59; S63; S79; S94) focus on finding the best TSR approach to determine the Reduced Suite (RS), while six studies (S3; S6; S11; S16; S30; S66) emphasize on evaluating the impact of TSR on Fault Detection Capacity (FDC).

We found nine studies (50%, 9/18) that lack in discussing the *hypothesis* (QAC3), which poses external validity threats related to their results. Only two studies (11%, 2/18) formally define the hypothesis (S16; S43), whereas nine studies (50%, 9/18) partly define the hypothesis (S3; S6; S11; S16; S43; S30; S47; S66; S94).

Among all, six studies (33%, 6/18) reported different independent and dependent *variables* (QAC4) with no bias toward any TSR approach (S11; S43; S47; S59; S66; S94). To measure the cost-effectiveness of conducted studies, researchers have employed a number of cost and effectiveness measures. For example, four studies (22%, 4/18) considered only one effectiveness measure such as Percentage of Test Suite Reduction (PTSR) (S14; S26; S55; S79). Additionally, 14 studies (78%, 14/18) used two effectiveness measures (i.e., PTSR and FDC). Similarly, 19 studies used different types of coverage criteria as a measure of effectiveness. It was observed that the conducted studies in TSR mostly employ greedy-based approach, but only one study (S79) consider search-based approach. Furthermore, 14 studies (74%) used mutation score to measure the FDC-based effectiveness of TSR approaches. In contrast, we found six cost measures. Three studies (17%, 3/18) lack in using any cost measure (S39; S52; S57), while the remaining 15 studies used only one cost measure. The most prominent cost measure in 10 studies (56%, 10/18) was Fault Detection Loss (FDL). Moreover, four studies employed a single cost measures, including test suite size (S16), Percentage of increase in Expense of Fault Localization (PEFL) (S59), Ratio of Greedy Choice (RGC) (S63), and test overlapping ratio (S14). Additionally, two studies (11%, 2/18) applied algorithm running time as a cost measure to compare the performance of greedy-based and search-based approaches (S55; S79). Alternatively, all selected studies considered different scalability measures such as size of System under Test (SUT) and number of test cases. We found 13 studies (72%, 13/18) that reported different scalability measures, while four studies (22%, 4/18) failed to report SUT size (S26; S39; S43; S79).

Only three studies (17%, 3/18) mentioned standard experimental *design* (QAC5) in TSR (S11; S16; S43). For instance, the most commonly used standard design was one factor with two treatments.

2) EXPERIMENT ANALYSIS AND INTERPRETATION

A significant number of studies (44%, 8/18) lack in considering any *comparison baseline* (QAC6) technique. We found five studies (28%, 5/18) which used random techniques as a comparison baseline (S16; S26; S30; S39; S52), and three studies (17%, 3/18) used RS size a baseline technique (S55; S57; S79). Additionally, three studies (S14; S16; S79) applied unique comparison baseline techniques, including mutation adequacy ratio (S16), test overlapping ratio (S14), and algorithm running time (S79). Furthermore, one study (S43) employed greedy algorithm as a comparison baseline technique.

Among all, 10 studies (56%, 10/18) failed to report any class of *descriptive statistics* (QAC7). In total, five studies mentioned the central tendency including, mean and median (S16; S11; S94; S43; S47), four referred to measures of dispersion including, standard deviation (S43; S57; S66; S94), and three studies mentioned dependency measures such as Spearman, Kendall, Pearson's r-correlation, and square of correlation (S3; S66; S11).

We found, six studies (S3; S11; S16; S43; S66; S94) that discussed the results of *hypothesis testing* (QAC8) such as using one sample parametric t-test (S43). In contrast, four studies (22%, 4/18) performed a statistical significance test such as F-test, t-test, paired t-test, ANOVA, and Bonferroni test (S16; S43; S47; S94). Moreover, four studies reported practical significance test (S16; S43; S47; S94), while 13 studies other than (S6; S16; S30; S94; S39) reported *scalability analysis* (QAC9) results.

Seven studies (39%, 7/18) discussed *validity threats* (QAC10) including, internal and external validity threats, whereas only one study (S43) reported conclusion validity threats. Conversely, three studies (17%, 3/18) reported construct validity threats (S11; S59; S94).

It is evident from Fig. 6 that experiment scope and planning activities (QAC1 to QAC5) have received less attention from researchers compared to experiment analysis and interpretation activities (QAC6 to QAC10). Certainly, lack of considering experiment scope and planning activities can negatively impact on the experiment's results. Moreover, researchers have given varying attentions to the experiment analysis and interpretation activities. In particular, scalability analysis (QAC9) attained highest focus from researchers compared to other activities of experiment analysis and interpretation. However, significant number of studies lack in achieving satisfactory quality assessment score due to non-availability of a "de-facto standard" or "unified guidelines", which can be used to systematically conduct and report the outcome of a TSR experiment. Therefore, a specialized set of recommendations is required to help in improving the quality of future experiments in the field of TSR.

D. RECOMMENDATIONS FOR CONDUCTING RELIABLE EXPERIMENTS IN TSR

This section presents the recommendations based on the overall trends of TSR experiments, which would be useful in improving the quality of future experiments. First of all, we briefly discuss the basic experiment principles adopted from [46], which are universal for experiments in any field and may be tailored according to each discipline.

(i) *Hypothesis*: In theory, the fundamental conviction is that a relationship exists between *cause* and *effect* constructs, which is also known as a *hypothesis*. By experimenting, it is possible to test the hypothesis to determine if there actually is a connection between *cause* and *effect* via *experiment operation* [46]. In the current TSR context, this distinctive principle can be defined as follows:

"By applying a *TSR Approach* belonging to either a *Search*, *Greedy*, *Clustering,* or *Hybrid*-based approach one can achieve similar/same effectiveness (i.e., *Effect(RS)*$\leq$*Effect(TS)*) with smaller cost (i.e., *Cost(RS)* $<$ *Cost(TS)*)"

To test such hypothesis, an experiment needs to be conducted. Below, typical experiment terminology is defined to facilitate discussion in subsequent sections.

(ii) *Variables*: In an experiment, there are two types of variables: (i) independent and (ii) dependent. The *independent* variables are the ones which are manipulated and whose effects are observed on *dependent* variables [46]. In our context, an example of typical *independent* variables may be TSR approaches: e.g., *search-based*, *greedy-based*, and their effectiveness in terms of *number of test cases* in the RS along with *coverage* achieved with RS.

(iii) *Factors and treatments*: In experiments, one or more independent variables are typically changed with fixed intervals (called *factors*) while the rest are kept fixed to study their effect on the dependent variables. A *treatment* is one instance of factors [46]. In the current perspective, a *TSR Approach* is a factor and *search-based* is a treatment.

(iv) *Objects and subjects*: A *treatment* is applied to combinations of *Objects* and *Subjects*. In our context, for instance, a system to be tested (e.g., a Java program) is an *object*. *Subjects* are people applying *treatments* to *objects*. In our TSR situation there are no explicit *subjects* since the proposed algorithms perform TSR automatically.

1) EXPERIMENT SCOPING

Experiment scoping is the first activity in an experimental process, which focuses on defining the goals of an experiment.

Recommendation 1: A good quality experiment in TSR must explicitly define the scope of the experiment using Goal/Question/Metric (GQM) template.

As proposed by Wohlin *et al.* [46], a Goal/Question/Metric (GQM) template should be followed to define the goal of an experiment as defined in [50].

Analyze < Object(s) of study >
for the purpose of < Purpose >
with respect to their < Quality focus >
from the point of the view of the < Perspective >
in the context of < Context >

The objective of a typical experiment in our context can be formulated as follows:

Analyze < A set of Java Programs >
for the purpose of < Comparing TSR Approaches >
with respect to their < Cost and Effectiveness >
from the point of the view of the < Testers >
in the context of < an Industrial setting >

2) EXPERIMENT PLANNING

Upon defining the experiment scope using a GQM template, the next activity in the experimental process, namely experiment planning is initiated. It consists of a number of planning steps to achieve the defined experiment scope. Moreover, it serves as a road map to monitor the experiment's progress.

Recommendation 2: A well-designed experiment in TSR must clearly specify the context along with the objects of the experiment.

Context selection is the first stage in experiment planning, whereby clarifying the context (i.e., industrial or academic settings) in which the experiment will be executed plays an important role. Ideally, an experiment must be executed with real-world *'objects'* in an actual setting to generalize the results, but in practice it is not possible due to time and resource limitations. Wohlin *et al.* [46] has mentioned four dimensions of context selection: (1) offline vs. online, (2) student vs. professional, (3) specific vs. general, and (4) toy vs. real problems. It should be noted that real-world problems actually come from the industry (i.e., industrial application). In contrast, toy problems are artificial problems devised by researchers to evaluate the efficacy of proposed TSR approaches.

From a TSR approach standpoint (as mentioned in Section IV), humans do not normally perform the experiments, for which reason dimension *2* of context selection is not relevant in our case. However, the context of the experiment for TSR approaches can be classified into the other three dimensions. Note that lack in explicitly reporting the experimental context poses external validity threats.

Recommendation 3: A well-designed experiment in TSR must clearly specify the hypotheses to facilitate hypothesis testing.

To facilitate statistical tests, it is essential to formally state the hypotheses. Researchers have suggested to specifying the following two hypotheses:

Null Hypothesis (H_0): It is assumed that there are no significant differences in the experiment settings and the aim is rejection of the hypothesis with as high significance as possible. In the present context of TSR, an example of a null hypothesis is:

H_0: There is no significant difference between search-based (*Search*) and greedy-based (*Greedy*) TSR approaches in terms of achieving high statement coverage with RS (20):

$$H_o : \mu_{Effect}(Search) = \mu_{Effect}(Greedy) \tag{20}$$

where μ is the average and *Effect( )* is the effectiveness (discussed in Section IV(A)). Alternative Hypothesis (H_1): This is an alternate to the null hypothesis, whereby it is known beforehand that one approach is superior to another (21):

$$H_1 : \mu_{Effect}(Search) < \mu_{Effect}(Greedy) \tag{21}$$

Appropriate statistical tests are chosen according to the hypotheses. Testing these hypotheses may involve different risks such as Type-I-error and Type-II-error. For further details regarding these errors, please consult [51].

Recommendation 4: A well-designed experiment in TSR must use percentage of test suite reduction (PTSR) with at least one other effectiveness measure such as percentage of coverage achieved (PCOV) or fault detection capability (FDC).

To judge the effectiveness of TSR approaches, a number of effectiveness *variables* (or measures) have been applied in experiments.

- *Percentage of Test Suite Reduction (PTSR)*: In any TSR approach, the key purpose is to obtain a RS from TS. This means that the effectiveness of a TSR approach can be measured based on the size of RS or by simply calculating a percentage of reduction [47], [52] as mentioned in (22).

$$PTSR = (1 - \frac{Size(RS)}{Size(TS)}) \times 100 \tag{22}$$

where *Size* is a function that returns the number of test cases in a test suite and PTSR is the percentage of test suite reduction. The purpose is to obtain a higher PTSR with a TSR approach, which ultimately has a significant impact on effectiveness. It is worth mentioning that measuring PTSR alone is not sufficient and it must be combined with other measures, such as coverage and fault detection. This is mainly due to the fact that by having an empty set of *RS*, it is possible to achieve 100% reduction using the above formula.

- *Percentage of Test Time Reduction (PTTR)*: Similarly, the effectiveness of a TSR approach can be measured based on the computational time taken by RS or by simply calculating percentage of test time reduction [53] mentioned in (23).

$$PTTR = (1 - \frac{Time(RS)}{Time(TS)}) \times 100 \tag{23}$$

where *Time* is a function that returns the total computational time taken by test cases in a test suite and the objective is to get a higher PTTR with a TSR approach.

- *Percentage of Coverage Achieved (PCOV)*: One typical measure pertaining to TSR is to achieve higher coverage with *RS*. The type of coverage varies depending on the technique, such as code coverage and model coverage, as discussed in greedy-based approaches Section 3.2.1. This measure is combined with PTSR with the aim of achieving higher *Coverage* with fewer test cases in the RS. The objective can be defined in (24):

$$PCOV = (\frac{\text{number of elements covered by RS}}{\text{total number of elements covered by TS}}) \times 100 \tag{24}$$

Overall, the goal is to attain higher PTSR and PCOV with a TSR approach. When comparing two TSR approaches (A1 (e.g., search-based) and A2 (e.g., greedy-based)), a particular approach A1 is considered better than A2 if:

$$PTSR(A1) < PTSR(A2) \text{ and } PCOV(A1) > PCOV(A2) \tag{25}$$

The significance of the results can be determined from significance tests.

Likewise, greedy algorithms use effectiveness measures (i.e., *PCOV* and *Time*) to determine the best test case (tc_i) to this point by applying the ratio of greedy choice (RGC) [53], refer to (26):

$$RGC(tc_i) = (\frac{PCOV(tc_i)}{Time(tc_i)}) \tag{26}$$

- *Fault Detection Capability (FDC)*: Another criterion of measuring the effectiveness of a TSR approach is related to the Fault Detection Capability (FDC) of RS [47]. FDC is defined in (27).

$$FDC = (\frac{\text{number of faults detected by RS}}{\text{total number of faults detected by TS}}) \tag{27}$$

Based on our example, it can be said that A1 is better than A2 if:

$$PTSR(A1) < PTSR(A2) \text{ and } FDC(A1) > FDC(A2) \tag{28}$$

Recommendation 5: A well-designed experiment in TSR must use percentage of test suite reduction (PTSR) with at least one other effectiveness measure, and at least one cost measure.

Many cost measures have been used in selected studies, which consequently help in evaluating the efficiency of TSR approaches. In the following, we present the most frequently employed cost measures.

- *Cost of TSR algorithm*: An algorithm performing TSR requires a certain amount of time to find the RS. When comparing two different TSR approaches, such as search-based and greedy-based, it is essential to use a cost criterion that is analogous to both approaches being compared. For instance, in a search-based approach a search algorithm is normally run for a certain number of generations. Thus, if the number of generations serves as a cost criterion, the search-based approach cannot be compared with the greedy-based since there is no concept of "*generations*" in the greedy-based approaches. Instead, using time as a cost criterion could be among the comparable criteria across the TSR approaches. However, the time taken by TSR approaches to find RS must be measured in a unified way for all approaches, such as using the same machine to reduce any bias in the results. The time TSR requires is measurable in terms of CPU cycles or clock time. Although clock time is not clearly analogous in all various hardware designs, it is convenient in deciding whether a method is, in reality, applicable. CPU cycles are a valid measure for comparing various techniques and different hardware architectures too. However, when comparing two approaches from the same TSR category (e.g., search-based), different cost measures can be used. For instance, in case of search-based approaches, the number of generations may be one of the comparable criteria across the two search-based techniques. Interested readers may consult [54] for more information on comparing two search-based approaches.

- *Cost of executing test cases in RS*: Although the number of test cases in RS can provide a rough measure of cost to execute test cases since the number of test cases in RS will be less than TS and thus will take less time to execute than TS. However, when comparing two distinctive approaches, two different RS sets may be produced and thus, simply comparing them on the basis of number of test cases in RS is not an accurate measure. An explanation may be that test execution time for each test case is not uniform and therefore, the time to execute a RS with fewer test cases is potentially greater than the time taken to execute RS with more test cases.
- *Fault Detection Loss (FDL)*: A typical cost measure of a TSR approach is to determine the Fault Detection Loss (FDL) of RS [47], and defined by (29).

$$FDL = \left(\frac{\text{number of faults missed by RS}}{\text{total number of faults detected by TS}}\right) \qquad (29)$$

Similarly, the other cost measure is based on TS execution information. For example, greedy-based fault localization heuristics determine the possible faulty portion(s) of a SUT using the execution information of TS. PEFL is the percentage of increased expense of fault localization [11], and defined by (30).

$$PEFL = \left(\frac{\text{rank of faulty statement}}{\text{total number of executable statements}}\right) \qquad (30)$$

Diverse cost and effectiveness measures used in different approach pairs are summarized in Table 11.

Recommendation 6: A well-designed experiment in TSR must include scalability measures to assess whether the proposed TSR is applicable to a wide range of problems.

Evaluating the way in which a TSR method's cost effectiveness progresses with the rise in TSR problem intricacy is known as scalability assessment. One or more SUT size measures are implicated, besides their link to the researched TSR approaches' effectiveness or cost. Instances of measures with potential for improvement can be SUT size with regards to code lines or search space dimension pertaining to the array and amount of input-parameters. After that scaling impact is studied on diverse effectiveness and cost measures to deduce whether the TSR remains cost-effective with increasing, more complex SUT.

TABLE 11. Cost and effectiveness measures for different pairs of approaches.

Approaches Pairs	Effectiveness Measures	Cost Measures
greedy vs. greedy	PTSR, PTTR, PCOV, FDC, RGC	algorithm running time, mutants management time, test overlapping ratio, size of TS, test execution time of RS, FDL, PEFL
search vs. search	PTSR, FDC	number of fitness evaluations, number of generations, cumulative number of individuals in all generations, total searching time for RS, number of CPU cycles, size of TS, test execution time of RS, FDL
clustering vs. clustering	PTSR, FDC, PTTR	test overlapping ratio in clusters, size of TS, test execution time of RS, FDL
greedy vs. search	PTSR, FDC, PTTR	mutants management time, size of TS, test execution time of RS, FDL
greedy vs. clustering	PTSR, FDC, PTTR	mutants management time, size of TS, test execution time of RS, FDL
search vs. clustering	PTSR, FDC, PTTR	mutants management time, size of TS, test execution time of RS, FDL

Legends: PTSR = percentage of test suite reduction; PTTR = percentage of test time reduction; PCOV = percentage of coverage achieved; FDC = fault detection capability; RGC = ratio of greedy choice; TS = test suite; RS = reduced suite; FDL = fault detection loss; PEFL = percentage of increase in expense of fault localization

Recommendation 7: A well-designed experiment in TSR must explicitly specify the experimental design and a justification for using it.

An experiment must be planned and designed carefully to minimize validity threats as much as possible. There are several standard experiment designs proposed in the literature; however, this paper maps a few designs related to TSR. An interested reader may consult the work provided in [46] for additional designs, and the current mapping is extensible to other designs. Table 12 provides a list of the most regularly used experiment designs and their mapping as pertinent to TSR, with examples.

In Table 12, H_0 and H_1 represent null hypothesis and alternative hypothesis, respectively; $\mu_{\text{Search}_\text{F}\text{DC}}$ denotes the expected mean value of Fault Detection Capability (FDC) of the search-based approach and $\mu_{\text{Greedy}_\text{F}\text{DC}}$ indicates the expected mean value of FDC for the greed-based approach. So $\mu_{\text{Search}_\text{F}\text{DC}} = \mu_{\text{Greedy}_\text{F}\text{DC}}$ signifies that the projected FDC mean values using search-based and greedy-based approaches are the same; $\mu_{\text{Clustering}_\text{P}\text{TSR}}$ represents the expected mean value of Percentage of Test Suite Reduc-

TABLE 12. Typical experiment designs and their mapping to test suite reduction.

Design	Factor(s)	Treatment (s)	Dependent	Hypotheses	Statistical Tests
one factor with two treatments	TSR approach	search, greedy	FDC, PTSR	H_0: $\mu_{Search\_FDC} = \mu_{Greedy\_FDC}$ H_0: $\mu_{Search\_PTSR} = \mu_{Greedy\_PTSR}$ H_1: $\mu_{Search\_FDC} \neq \mu_{Greedy\_FDC}$, $\mu_{Search\_FDC} < \mu_{Greedy\_FDC}$ or $\mu_{Search\_FDC} > \mu_{Greedy\_FDC}$ H_1: $\mu_{Search\_PTSR} \neq \mu_{Greedy\_PTSR}$, $\mu_{Search\_PTSR} < \mu_{Greedy\_PTSR}$ or $\mu_{Search\_PTSR} > \mu_{Greedy\_PTSR}$	t-test, Mann-Whitney
one factor with more than two treatments	TSR approach	search, greedy, clustering	FDC, PTSR	H_0: $\mu_{Search\_FDC} = \mu_{Greedy\_FDC} = \mu_{Clustering\_FDC}$ H_0: $\mu_{Search\_PTSR} = \mu_{Greedy\_PTSR} = \mu_{Clustering\_PTSR}$ H_1: $\mu_{i\_FDC} \neq \mu_{j\_FDC}$, for at least one pair (i, j) H_1: $\mu_{i\_PTSR} \neq \mu_{j\_PTSR}$, for at least one pair (i, j)	ANOVA, Kruskal-Wallis for comparing across more than two treatments, and t-test or Mann-Whitney for pair-wise comparison
two factors	search-based approach; fitness	search1, search2,	PTSR	H_0: $\tau_i = \tau_j = 0$ H_1: at least one $\tau_i \neq 0$ H_0: $\beta_i = \beta_j = 0$ H_1: at least one $\beta_i \neq 0$ H_0: $(\tau\beta)_{ij} = 0$ for all i, j H_1: at least one $(\tau\beta)_{ij} \neq 0$ τ, β are search1 and search2, whereas i, j are fitness1 and fitness2	ANOVA

Legends: TSR = test suite reduction; FDC = fault detection coverage; PTSR = percentage of test suite reduction; μ = expected mean value; μ_{Search_FDC} = expected mean value of FDC for the search-based approach; τ and β represent two different single search algorithms

tion (PTSR) for the clustering-based approach; μ_{iFDC} denotes the expected mean value of FDC for any TSR approach employed such as greedy, search, or clustering). Likewise, μ_{jPTSR} indicates the expected mean value of the PTSR of any TSR approach; $\mu_{\mathrm{iFDC}} \neq \mu_{\mathrm{jFDC}}$ shows that the expected mean value of FDC of at least one pair (i, j) is not the same, where i and j belong to a TSR approach; $\tau_i = \tau_j = 0$ and $\beta_i = \beta_j = 0$ signify that employing a similar single search algorithm (i.e., τ or β) with two different fitness functions (i.e., i and j) ascertains the same result. In contrast, $(\tau\beta)_{ij} = 0$ indicates that employing both search algorithms using all fitness functions produces the same result.

3) EXPERIMENT ANALYSIS AND INTERPRETATION

Analysis and interpretation are among the important activities in the experiment process, which facilitate analysis and evaluation of data collected after experiment scoping and planning activities. Consequently, analysis activity assists researchers in deducing meaningful conclusions about the conducted experiment.

Recommendation 8: A well-designed experiment in TSR must compare the proposed TSR approach with at least one simple alternative baseline, such as Random technique.

A TSR approach can only be evaluated by comparing it with a carefully selected and meaningful baseline, otherwise it is not possible to judge if the proposed TSR approach is better than any existing one. Since absolutely evaluating TSR methods might be tricky, it is significant to at least indicate that the issue cannot be dealt with in an easier way. Essentially, each study ought to employ at least one comparison baseline in evaluating TSR approaches, while the bare minimum should be some contrast against *Random* [55]. Even though the SUT being assessed might be straightforward and small, and a TSR approach functioning well might not be very relevant. Thus, *Random* might facilitate fundamental confirmation that a simple algorithm might not be able to deal with the TSR issue and a more intricate method may be required. Various straightforward TSR techniques the likes of *Random* are also recommended to act as a comparison baseline.

Following baseline approach selection, plausibly efficient implementation should be considered so that effectiveness and cost can be compared. Credentials, downloadable tool URLs, a source code, or at least a good account of the implementation needs to be available.

Recommendation 9: A well-designed experiment in TSR must report descriptive statistics for cost, effectiveness, and scalability measures.

It is better for cost, together with effectiveness, and perhaps scalability to be stated, such as in the form of descriptive statistics followed by analysis. Observing the standard deviation could be indicative of the uncertainty level with respect to the effectiveness and cost related with TSR approaches.

Descriptive statistics typically provide initial insight into results, for instance, comparing mean values of the number of test cases in RS achieved by two different TSR approaches. Descriptive statistics normally come into play prior to hypothesis testing to determine if the differences between two or more approaches are statistically significant or not. Descriptive statistics are classified into three main classes and are summarized in Table 13 along with examples related to TSR.

Recommendation 10: A well-designed experiment in TSR must report hypothesis testing results together with a justification of test selection.

Statistical testing is meant to identify if any discrepancies in TSR approaches regarding basic cost and effectiveness inclinations occur by chance or, in fact, follow any specific movement. Statistical hypothesis testing is valuable particularly in relation to search-based TSR approaches, which always interrelate with certain random disparity in terms of effectiveness and cost [41]. Since statistical testing comprises typical practice, it will not be detailed more here but for further reading D. J. Sheskin's [51] book may be of interest.

TABLE 13. Commonly used descriptive statistics and examples in test suite reduction.

Descriptive Statistics Type	Commonly used Measures	Examples in TSR
measures of central tendency	mean, median, mode	mean (PTSR) of a greedy-based approach on a set of objects
measures of dispersion	standard deviation, variance	standard deviation (PTSR) on a set of objects
measures of dependency	Spearman, Kendall, and Pearson correlation coefficient	Pearson correlation coefficient between PTSR and PCOV of a greedy-based approach on a set of objects

Legends: PTSR = percentage of test suite reduction; PCOV = percentage of coverage achieved

Statistical hypothesis testing may serve for rejecting or accepting hypotheses associated with TSR approach cost effectiveness analysis as well as comparison baselines. Statistical test selection is dependent upon the objective specified. It is customary in different fields to choose particular statistical tests and justify them according to the compared samples' data distributions so as to avert incorrectly making deductions from analysis. The statistical tests are frequently categorized as either parametric or non-parametric [51]. Particular parametric tests are valid, such as a t-test, if a sample has a specific distribution, for instance, normal. Otherwise, non-parametric statistical tests are applied when it is not possible to make any adequate assumptions regarding sample distribution. Data normality may be established via suitable statistical tests, such as the Shapiro-Wilk W test [51]. Notions pertaining to appropriate test selection will not be discussed here in further detail and are available in standard texts.

Table 14, provides a mapping that links analysis situations to the types of appropriate statistical tests (for simplicity, two samples are assumed, that is, two TSR approaches are compared). The mapping has been explained with examples.

Table 15 indicates that search-based approaches are randomized and must be run more than once to reduce the random variation in the results, since each run may produce varying results. Nonetheless, greedy/clustering-based approaches may produce the same results regardless of whether they are run more than once. In this case, one-sample tests were carried out because the values for cost/effectiveness measures will be the same.

Recommendation 11: A well-designed experiment in TSR must report the results of scalability analysis.

Scalability helps in assessing whether a TSR approach is applicable to either bigger or more intricate SUTs while at the same time it is economical and adequately effective. If an experiment is meant to demonstrate how scalable a TSR approach is, suitable complexity and size measures ought to be visibly characterized. A minimum of two measures will be concerned: one related to size, which is scaled up via consecutive SUTs, and second, another measure of analogous performance (effectiveness and cost). As such, the impact of

TABLE 14. Mapping of test suite reduction approaches to statistical tests.

Comparison	Type of analysis	Type of statistical comparison	Example	Type of statistical test (assuming two samples)
search vs. search	comparing samples of runs in terms of effectiveness (e.g., PTSR) and cost (average time taken by search-based algorithm)	comparing central tendencies of two or more independent samples, each corresponding to a search-based approach	comparing maximum statement coverage achieved across all runs between two search-based approaches	Parametric *t*-tests or Non-Parametric Mann-Whitney U test
search vs. search	comparing samples of target (e.g., 80% branch coverage) in terms of cost (e.g., number of test cases) to reach them	comparing central tendencies of matched pairs across samples	comparing each Search approach, according to which a target is achieved. Note that the observations across samples are paired as they correspond to identical target.	Parametric Paired *t*-tests or Non-Parametric Wilcoxon or Sign test
Search vs. Greedy/ Search vs. Clustering	comparing samples of runs in terms of effectiveness and cost	comparing central tendencies of two or more independent samples, each corresponding to a TSR approach	comparing maximum branch coverage achieved across all runs between a search-based and greedy/clustering-based approach	one sample Parametric *t*-tests or one sample Non-Parametric Mann-Whitney U test
Search vs. Greedy/ Search vs. Clustering	comparing samples of target (e.g., 80% branch coverage) in terms of cost (e.g., number of test cases) to reach them	comparing central tendencies of matched pairs across samples	comparing each search approach, according to which a target is achieved. Note that the observations across samples are paired as they correspond to identical target.	one sample Parametric Paired *t*-tests or one sample Non-Parametric Wilcoxon or Sign test
Greedy vs. Clustering	-	-	-	-

Legends: PTSR = percentage of test suite reduction; TSR = test suite reduction

scaling up a certain measure may be detailed with respect to a statistical relationship (recollect the inevitable random variation). For instance, it is possible to examine a number of different sized SUTs with respect to code lines, after which appraise if a TSR approach is able to attain a particular coverage degree at tolerable cost for bigger SUTs and assess the way this expenditure progresses in relation to SUT size. Problems may consequently be posed by a positive exponential cost-size relationship, since the method's pertinence to large-scale test models and systems would be undermined. Likewise, scalability matters would also arise if effectiveness, for instance in terms of coverage attained, greatly decreases as a function of SUT size [41].

Concerning scalability investigation, it is necessary to distinguish the link among SUT size variables and the

TABLE 15. Distribution of test suite reduction approach classification.

TSR approaches	Greedy-based		Search-based	Clustering-based	Hybrid		Total
	Approximate (Greedy)	Exact			Intra-Hybrid	Inter-Hybrid	
No. of proposed approaches	63	2	19	3	4	4	95
Percentage	67%	2%	20%	3%	4%	4%	100%

effectiveness and cost measures of the TSR approach. These sorts of methods are normally investigated via regression analysis. Nonetheless, since the SUTs being studied are practically most likely few, this type of analysis is expected to be qualitative; basically, it would merely found on discerning scatter plots within the space of size and cost effectiveness.

Recommendation 12: A well-designed experiment in TSR must report threats to validity as well as how these threats have been dealt within the experiment.

Any experimental study should consider the validity threats starting from experiment scoping to result analysis. The following types of threats should be debated.

- *Construct validity threats*: Cost, SUT size and effectiveness measures need to be suitable and justified, in line with the given investigation's circumstance and objectives. Measures are not likely to be faultless since it is not usually easy to gauge the above-mentioned elements. Nonetheless, it is feasible in reality to compare the scalability and cost-effectiveness of substitute TSR approaches' cost effectiveness and scalability via numerous corresponding effectiveness, cost and SUT size measures.
- *Internal validity threats*: If the performance of a TSR approach is better than another with respect to effectiveness or cost, is it because of something other than the TSR approach? Some possibilities may be: (i) parameter of one or more TSR approaches is poorly defined and (ii) biasness in choosing such SUTs which possess specific features favoring a particular TSR approach.
- *Conclusion validity threats*: Search-based approach necessitates ensuring that random variation gets adequately justified because Search Based Software Testing (SBST) techniques apply Meta-Heuristic Search (MHS) algorithms, which yields result randomness, an intrinsic aspect of meta-heuristic methods. Essentially, enough independent runs need to be carried out to acquire sufficient observations and to permit statistical comparisons.

There should be a vigilant selection of statistical test procedures keeping in view hypothesis technique such as one-tailed or two-tailed, and data gathered (effectiveness and cost distributions) to ensure that the correct statistical test is utilized. Otherwise, particular requisite characteristics of a certain statistical test may be unintentionally debased, resulting in erroneous conclusions. For instance, a common assumption while applying statistical tests is that data is normally distributed.

Are there any virtually noteworthy differences? In retrospective, the extent of variation must be reported, something recognized as effect size, and which helps determine useful result significance.

- *External validity threats*: External validity threats are a complex problem. The outcomes may be generalized if the SUTs which are selected as the sample represents the target application domain. Furthermore, if the considered faults employed to evaluate the test effectiveness are characteristics of real faults. Search-based TSR related experiments should preferably also be carried out on numerous target type SUTs, but all study endeavors face time and resource restrictions. The notion should in any case be circumspectly addressed and the question of why the generalized observation outcome should be answered in a reliable way.

VI. OVERALL RESULTS AND DISCUSSION

The objective of this systematic review is to classify Test Suite Reduction (TSR) approaches and evaluate the quality of TSR experiments based on the defined quality assessment criteria. The TSR approaches are grouped into four main categories: greedy, search, clustering, and hybrid. These approaches mainly differ from each other in terms of type of algorithms used. Greedy-based approaches mostly use greedy algorithms, clustering-based approaches use different clustering and sampling algorithms, and search-based approaches employ various search algorithms. Similarly, hybrid approaches utilize one or more algorithm types to provide superior cost-effective solutions by integrating the strengths of other approaches.

Table 15 shows the distribution of TSR approaches. Among 95 approaches, 69% fall under the greedy-based category, which in turn uses three main sources of coverage including, source code, model, and execution profile. From these, 37 heuristics use source code coverage, 23 employ model coverage, and 5 use execution-profile coverage to obtain the Reduced Suite (RS). Moreover, majority of greedy-based approaches (66%, 63/95) employ various types of greedy algorithms. There are only two greedy-based approaches (i.e., S31, S82) that use exact techniques to determine global optimal solution. Conversely, search-based approaches are gaining attention of researchers to optimally solve the test-size problem. We have identified 19 studies (20%, 19/95) on search-based approaches. However, less attention has been given to hybrid (8%, 8/95) and clustering-based approaches (3%, 3/95). The existing TSR approaches (80%, 76/95) are mainly aimed at solving the single-objective optimization problem. In contrast, some TSR approaches (20%, 19/95) including greedy-based (14 approaches) and

TABLE 16. Distribution of experiment scoping and planning steps in the reported experiments.

Scope (QAC1)	Context (QAC2)	Null Hypothesis (QAC3)	Variables (QAC4)	Experiment Design (QAC5)	Total
0	9	2	6	3	18
0%	50%	11%	33%	17%	

TABLE 17. Distribution of experiment results and analysis steps in the reported experiments.

Comparison Baseline (QAC6)	Descriptive Statistics (QAC7)			Hypothesis Testing (QAC8)	Scalability Analysis (QAC9)	Validity Threats (QAC10)				Total
	Central Tendency	Measure of Dispersion	Measure of Dependency			Conclusion	Internal	External	Construct	
10	5	4	3	6	13	1	7	4	7	18
56%	28%	21%	17%	33%	72%	5%	37%	21%	37%	

search-based (5 approaches) focus on solving the multi-objective optimization issue. To solve the practical problems related to TSR, researchers ought to concentrate on multi-objective optimization problem-based TSR approaches to ultimately facilitate the decision making process for test engineers.

Based on the assessments in Section V(C), it can be concluded that very few experiments in TSR follow the defined quality assessment criteria, which is why the reliability of the evidence cannot be evaluated. The distribution of experiment scoping and planning steps in reported experiments is provided in Table 16. Similarly, the distribution of experiment results and analysis steps in reported experiment is listed in Table 17. The results clearly indicate that experiments conducted in TSR fail to abide by the proposed recommendations. Thus, further experiments should be conducted that follow the suggested recommendations. As a result, the body of credible evidence in TSR literature would improve and researchers would be ensured about TSR usefulness in obtaining the RS for various real-world problems.

VII. VALIDITY THREATS

To assure result accuracy and acceptability from this systematic review, the four following main types of threats to validity were taken into consideration.

A. CONSTRUCT VALIDITY

The objective of the conducted systematic literature review is to obtain a complete set of relevant studies. To achieve this, an unbiased study selection strategy was carried out. Search repositories were chosen to ensure the completeness of study selection, especially with respect to the software engineering and computer science domains. Alternatively, the development of inappropriate search strings may potentially result in incomplete selection of relevant studies. To tackle this threat, the search string was iteratively refined so as to make sure that no relevant study was omitted.

B. CONCLUSION VALIDITY

We have conducted a systematic review, discussed and justified our devised review protocol, which helps in achieving the stated objectives of this review. Consequently, it supports other researchers to replicate the literature review.

Three researchers took part in the study selection process. Of course it is possible to miss some relevant studies, but this has a minor effect on the general classification of TSR approaches.

C. INTERNAL VALIDITY

The inclusion and exclusion criteria of studies have been discussed from various TSR perspectives. For the selection of related studies, three researchers participated in the selection process. However, some relevant studies could be overlooked due to different levels of understanding of the participating researchers. An attempt has been made to avoid such situation by two means. First, a pilot study was conducted to ensure the same level of understanding among researchers. Second, follow-up meetings for consensus were held. An additional threat can be inaccuracy in the data extraction process, something that can negatively impact the classification and assessment of experiments related to TSR. A Microsoft Excel sheet was designed to help the researchers in accurately extracting the required data items. Moreover, follow-up meetings were held to remove anomalies from the extracted data items.

D. EXTERNAL VALIDITY

Keeping in view the types of employed algorithms as well as optimization problems, TSR approaches were grouped into four main categories. This classification may serve for further research because it is founded upon a thorough literature review. So as to ensure the completeness of the relevant studies, all selected studies together with their related references were reviewed.

A specialized set of recommendations for performing TSR related experiments was suggested owing to the poor quality of current experiments in the field of TSR. These recommendations were formulated by considering the principles behind well-designed software engineering experiments [46]. They should be of assistance to researchers in performing quality TSR experiments, as the recommendations have been finalized bearing in mind existing TSR experiment deficiencies exposed during the extensive literature review.

VIII. CONCLUSION AND FUTURE WORK

This paper presents the results of a first systematic review conducted in the field of Test Suite Reduction (TSR). We investigated 113 studies published between a period 1993 to 2016 as a basis to classify existing TSR approaches and to assess the quality of experiments conducted in the area of TSR.

According to our methodical search, 95 TSR heuristics were identified, which were classified into four main categories based on the type of employed algorithms: greedy-based, clustering-based, search-based, and hybrid approaches. As TSR approaches are mainly evaluated empirically, 18 experiments were identified whose quality was evaluated in accordance with the defined quality assessment criteria.

The results of this systematic review indicate that most of the TSR research is related to greedy-based approaches (69%, 65/95) using one or more forms of greedy algorithms to achieve high coverage as quickly as possible based on the assumption that achieving high coverage will reveal more faults. Furthermore, search-based approaches (20%, 19/95) are gaining increasing attention on account of several successful results reported for solving numerous search-based software engineering problems. In contrast, hybrid approaches (8%, 8/95) focus on combining the strengths of other TSR approaches to provide enhanced cost-effective solutions by employing various algorithms. Moreover, the majority of TSR approaches (80%, 76/95) focus on solving the single-objective optimization problem.

Finally, the quality of experiments related to TSR approaches was analyzed and it was observed that the reported experiments do not adhere to the guidelines of well-designed software engineering experiments, and thus may pose validity threats related to their results. The current literature lacks information on guidelines for performing quality experiments in the field of TSR. Our systematic review bridges this research gap by providing a set of appropriate recommendations that may be applied by researchers in future to conduct quality experiments related to TSR.

Based on this systematic review, we suggest the following potential research directions in the field of TSR:

- *Developing model-based TSR* heuristics

It is evident that researchers have mainly (39%, 37/95) focused on developing code-based TSR heuristics. Although, some work has focused on model-based heuristic (24%, 23/95), still further research is required in this class of greedy-based approaches that can fulfill the demands of different domains (e.g., Service Oriented Architecture (SOA), Software Product Lines (SPLs), Model-based Testing (MBT)). As motivational drive to fellow researchers, it has to be said that model-based heuristics possess some characteristics that give an extra advantage over code-based heuristics such as managing complexity with abstraction, facilitating automation, and providing a systematic mechanism.

- *Evaluating proposed TSR heuristic with cost-effectiveness and efficiency measures*

Researchers have mostly used different effectiveness measures, such as size of reduced test suite and coverage achieved, to assess the performance of proposed heuristics. In contrast, some proposed heuristics (55%, 52/95) have been evaluated using various cost measures (e.g., fault detection loss and cost of executing test cases in RS). For wider acceptance of the proposed heuristics, it is crucial to investigate their performance using both cost and effectiveness measures of diverse nature. In addition, studying cost and effectiveness alone is not sufficient and requires studying cost and effectiveness together, for instance, in terms of efficiency measures that can be derived from cost and effectiveness measures. For example, number of test cases to be executed per unit time that can be calculated by combining an effectiveness measure (e.g., number of test cases to be executed) and a cost measure (e.g., time to execute test cases).

- *Comparing the performance of proposed TSR heuristic with consolidated benchmark techniques*

We found 20 studies that used HGS heuristic [18] as a benchmark in order to judge the performance of their proposed heuristic. In contrast, four studies used random technique for evaluation purpose. It is necessary to employ consolidated benchmark techniques, such as HGS heuristic [18] and random technique for unbiased and relative performance evaluation of the proposed heuristic. Ultimately, complete unbiased results can be presented for the considered dataset.

- *Focusing industrial-strength datasets*

The majority of the investigated studies (57%, 54/95) used small-sized subject programs including Siemens [48] and Space Programs taken from Software Infrastructure Repository [49]. Both subject programs are publically available for empirical evaluation. However, practitioners are keen to observe the behavior of proposed TSR approaches on large-scale industrial projects, which represent the real world problems. As a result, the efficacy of the proposed approaches can be better analyzed.

- *Conducting comparative studies*

Only 14% (13/95) of the investigated studies focused on comparing the performance of various TSR approaches. We observed that main research focus in the field of TSR is on proposing TSR approaches (85%, 95/112). Therefore, it is crucial to perform more comparative studies since new TSR approaches are constantly developed and existing ones are evolved.

- *Tackling branchmania situation*

In case of Software Product Lines (SPLs), a feature model may grow exponentially resulting into large number of numerous product variants. This situation is named as *Branchmania* [56] since SPL architecture contains large number of branches, which makes it impossible to provide an optimal product line. One of the viable solutions to deal with branchmania situation is to minimize the number of branches. Therefore, future work should focus on proposing such TSR heuristics that effectively determine and eliminate similarities between the branches of product line architecture. Consequently, it enables the developer to control branchmania situation and maintain a cost-effective product line.

- *Targeting scalability using search algorithms*

Scalability remains a key issue especially for testing large systems. Search algorithms are also referred as 'embarrassingly parallel' since they support natural formu-

TABLE 18. Overview of selected studies.

Study ID	Reference	Publication Type	RQ1	RQ2
S1	[18]	journal	✓	×
S2	[59]	symposium	✓	×
S3	[47]	conference	×	✓
S4	[60]	journal	✓	×
S5	[29]	conference	✓	×
S6	[52]	conference	×	✓
S7	[61]	journal	✓	×
S8	[62]	journal	✓	×
S9	[19]	conference	✓	×
S10	[63]	journal	✓	×
S11	[64]	conference	×	✓
S12	[65]	conference	✓	×
S13	[66]	conference	✓	×
S14	[67]	journal	×	✓
S15	[68]	conference	✓	×
S16	[30]	journal	×	✓
S17	[69]	conference	✓	×
S18	[70]	journal	✓	×
S19	[71]	journal	✓	×
S20	[72]	symposium	✓	×
S21	[73]	journal	✓	×
S22	[74]	journal	✓	×
S23	[75]	conference	✓	×
S24	[76]	journal	✓	×
S25	[77]	journal	✓	×
S26	[38]	symposium	×	✓
S27	[78]	journal	✓	×
S28	[79]	conference	✓	×
S29	[80]	journal	✓	×
S30	[81]	conference	✓	✓
S31	[82]	conference	✓	×
S32	[83]	journal	✓	×
S33	[84]	conference	✓	×
S34	[85]	journal	✓	×
S35	[86]	symposium	✓	×
S36	[87]	conference	✓	×
S37	[88]	symposium	✓	×
S38	[89]	conference	✓	×
S39	[90]	conference	×	✓
S40	[91]	conference	✓	×
S41	[92]	journal	✓	×
S42	[12]	conference	✓	×
S43	[93]	conference	×	✓
S44	[94]	conference	✓	×
S45	[95]	workshop	✓	×
S46	[96]	conference	✓	×
S47	[97]	conference	×	✓
S48	[98]	conference	✓	×
S49	[16]	conference	✓	×
S50	[99]	conference	✓	×
S51	[100]	conference	✓	×
S52	[101]	journal	×	✓
S53	[102]	conference	✓	×
S54	[103]	conference	✓	×
S55	[104]	journal	×	✓
S56	[105]	journal	✓	×
S57	[106]	conference	×	✓
S58	[107]	conference	✓	×
S59	[108]	conference	×	✓
S60	[109]	workshop	✓	×
S61	[110]	journal	✓	×
S62	[111]	conference	✓	×
S63	[53]	symposium	×	✓
S64	[14]	journal	✓	×
S65	[112]	conference	✓	×
S66	[113]	symposium	×	✓
S67	[114]	conference	✓	×
S68	[115]	conference	✓	×
S69	[116]	workshop	✓	×
S70	[13]	conference	✓	×
S71	[117]	conference	✓	×
S72	[118]	journal	✓	×
S73	[119]	conference	✓	×
S74	[120]	conference	✓	×
S75	[121]	conference	✓	×
S76	[42]	conference	✓	×
S77	[122]	conference	✓	×
S78	[123]	journal	✓	×
S79	[124]	journal	×	✓
S80	[125]	conference	✓	×
S81	[126]	conference	✓	×
S82	[127]	journal	✓	×
S83	[128]	conference	✓	×
S84	[129]	journal	✓	×
S85	[130]	conference	✓	×
S86	[131]	symposium	✓	×
S87	[132]	symposium	✓	×
S88	[133]	conference	✓	×
S89	[134]	conference	✓	×
S90	[39]	journal	✓	×
S91	[34]	conference	✓	×
S92	[135]	conference	✓	×
S93	[136]	conference	✓	×
S94	[137]	symposium	×	✓
S95	[138]	conference	✓	×
S96	[139]	conference	✓	×
S97	[140]	conference	✓	×
S98	[22]	conference	✓	×
S99	[141]	journal	✓	×
S100	[142]	conference	✓	×
S101	[143]	conference	✓	×
S102	[144]	conference	✓	×
S103	[145]	conference	✓	×
S104	[146]	conference	✓	×
S105	[24]	conference	✓	×
S106	[23]	conference	✓	×
S107	[147]	conference	✓	×
S108	[148]	symposium	✓	×
S109	[17]	conference	✓	×
S110	[149]	journal	✓	×
S111	[150]	journal	✓	×
S112	[151]	conference	✓	×

lations that can produce significant scalability improvements. For instance, global search algorithms, such as Genetic Algorithm (GA), can easily compute the fitness of each individual solution in a parallel fashion. Similarly, local search algorithm, such as Hill Climbing (HC), requires multiple restarts for a sequential setting and all restarts can be initiated in a parallel manner.

Some previously conducted work has focused on adapting General Purpose Graphics Processing Units (GPGPUs) for regression test case prioritization [57] and selection [58] using search algorithms. Future work should focus on adapting GPGPUs for search-based TSR, which can yield significant scalability improvements for large-size software systems.

APPENDIX

See Table 18.

ACKNOWLEDGMENT

The authors would like to acknowledge Dr. Shaukat Ali for his valuable comments on an earlier version of the manuscript that greatly improve the manuscript. Moreover, the authors would like to acknowledge anonymous reviewers for their valuable comments.

REFERENCES

[1] A. P. Mathur, *Foundations of Software Testing*. 2nd ed. Chennai, India: Pearson Education, 2008.

[2] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001.

[3] C. Catal and D. Mishra, "Test case prioritization: A systematic mapping study," *Softw. Qual. J.*, vol. 21, no. 3, pp. 445–478, 2013.

[4] H. K. N. Leung and L. White, "Insights into regression testing (software testing)," in *Proc. IEEE ICSM*, Miami, FL, USA, Oct. 1989, pp. 60–69.

[5] 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.

[6] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," *J. Softw. Qual. J.*, vol. 12, no. 3, pp. 185–210, 2004.

[7] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. Hoboken, NJ, USA: Wiley, 2011.

[8] S. U. R. Khan, S. P. Lee, R. W. Ahmad, A. Akhwanzada, and V. Chong, "A survey on Test Suite Reduction frameworks and tools," *Int. J. Inf. Manag.*, vol. 36, no. 6, pp. 963–975, 2016.

[9] A. A. Issa, F. A. A. Rub, and F. F. Thabata, "Using test case patterns to estimate software development and quality management cost," *J. Softw. Qual. J.*, vol. 17, no. 3, pp. 263–281, 2003.

[10] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Inf. Softw. Tech.*, vol. 52, no. 1, pp. 14–30, 2010.

[11] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surveys*, vol. 45, no. 1, p. 11, 2012.

[12] X.-Y. Ma, B.-K. Sheng, and C.-Q. Ye, "Test-suite reduction using genetic algorithm," in *Advanced Parallel Processing Technologies*. Berlin, Germany: Springer, 2005, pp. 253–262.

[13] U. Farooq and C. P. Lam, "A max-min multiobjective technique to optimize model based test suite," in *Proc. ACIS*, Daegu, South Korea, May 2009, pp. 569–574.

[14] L. C. Briand, Y. Labiche, Z. Bawar, and N. T. Spido, "Using machine learning to refine category-partition test specifications and test suites," *Inf. Softw. Tech.*, vol. 51, no. 11, pp. 1551–1564, 2009.

[15] D. Qiu, B. Li, S. Ji, and H. Leung, "Regression testing of Web service: A systematic mapping study," *ACM Comput. Surveys*, vol. 47, no. 2, p. 21, 2014.

[16] S.-U.-R. Khan, A. Nadeem, and A. Awais, "TestFilter: A statement-coverage based test case reduction technique," in *Proc. IEEE INMIC*, Dec. 2006, pp. 275–280.

[17] A. Gotlieb and D. Marijan, "FLOWER: Optimal test suite reduction as a network maximum flow," in *Proc. ACM ISSTA*, San Jose, CA, USA, 2014, pp. 171–180.

[18] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Trans. Softw. Eng. Method*, vol. 2, no. 3, pp. 270–285, 1993.

[19] I. Pomeranz and S. M. Reddy, "On the compaction of test sets produced by genetic optimization," in *Proc. IEEE ATS*, Akita, Japan, 1997, pp. 4–9.

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability*. New York, NY, USA: Wh Freeman, 2002.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[22] D. Hao, L. Zhang, X. Wu, H. Mei, and G. Rothermel, "On-demand test suite reduction," in *Proc. IEEE ICSE*, Jun. 2012, pp. 738–748.

[23] C. Zhang, A. Groce, and M. A. Alipour, "Using test case reduction and prioritization to improve symbolic execution," in *Proc. ACM ISSTA*, 2014, pp. 160–170.

[24] D. Blue, I. Segall, R. Tzoref-Brill, and A. Zlotnick, "Interaction-based test-suite minimization," in *Proc. IEEE ICSE*, San Francisco, CA, USA, May 2013, pp. 182–191.

[25] F. Elberzhager, A. Rosbach, J. Münch, and R. Eschbach, "Reducing test effort: a systematic mapping study on existing approaches," *Inf. Softw. Tech.*, vol. 54, no. 10, pp. 1092–1106, 2012.

[26] B. A. Kitchenham, "Guidelines for performing systematic literature reviews in software engineering," Dept. Comput. Sci., Univ. Durham, Durham, U.K., Tech. Rep. EBSE-2007-01, 2007.

[27] T. Dyba, T. Dingsøyr, and G. K. Hanssen, "Applying systematic reviews to diverse study types: an experience report," in *Proc. ESEM*, Madrid, Spain, Sep. 2007, pp. 225–234.

[28] V. Chvátal, "A greedy heuristic for the set-covering problem," *Math. Oper. Res.*, vol. 4, no. 3, pp. 233–235, 1979.

[29] A. J. Offutt, J. Pan, and J. M. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proc. ICTCS*, Washington, DC, USA, 1995, pp. 111–123.

[30] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, "Test set size minimization and fault detection effectiveness: A case study in a space application," *J. Syst. Softw.*, vol. 48, no. 2, pp. 79–89, 1999.

[31] W. Dickinson, D. Leon, and A. Podgurski, "Finding failures by cluster analysis of execution profiles," in *Proc. IEEE ICSE*, May 2001, pp. 339–348.

[32] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull. Waldensian Soc. Nat. Sci.*, vol. 37, pp. 547–579, Jun. 1901.

[33] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Phys. Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[34] Y.-K. Zhang, J.-C. Liu, Y.-A. Cui, X.-H. Hei, and M.-H. Zhang, "An improved quantum genetic algorithm for test suite reduction," in *Proc. IEEE CSAE*, Shanghai, China, Jun. 2011, pp. 149–153.

[35] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 226–247, Mar. 2010.

[36] M. Harman and B. F. Jones, "Search-based software engineering," *Inf. Softw. Tech.*, vol. 43, no. 14, pp. 833–839, 2001.

[37] A. E. V. B. Coutinho, E. G. Cartaxo, and P. D. de Lima Machado, "Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing," *Softw. Qual. J.*, vol. 24, no. 2, pp. 407–445, 2016.

[38] D. Leon and A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in *Proc. IEEE ISSRE*, Nov. 2003, pp. 442–453.

[39] I. Dincă, "Multi-objective test suite optimization for event-B models," in *Informatics Engineering and Information Science*. Berlin, Germany: Springer, 2011, pp. 551–565.

[40] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, Nov. 1999.

[41] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Trans. Softw. Eng.*, vol. 36, no. 6, pp. 742–762, Nov. 2010.

[42] Z. Yi-kun, L. Ji-ceng, Y. Xue-min, C. Ying-an, and Z. Yi-kun, "Study of test suite reduction based on quantum evolutionary algorithm," in *Proc. AICI*, Sanya, China, 2010, pp. 483–487.

[43] B. A. Kitchenham *et al.*, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, Aug. 2002.

[44] J. Singer, "Using the American Psychological Association (APA) style guidelines to report experimental results," in *Proc. WESSM*, 1999, pp. 71–75.

[45] A. Jedlitschka and D. Pfahl, "Reporting guidelines for controlled experiments in software engineering," in *Proc. IEEE ISESE*, Nov. 2005, pp. 95–104.

[46] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Germany: Springer, 2012.

[47] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set size and block coverage on the fault detection effectiveness," in *Proc. IEEE ISSRE*, Monterey, CA, USA, Nov. 1994, pp. 230–238.

[48] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria," in *Proc. IEEE ICSE*, Los Alamitos, CA, USA, May 1994, pp. 191–200.

[49] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng.*, vol. 10, no. 4, pp. 405–435, 2005.

[50] V. R. Basili, *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. College Park, MD, USA: Univ. Maryland College Park, 1992.

[51] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. New York, NY, USA: CRC Press, 2003.

[52] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," in *Proc. IEEE ICSE*, Seattle, DC, USA, Apr. 1995, pp. 1–11.

[53] A. M. Smith and G. M. Kapfhammer, "An empirical study of incorporating cost into test suite reduction and prioritization," in *Proc. ACM SAC*, 2009, pp. 461–467.

[54] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of evolutionary testing," Simula Res. Lab., Oslo, Norway, Tech. Rep. 293, 2008, pp. 1–79.

[55] A. Arcuri, M. Z. Iqbal, and L. Briand, "Formal analysis of the effectiveness and predictability of random testing," in *Proc. ACM ISSTA*, Trento, Italy, 2010, pp. 219–230.

[56] C. Bird and T. Zimmermann, "Assessing the value of branches with what-if analysis," in *Proc. ACM FSE*, 2012, pp. 1–11.

[57] Z. Li, Y. Bian, R. Zhao, and J. Cheng, "A fine-grained parallel multi-objective test case prioritization on GPU," in *Proc. SBSE*, 2015, pp. 111–125.

[58] S. Yoo, M. Harman, and S. Ur, "GPGPU test suite minimisation: Search based software engineering performance improvement using graphics cards," *Empirical Softw. Eng.*, vol. 18, no. 3, pp. 550–593, 2013.

[59] H. Agrawal, "Dominators, super blocks, and program coverage," in *Proc. ACM SPPL*, 1994, pp. 25–34.

[60] P. J. Bernhard, "A reduced test suite for protocol conformance testing," *ACM Trans. Softw. Eng. Meth.*, vol. 3, no. 3, pp. 201–220, 1994.

[61] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," *Eur. J. Oper. Res.*, vol. 94, no. 2, pp. 392–404, Oct. 1996.

[62] K. Al-Sultan, M. Hussain, and J. Nizami, "A genetic algorithm for the set covering problem," *J. Oper. Res. Soc.*, vol. 47, no. 5, pp. 702–709, 1996.

[63] T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction," *Inf. Softw. Tech.*, vol. 40, nos. 5–6, pp. 347–354, 1998.

[64] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *Proc. IEEE ICSM*, Bethesda, MD, USA, Nov. 1998, pp. 34–43.

[65] A. Markus, J. Raik, and R. Ubar, "Test set minimization using bipartite graphs," in *Proc. IEEE BEC*, Oct. 1998, pp. 175–178.

[66] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," in *Proc. IEEE/ACM ICCAD*, San Jose, CA, USA, Nov. 1998, pp. 283–289.

[67] T. Y. Chen and M. F. Lau, "A simulation study on some heuristics for test suite reduction," *Inf. Softw. Tech.*, vol. 40, no. 13, pp. 777–787, 1998.

[68] T. Csoendes, S. Dibuz, and B. Kotnyek, "Test suite reduction in conformance testing," *Acta Cybern.*, vol. 14, no. 2, pp. 229–238, 1999.

[69] H. Agrawal, "Efficient coverage testing using global dominator graphs," *ACM SIGSOFT Softw. Eng. Notes*, vol. 24, no. 5, pp. 11–20, 1999.

[70] 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.

[71] J. Lee and C. Chung, "An optimal representative set selection method," *Inf. Softw. Tech.*, vol. 42, no. 1, pp. 17–25, 2000.

[72] P. J. Schroeder and B. Korel, "Black-box test reduction using input-output analysis," in *Proc. ACM SIGSOFT ISSTA*, 2000, pp. 173–177.

[73] B. Vaysburg, L. H. Tahat, and B. Korel, "Dependence analysis in reduction of requirement based test suites," *ACM SIGSOFT Softw. Eng. Notes*, vol. 27, no. 4, pp. 107–111, 2002.

[74] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 183–200, Feb. 2002.

[75] M. Harder, J. Mellen, and M. D. Ernst, "Improving test suites via operational abstraction," in *Proc. IEEE ICSE*, May 2003, pp. 60–71.

[76] M. Marre and A. Bertolino, "Using spanning sets for coverage testing," *IEEE Trans. Softw. Eng.*, vol. 29, no. 11, pp. 974–984, Nov. 2003.

[77] A. Cavalli, L. Lima, and N. Yevtushenko, "Test suite minimization for testing in context," *Softw. Test., Verification Rel.*, vol. 13, no. 3, pp. 141–155, 2003.

[78] J. A. Jones and M. J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Trans. Softw. Eng.*, vol. 29, no. 3, pp. 195–209, Mar. 2003.

[79] P. Saraph, M. Last, and A. Kandel, "Test case generation and reduction by automated input-output analysis," in *Proc. IEEE ICSMC*, Oct. 2003, pp. 768–773.

[80] S. Seol, M. Kim, S. T. Chanson, and S. Kang, "Interoperability test generation and minimization for communication protocols based on the multiple stimuli principle," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 10, pp. 2062–2074, Dec. 2004.

[81] M. P. E. Heimdahl and D. George, "Test-suite reduction for model based tests: Effects on test quality and implications for testing," in *Proc. IEEE ASE*, Sep. 2004, pp. 176–185.

[82] J. Black, E. Melachrinoudis, and D. Kaeli, "Bi-criteria models for all-uses test suite reduction," in *Proc. IEEE ICSE*, May 2004, pp. 106–115.

[83] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," in *Proc. ACM SIGSOFT Softw. Eng. Notes*, 2005, pp. 35–42.

[84] F. Belli and C. J. Budnik, "Minimal spanning set for coverage testing of interactive systems," in *Proc. ICTAC*, 2005, pp. 220–234.

[85] G.-V. Jourdan, H. Ural, and N. Zaguia, "Minimizing the number of inputs while applying adaptive test cases," *Inf. Process. Lett.*, vol. 94, no. 4, pp. 165–169, 2005.

[86] L. Pan, B. Zou, J. Li, and H. Chen, "Bi-objective model for test-suite reduction based on modified condition/decision coverage," in *Proc. PRDC*, Dec. 2005, pp. 1–7.

[87] D. Jeffrey and N. Gupta, "Test suite reduction with selective redundancy," in *Proc. IEEE ICSM*, Sep. 2005, pp. 549–558.

[88] Y. Lei and J. H. Andrews, "Minimization of randomized unit test cases," in *Proc. IEEE ISSRE*, Chicago, IL, USA, Nov. 2005, pp. 1–10.

[89] S. McMaster and A. M. Memon, "Call stack coverage for test suite reduction," in *Proc. IEEE ICSM*, Sep. 2005, pp. 539–548.

[90] D. Leon, W. Masri, and A. Podgurski, "An empirical evaluation of test case filtering techniques based on exercising complex information flows," in *Proc. ACM CSE*, 2005, pp. 412–421.

[91] M. Hennessy and J. F. Power, "An analysis of rule coverage as a criterion in generating minimal test suites for grammar-based software," in *Proc. IEEE/ACM ASE*, 2005, pp. 104–113.

[92] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. Souter, "Analyzing clusters of Web application user sessions," in *Proc. ACM SIGSOFT Softw. Eng. Notes*, 2005, pp. 1–7.

[93] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock, and A. Souter, "An empirical comparison of test suite reduction techniques for user-session-based testing of Web applications," in *Proc. IEEE ICSM*, Sep. 2005, pp. 587–596.

[94] M. A. Xue-ying, H. Zhen-feng, S. Bin-kui, and Y. Cheng-Qing, "A genetic algorithm for test-suite reduction," in *Proc. IEEE ICSMS*, Oct. 2005, pp. 133–139.

[95] R. Santelices, S. Sinha, and M. J. Harrold, "Subsumption of program entities for efficient coverage and monitoring," in *Proc. ACM SOQUA*, 2006, pp. 2–5.

[96] G. Misherghi and Z. Su, "HDD: Hierarchical delta debugging," in *Proc. ACM ICSE*, 2006, pp. 142–151.

[97] S. Sampath, S. Sprenkle, E. Gibson, and L. Pollock, "Web application testing with customized test requirements—An experimental comparison study," in *Proc. IEEE ISSRE*, Raleigh, NC, USA, Nov. 2006, pp. 266–278.

[98] B. Baudry, F. Fleurey, and Y. Le Traon, "Improving test suites for efficient fault localization," in *Proc. ACM ICSE*, 2006, pp. 82–91.

[99] G. Fraser and F. Wotawa, "Redundancy based test-suite reduction," in *Proc. FASE*, 2007, pp. 291–305.

[100] A. M. Smith, J. Geiger, G. M. Kapfhammer, and M. L. Soffa, "Test suite reduction and prioritization with call trees," in *Proc. IEEE/ACM ASE*, Atlanta, GA, USA, Nov. 2007, pp. 539–540.

[101] W. Masri, A. Podgurski, and D. Leon, "An emprical study of test case filtering techniques based on exercising information flows," *IEEE Trans. Softw. Eng.*, vol. 33, no. 7, pp. 454–477, Jul. 2007.

[102] A. Leitner, M. Oriol, A. Zeller, I. Ciupa, and B. Meyer, "Efficient unit test case minimization," in *Proc. IEEE/ACM ASE*, Nov. 2007, pp. 417–420.

[103] R. Zhang, J. Jiang, J. Yin, A. Jin, J. Lou, and Y. Wu, "A new method for test suite reduction," in *Proc. IEEE ICYCS*, Nov. 2008, pp. 1211–1216.

[104] H. Zhong, L. Zhang, and H. Mei, "An experimental study of four typical test suite reduction techniques," *Inf. Softw. Tech.*, vol. 50, no. 6, pp. 534–546, 2008.

[105] S. McMaster and A. Memon, "Call-stack coverage for GUI test suite reduction," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 99–115, Jan. 2008.

[106] X. Zhang, B. Xu, Z. Chen, C. Nie, and L. Li, "An empirical evaluation of test suite reduction for boolean specification-based testing," in *Proc. IEEE QSIC*, Oxford, U.K., Aug. 2008, pp. 270–275.

[107] W. Dong, "Test case reduction technique for BPEL-based testing," in *Proc. IEEE ISECS*, Aug. 2008, pp. 814–817.

[108] Y. Yu, J. A. Jones, and M. J. Harrold, "An empirical study of the effects of test-suite reduction on fault localization," in *Proc. ACM ICSE*, 2008, pp. 201–210.

[109] A. Calvagna, A. Gargantini, and E. Tramontana, " Building T-wise combinatorial interaction test suites by means of grid computing," in *Proc. IEEE WETICE*, Jun. 2009, pp. 213–218.

[110] J.-W. Lin and C.-Y. Huang, "Analysis of test suite reduction with enhanced tie-breaking techniques," *Inf. Softw. Tech.*, vol. 51, no. 4, pp. 679–690, 2009.

[111] G. K. Kaminski and P. Ammann, "Using logic criterion feasibility to reduce test set size while guaranteeing fault detection," in *Proc. IEEE ICST*, Denver, CO, USA, Apr. 2009, pp. 356–365.

[112] S. Parsa, A. Khalilian, and Y. Fazlalizadeh, "A new algorithm to test suite reduction based on cluster analysis," in *Proc. IEEE ICCSIT*, Aug. 2009, pp. 189–193.

[113] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proc. ACM ISSTA*, 2009, pp. 57–68.

[114] U. Farooq and C. P. Lam, "Evolving the quality of a model based test suite," in *Proc. IEEE ICSTW*, Apr. 2009, pp. 141–149.

[115] W. Ding, J. Kou, K. Li, and Z. Yang, "An optimization method of test suite in regression test model," in *Proc. IEEE WCSE*, Xiamen, China, May 2009, pp. 180–183.

[116] W. Masri and M. El-Ghali, "Test case filtering and prioritization based on coverage of combinations of program elements," in *Proc. ACM IWDA*, Chicago, IL, USA, 2009, pp. 29–34.

[117] D. J. Mala, M. Kamalapriya, R. Shobana, and V. Mohan, "A non-pheromone based intelligent swarm optimization technique in software test suite optimization," in *Proc. IAMA*, 2009, pp. 1–5.

[118] P. Ng, R. Y. Fung, and R. W. Kong, "Incremental model-based test suite reduction with formal concept analysis," *J. Inf. Process. Syst.*, vol. 6, no. 2, pp. 197–208, 2010.

[119] S. Nachiyappan, A. Vimaladevi, and C. B. S. Lakshmi, "An evolutionary algorithm for regression test suite reduction," in *Proc. INCOCCI*, 2010, pp. 503–508.

[120] N. Pan, F. Zeng, and Y.-H. Huang, "Test case reduction based on program invariant and genetic algorithm," in *Proc. IEEE WiCOM*, Sep. 2010, pp. 1–5.

[121] S. Jia-Ze and W. Shu-Yan, "A novel chaos discrete particle swarm optimization algorithm for test suite reduction," in *Proc. ICISE*, 2010, pp. 1–4.

[122] L. Zhao and W. Luo, "An algorithm for reducing test suite based on interface parameters," in *Proc. IEEE CiSE*, Dec. 2010, pp. 1–4.

[123] S. Parsa and A. Khalilian, "On the optimization approach towards test suite minimization," *Int. J. Softw. Eng. Appl.*, vol. 4, no. 1, pp. 15–28, 2010.

[124] Y. Ma, Z. Zhao, Y. Liang, and M. Yun, "A usable selection range standard based on test suite reduction algorithms," *Wuhan Uni. J. Nat. Sci.*, vol. 15, no. 3, pp. 261–266, 2010.

[125] Q. Gu, B. Tang, and D. Chen, "Optimal regression testing based on selective coverage of test requirements," in *Proc. IEEE ISPA*, Sep. 2010, pp. 419–426.

[126] S. Selvakumar, M. Dinesh, C. Dhineshkumar, and N. Ramaraj, "Test suite diminuition using GRE heuristic with selective sedundancy approach," in *Proc. Rec. Tre. Net. Commun.*, 2010, pp. 563–571.

[127] D. Hao, T. Xie, L. Zhang, X. Wang, J. Sun, and H. Mei, "Test input reduction for result inspection to facilitate fault localization," *Autom. Softw. Eng.*, vol. 17, no. 1, pp. 5–31, 2010.

[128] D. Kichigin, "A method for test suite reduction for regression testing of interactions between software modules," in *Perspectives of Systems Informatics*. Berlin, Germany: Springer, 2010, pp. 177–184.

[129] S. Yoo and M. Harman, "Using hybrid algorithm for pareto efficient multi-objective test suite minimisation," *J. Syst. Softw.*, vol. 83, no. 4, pp. 689–701, 2010.

[130] N. Koochakzadeh and V. Garousi, "A tester-assisted methodology for test redundancy detection," *J. Adv. Softw. Eng.-Special Issue Softw. Test Autom.*, vol. 2010, Jan. 2010, p. 6.

[131] S. Yoo, "A novel mask-coding representation for set cover problems with applications in test suite minimisation," in *Proc. IEEE SSBSE*, Sep. 2010, pp. 19–28.

[132] X. Chen *et al.*, "A test suite reduction approach based on pairwise interaction of requirements," in *Proc. ACM SAC*, 2011, pp. 1390–1397.

[133] D. Drmanac and M. Laisne, "Wafer probe test cost reduction of an RF/A device by automatic testset minimization—A case study," in *Proc. IEEE ITC*, Anaheim, CA, USA, Sep. 2011, pp. 1–10.

[134] A. Srikanth, N. J. Kulkarni, K. V. Naveen, P. Singh, and P. R. Srivastava, "Test case optimization using artificial bee colony algorithm," in *Advances in Computing and Communications*. New York, NY, USA: Springer-Verlag, 2011, pp. 570–579.

[135] E. Bauer, J. M. Küster, and G. Engels, "Test suite quality for model transformation chains," in *Objects, Models, Components, Patterns*. Berlin, Germany: Springer, 2011, pp. 3–19.

[136] H. Cichos and T. S. Heinze, "Efficient test suite reduction by merging pairs of suitable test cases," in *Models in Software Engineering*. Berlin, Germany: Springer, 2011, pp. 244–258.

[137] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An empirical study of JUnit test-suite reduction," in *Proc. IEEE ISSRE*, Dec. 2011, pp. 170–179.

[138] Y. Liu, K. Wang, W. Wei, B. Zhang, and H. Zhong, "User-session-based test cases optimization method based on agglutinate hierarchy clustering," in *Proc. IEEE iThings/CPSCom*, Dalian, China, Oct. 2011, pp. 413–418.

[139] L. Guo, J. Yi, L. Zhang, X. Wang, and D. Tong, "CGA: combining cluster analysis with genetic algorithm for regression suite reduction of microprocessors," in *Proc. IEEE SOCC*, Sep. 2011, pp. 207–212.

[140] M. P. Usaola, P. R. Mateo, and B. P. Lamancha, "Reduction of test suites using mutation," in *Fundamental Approaches to Software Engineering*. Berlin, Germany: Springer, 2012, pp. 425–438.

[141] J. Regehr, Y. Chen, P. Cuoq, E. Eide, C. Ellison, and X. Yang, "Test-case reduction for C compiler bugs," *ACM SIGPLAN Notices*, vol. 47, no. 6, pp. 335–346, 2012.

[142] G. M. Kapfhammer, "Towards a method for reducing the test suites of database applications," in *Proc. IEEE ICST*, Montreal, QC, Canada, Apr. 2012, pp. 964–965.

[143] S. Xu, H. Miao, and H. Gao, "Test suite reduction using weighted set covering techniques," in *Proc. IEEE SNPD*, Aug. 2012, pp. 307–312.

[144] A. Khalilian and S. Parsa, "Bi-criteria test suite reduction by cluster analysis of execution profiles," in *Advances in Software Engineering Techniques*. Berlin, Germany: Springer, 2012, pp. 243–256.

[145] M. Bozkurt, "Cost-aware pareto optimal test suite minimisation for service-centric systems," in *Proc. ACM AGECCO*, New York, NY, USA, 2012, pp. 1429–1436.

[146] A. Marchetto, M. Islam, A. Susi, and G. Scanniello, "A multi-objective technique for test suite reduction," in *Proc. ICSEA*, Venice, Italy, 2013, pp. 18–24.

[147] Q. Mayo, R. Michaels, and R. Bryce, "Test suite reduction by combinatorial-based coverage of event sequences," in *Proc. IEEE ICSTW*, Cleveland, OH, USA, Apr. 2014, pp. 128–132.

[148] S. Arlt, A. Podelski, and M. Wehrle, "Reducing gui test suites via program slicing," in *Proc. ACM ISSTA*, 2014, pp. 270–281.

[149] S. Wang, S. Ali, and A. Gotlieb, "Cost-effective test suite minimization in product lines using search techniques," *J. Syst. Softw.*, vol. 103, pp. 370–391, May 2014.

[150] P. Bokil, P. Krishnan, and R. Venkatesh, "Achieving effective test suites for reactive systems using specification mining and test suite reduction techniques," *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 1, pp. 1–8, 2015.

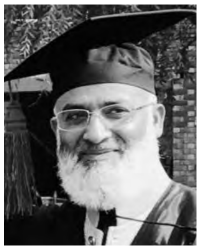
[151] J. Geng, Z. Li, R. Zhao, and J. Guo, "Search based test suite minimization for fault detection and localization: A co-driven method," in *Proc. ISSBSE*, 2016, pp. 34–48.

SAIF UR REHMAN KHAN received the M.S. degree in computer science from Mohammad Ali Jinnah University, Islamabad, Pakistan, in 2007. He is currently pursuing the Ph.D. degree in computer science with the University of Malaya, Kuala Lumpur, Malaysia.

From 2005 to 2011, he was a Lecturer with the COMSATS Institute of Information Technology, Islamabad, Pakistan. His research interests in software engineering include optimal software testing, model-based testing, search-based software testing, and empirical software engineering. He was a recipient of the Best Paper Presentation Award from the Faculty of Computer Science and Information Technology, UM, in 2014, and the Paper Reviewing Award (Future Generation Computer Systems) in 2016.

SAI PECK LEE (M'10) received the Ph.D. degree in computer science from Université Paris I Panthéon-Sorbonne, France. She is currently a Professor with the Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia.

Her current research interests include object-oriented techniques and CASE tools, software reuse, requirements engineering, and software quality. She is a Founding Member of the Informing Science Institute. She has been in several expert's review panels, both locally and internationally.



NADEEM JAVAID (S'08–M'11–SM'16) received the bachelor's degree in computer science from Gomal University, Dera Ismail Khan, Pakistan, in 1995, the master's degree in electronics from Quid-I-Azam University, Islamabad, Pakistan, in 1999, and the Ph.D. degree in computer science from the University of Paris-Est, France, in 2010. He is currently an Associate Professor and the founding Director of the ComSens (Communications over Sensors) Research Lab, Department of Computer Science, COMSATS Institute of Information Technology, Islamabad. He has supervised seven Ph.D. and 75 masters' theses. He has authored over 500 papers in technical journals and international conferences. His research interests include energy optimization in smart/micro grids, cloud computing for smart grids, IoT-enabled wireless sensor networks, and big data analytics in smart grids. He received the Best University Teacher Award for 2016 from Higher Education Commission of Pakistan and the Research Productivity Award 2017 from Pakistan Council for Science and Technology. He is an Associate Editor of the IEEE ACCESS and an Editor of the *International Journal of Space Based and Situated Computing*.

WADOOD ABDUL received the Ph.D. degree in signal and image processing from the University of Poitiers, France, in 2011. He is currently an Assistant Professor with the Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Saudi Arabia.

His research interests are focused on color image watermarking, multimedia security, steganography, fingerprinting, and biometric template protection.