# *A Survey on Test Suite Reduction Frameworks and Tools*

*Saif Ur Rehman Khan*[1], *Sai Peck Lee*[1], *Raja Wasim Ahmad*[2], *Adnan Akhunzada*[2] *and Victor Chang*[3]

[1]*Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia.*
*saif_rehman@siswa.um.edu.my, saipeck@um.edu.my*
[2]*Department of Computer Science, COMSATS Insititute of Information Technology (CIIT), Pakistan.*
*wasimraja@ciit.net.pk, a.adnan@siswa.um.edu.my*
[3]*International Business School Suzhou, Xi'an Jiaotong Liverpool University, Suzhou, China*.
*ic.victor.chang@gmail.com*

**Abstract-**Software testing is a widely accepted practice that ensures the quality of a System under Test (SUT). However, the gradual increase of the test suite size demands high portion of testing budget and time. Test Suite Reduction (TSR) is considered a potential approach to deal with the test suite size problem. Moreover, a complete automation support is highly recommended for software testing to adequately meet the challenges of a resource constrained testing environment. Several TSR frameworks and tools have been proposed to efficiently address the test-suite size problem. The main objective of the paper is to comprehensively review the state-of-the-art TSR frameworks to highlights their strengths and weaknesses. Furthermore, the paper focuses on devising a detailed thematic taxonomy to classify existing literature that helps in understanding the underlying issues and proof of concept. Moreover, the paper investigates critical aspects and related features of TSR frameworks and tools based on a set of defined parameters. We also rigorously elaborated various testing domains and approaches followed by the extant TSR frameworks. The results reveal that majority of TSR frameworks focused on randomized unit testing, and a considerable number of frameworks lacks in supporting multi-objective optimization problems. Moreover, there is no generalized framework, effective for testing applications developed in any programming domain. Conversely, Integer Linear Programming (ILP) based TSR frameworks provide an optimal solution for multi-objective optimization problems and improve execution time by running multiple ILP in parallel. The study concludes with new insights and provides an unbiased view of the state-of-the-art TSR frameworks. Finally, we present potential research issues for further investigation to anticipate efficient TSR frameworks.

**Keywords:** *Regression testing; test suite optimization; test suite reduction; frameworks; fault localization*.

## 1. Introduction

Software testing ensures the quality and reliability of a System Under Test (SUT) by revealing maximum possible defects (Myers et al., 2011). However, software testing is the most expensive quality assurance practice since it consumes up to 50% of the total software development cost (Ramler and Wolfmaier, 2006). Although, exhaustive testing (e.g., running all possible paths in the SUT (Lee and Chung, 2000)) is putative to provide high confidence to development organizations regarding the SUT quality (Kuhn and Okun, 2006). However, exhaustive testing is impractical due to time and budget constraints (Tassey, 2002). Moreover, execution of the entire test suite is also impractical, if high human interventions are required for testing a software system (Haug et al., 2001). In the literature, researchers have reported different testing scenarios (Rothermel et al., 2001, Lin et al., 2012), which concludes that exhaustive testing requires a considerable amount of testing budget. Rothermel et al. (Rothermel et al., 2001) reported that testing a product containing 20,000 lines of code requires seven weeks and several hundred thousand dollars to execute the entire test suite. Moreover, Lin et al. (Lin et al., 2012) performed an empirical analysis of regression testing using 57,758 functions and 2,320 test cases. They reported that the running time of a single test case ranges from 10 minutes to 100 minutes, which is based on the configuration sequence and required test activities. Finally, they estimated that to execute all 2,320 test cases would require 36 test-bed days.

To deal with the aforementioned exhaustive testing problems, development organizations are attracted to adopt optimal testing strategies (Nachmanson et al., 2004). Analytics is useful in software design and testing (Chang, 2015b).

Similarly, when we design a software, do not forget diverse security concerns (Akhunzada et al., 2015d, Akhunzada et al., 2015a) to ensure that the software design supports penetration testing (ethical hacking) against hacking (Chang and Ramachandran, 2016).

In the literature, three main techniques have been discussed to support regression testing (Yoo and Harman, 2012): (i) *test suite reduction*: to find a reduced suite by permanently eliminating redundant test cases according to certain criteria, (ii) *test case selection*: to select such previously generated test cases that cover the modified portion of the software, and (iii) *test case prioritization*: to determine the ordering of test cases based on a particular objective such as to increase the rate of Fault Detection Effectiveness (FDE). In our context, the real challenge is to determine a subset of non-redundant test cases, which finds a maximum number of possible defects similar to the original test suite (Khan et al., 2006). A tester can meet this challenge by randomly selecting the generated test cases (Chen et al., 2010), but such random selection may end up as exclusion of essential test cases (Hao et al., 2012). Consequently, it has a negative impact on the fault detection capability of the reduced suite. A practical approach to solve the test-suite size problem is to find a minimal subset of test cases automatically, while keeping their fault detection capability similar to the original test suite. Test Suite Reduction (TSR) approach focuses on finding a minimal test suite by permanently discarding the redundant test cases from the original test suite (Dandan et al., 2013). The optimal TSR problem is formally defined by Harrold et al. (Harrold et al., 1993) as stated below:

> **Given:** *A test suite, TS, and a set of test requirements $R_1$, $R_2$,…, $R_n$, that must be satisfied to provide the desired test coverage of the program, and subsets of TS, $T_1$, $T_2$,…, $T_m$, where each $T_i$ is associated with each of the $R_i$s such that any one of the test cases $tc_j$ of $T_i$ can be used to test requirement $R_i$.*
>
> **Problem:** *Find a Reduced Suite (RS) containing minimal test cases from TS that satisfies all test requirements $R_i$ at least once.*

The optimal TSR problem is known to be NP-complete problem and is equivalent to set cover problem (Michael and David, 1979). Researchers have recommended complete automation support for various activities of test suite development cycle, such as test generation, execution, and evaluation, to minimize high cost of regression testing. Bertolino (Bertolino, 2007) has recommended 100% automated testing, which facilitates the tester to meet the challenges of a resource constrained testing environment. Motivated by this, researchers have proposed various frameworks that results various tools to provide automation facility during TSR process (Horgan and London, 1992, Xie et al., 2004, Xie et al., 2006, Andrews et al., 2006, Pacheco and Ernst, 2007, Jaygarl et al., 2010, Zhang et al., 2009, Dadeau et al., 2007, Wang et al., 2015, Kauffman and Kapfhammer, 2012, Sampath et al., 2007, Sampath et al., 2011, Woo et al., 2007, Chae et al., 2011, Hsu and Orso, 2009, Li et al., 2013, Zhang et al., 2010, Burger and Zeller, 2011, Campos et al., 2012). Consequently, automated TSR helps in accelerating the software delivery process compared to manual test filtering (Hsu and Orso, 2009). Automation has been achieved for Cloud storage for big data, which provide a supporting use case (Chang and Wills, 2016). Cloud service APIs Plays a vital role to integrate enterprise applications as they offer a set of protocols, which helps in connecting applications to various cloud services. APIs are useful for different disciplines such as business intelligence (Chang, 2014a) and social networks (Chang, 2015a, Chang, 2014b).

Prior works (Yoo and Harman, 2012, Elberzhager et al., 2012) lack in considering and analyzing TSR frameworks that is necessary to understand the body of knowledge in the area of TSR. To the best of our knowledge, this is the first effort that studies the TSR frameworks and tools comprehensively. The main objective of this survey is to provide an up-to-date view and in-depth analysis of state-of-the-art that is necessary to understand the body of knowledge. The survey analyzes, synthesizes, and categorizes current state-of-the-art TSR frameworks and tools. Furthermore, the survey focuses on identifying future research opportunities in this field of study. The main contributions of the paper are listed below:

- An extensive review on automated support for TSR.
- Presenting a thematic taxonomy to categorize the existing literature based on various testing domains, approaches, and their corresponding parameters.
- Providing a detailed comparative analysis of TSR frameworks based on the devised taxonometric parameters.

- Highlighting the strengths and limitations of the TSR tools and frameworks related to a particular testing domain.
- Synthesizing current state-of-the-art based on the underlying common philosophies.
- Highlighting several potential research issues in this field of study.

The rest of the paper is organized as follows: Section 2 presents a thematic taxonomy of TSR frameworks. Section 3 discusses current state-of-the-art TSR frameworks/tools based on various testing domains and approaches. Furthermore, it discusses strengths and weaknesses of existing TSR frameworks. Section 4 provides a comparison of current TSR frameworks based on the devised thematic taxonomy. Section 5 discusses potential research issues followed by concluding remarks in Section 6.

## 2. Taxonomy of Test Suite Reduction (TSR) Frameworks/Tools

This section presents a taxonomy for the thematic classification of TSR frameworks and tools based on defined parameters as depicted clearly in Figure 1. The defined parameters include: (i) approach type, (ii) testing paradigm, (iii) optimization type, (iv) coverage source, (v) execution platform, (vi) computational mode, (vii) license type, (viii) evaluation, (ix) customizability, and (x) support.

The first identified parameter is the *approach type*, which represents the principal category of TSR approaches focused by a TSR framework. There are four main attributes for approach type: (i) coverage-based, (ii) search-based, (iii) Integer Linear Programming (ILP) based, and (iv) similarity-based. The coverage-based TSR approaches greedily select such test cases that cover maximum portions (e.g., statements or branches) of a program under test. In contrast, search-based approaches employ various search algorithms, such as Genetic Algorithm (Deb et al., 2002), to find diverse test cases from the initial population. Conversely, ILP-based approaches determine minimal global solution for TSR problem based on the defined objectives and constraints (Black et al., 2004). Furthermore, ILP-based approaches achieve Pareto-optimal solution for multi-objective TSR problem (Baller et al., 2014). Notice that Pareto-optimal solution is a *non-dominated* solution, which cannot be further improved (Yoo and Harman, 2007). In contrast, similarity-based approaches focus on finding most different test cases based on the computed similarity degree between test cases (Coutinho et al., 2013). Note that similarity-based approaches heavily depend on a similarity matrix that shows the degree of similarity for all pairs of test cases. The similarity degree is calculated using a similarity measure such as Jaccard index (Jaccard, 1901).

The parameter *testing paradigm* represents the type of implementation language of the targeted SUT. The supported attributes are structured, object-oriented, or aspect-oriented. The other important parameter is the *optimization type*, which is regarded as a number of optimization objectives considered by a TSR framework. The optimization type is categorized into (i) single-objective: focuses either on finding a RS or computing the FDE, and (ii) multi-objective: focuses on both RS and FDE. In comparison to single-objective optimization, multi-objective supported tools need to find a best tradeoff between the targeted objectives and constraints (Wang et al., 2015).

The parameter *coverage source* is employed by the coverage supported frameworks to find a reduced test suite. The coverage source uses two main attributes: (i) source code and (ii) test execution profile. The frameworks that use the source code as a coverage source, determine the reduced suite by picking such test cases that cover maximum elements of the program source code. On the other hand, in the test execution profile-based category, first execution profiles are captured by running the entire test suite (Leon and Podgurski, 2003). Next, the supported framework determines the representative test cases based on the achieved coverage score of execution profiles.

The parameter *execution platform* represents the number of servers used by a test reduction framework to determine the RS. It can be broadly classified into: (i) single server and (ii) multiple servers. In the case of multiple servers, the TSR problem is solved by using divide-and-conquer strategy. Initially, the problem (large test suite) is divided into many sub-problems (several small test suites). Next, each sub-problem is executed on a single server with the aim to speed up the TSR process. The *computational mode* is the type of processing supported by a framework during TSR process and classified into two attributes: (i) online and (ii) offline. The online attribute accepts several smaller test suites in a sequential manner to determine the RS. In contrast, the offline attribute accepts the whole test suite to find an optimal solution.

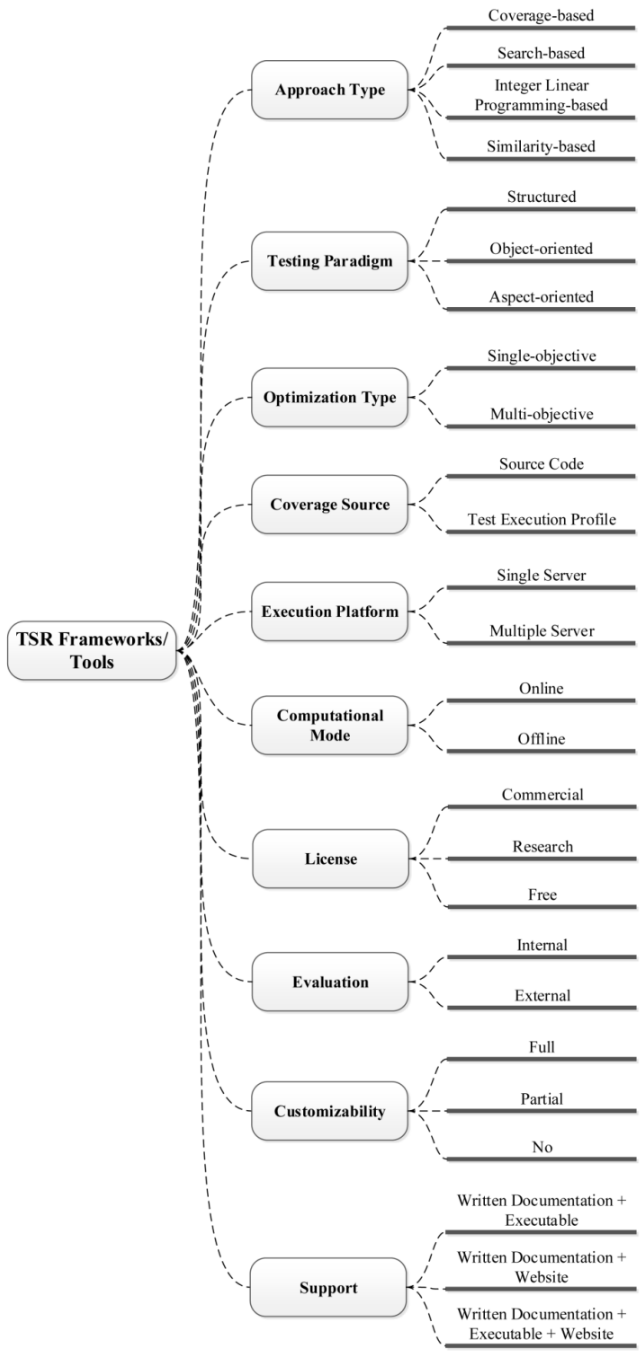

Figure 1: Taxonomy of the Current State-of-the-Art Test Suite Reduction (TSR) Frameworks/Tools

The parameter *license type* is considered as a primary characteristic for the selection of TSR tools. This parameter defines three main categories including commercial, academic research, and free tools. License of Commercial tools can be acquired through payment. Academic research tools include prototypes and tools developed by research labs/ groups, while Free tools are available free of cost. These tools are mostly open source having common licenses such as GPL, ASL, EPL and CeCILL.

*Evaluation* parameter refers to external or internal evaluation of the selected frameworks. Internal evaluation disscusses those framework which have been evaluated by builders/design teams. While external evaluation refers to those framework which have been tested outside the environment where they have been originally built. Such external evaluation enhances the degree of confidence regarding usefulness of certain framework.

*Customizability* can be regarded as an ability of the tool to support desired alterations. This parameter defines three categories including Full, partial and No support for customizability. Fully customizable tools support major alterations (e.g., customizing major basic functions) while partial customizability allows only minor alterations (e.g., seamless integration into a given environment). For example, most of the open source tools support full customizability. On the other hand, most of the proprietary and commercial tools provide partial or no customizability.

Another significant characteristic of any tool is the *support* availability. Information about executables and written documentation is indispensable for a wider analysis of TSR tools. In our case, very few studies provided download links for executables and source code. So we have performed rigorous search to find the web links where tool support (executables, source code, written documentation) was actually available. We have approached personal web sites of authors and research groups in this regard. We have mainly divided available support into executables, written documentation (manuals, tutorials, instructions, examples etc.) and tool web site.

## 3. State-of-the-Art Test Suite Reduction (TSR) Frameworks

This section discusses current state-of-the-art TSR frameworks and also highlights their strengths and weaknesses. The effectiveness of TSR process heavily depends on the employed frameworks to support the test reduction process. Figure 2 presents the timeline diagram of proposed TSR frameworks covering time period between year 1992 to year 2015.

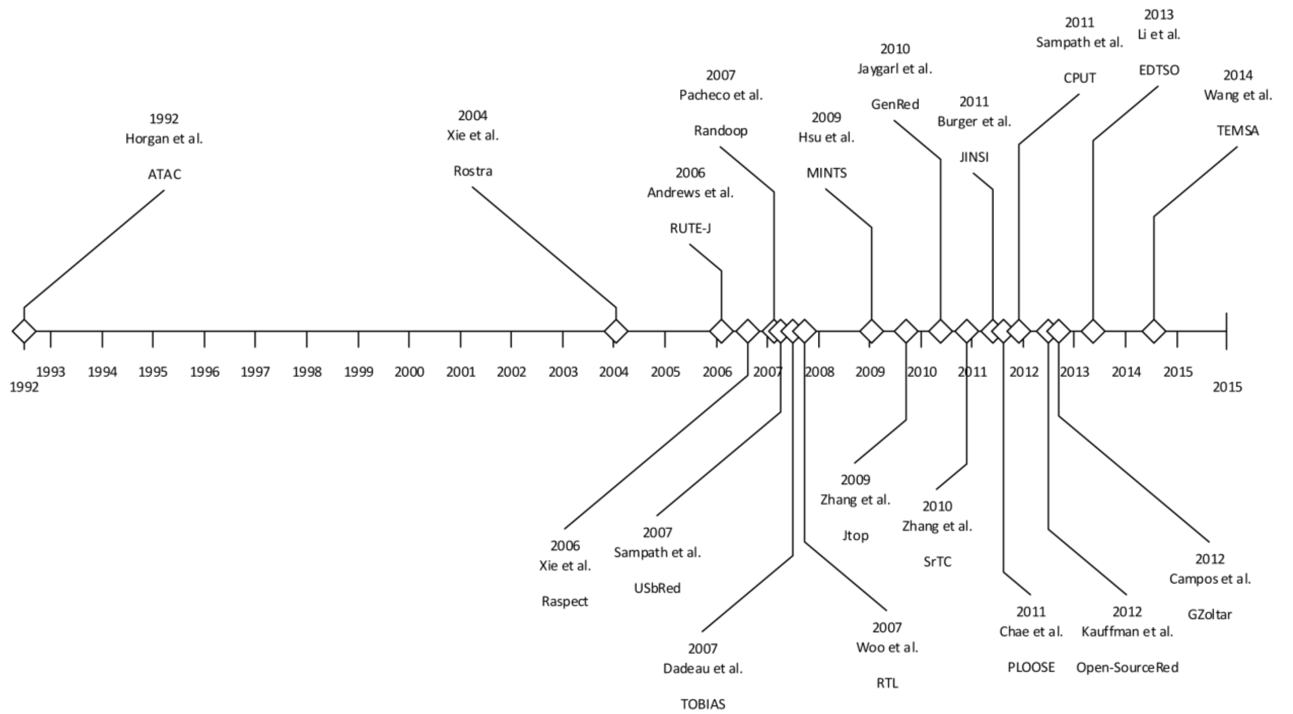

Figure 2: Timeline Diagram of TSR Tools/Frameworks

Researchers have proposed a number of frameworks that support different programming languages, testing paradigms, and testing approaches. Existing TSR frameworks are categorized into five main classes based on the targeted testing domains and approaches: (i) randomized unit testing, (ii) user session testing, (iii) retargeted compilers testing, (iv) integer linear programming, and (v) automated fault-localization.

### 3.1. Randomized Unit Testing

Unit testing is one of the prominent software testing method that focuses on testing the smallest testable part of the SUT such as source code method, a group of methods, or classes along with related control data and operating procedures (Myers et al., 2011). Randomized unit testing refers to such testing mechanism, where some random elements are involved in selecting the parameters and/or methods (Beizer, 2002). Therefore, it generates distinct test inputs easily and quickly from common data structures, which can be useful in exposing defects (Andrews et al., 2011). The following sub-sections debate on the existing randomized unit testing based TSR tools with special emphasis on their strengths and shortcomings.

#### 3.1.1. Randomized Unit Testing based TSR Tools

Majority of existing TSR tools focused on randomized unit testing, including *ATAC* (Horgan and London, 1992), *Rostra* (Xie et al., 2004), *Raspect* (Xie et al., 2006), *RUTE-J* (Andrews et al., 2006), *Randoop* (Pacheco and Ernst, 2007), *GenRed* (Jaygarl et al., 2010), *Jtop* (Zhang et al., 2009), *TOBIAS* (Dadeau et al., 2007), TEMSA (Wang et al., 2015) and *Open-SourceRed* (Kauffman and Kapfhammer, 2012). In the following, we briefly discuss current randomized unit testing focused TSR tools.

- *ATAC*: Horgan and London (Horgan and London, 1992) proposed a unit testing based TSR tool called *ATAC* that eliminates redundant test cases based on the high coverage of selected data-flow coverage metric. The main advantage of *ATAC* is to support the test evaluation process by implicitly facilitating in obtaining execution slices of program paths. However, *ATAC* lacks in detecting a significant number of crucial defects, since it only supports selective testing.
- *Rostra*: Xie et al. (Xie et al., 2004) presented a formal object-oriented unit testing based framework, called *Rostra* that facilitates to evaluate a test suite's quality by finding similar unit test cases based on the equivalent objects. Although, it is argued that *Rostra* efficiently generates test sequences such as violating and satisfying with few ineffective test sequences. However, *Rostra* uses state-space exploration, which is heuristically pruned. Ultimately, the proposed tool never guarantees to determine minimal solution.
- *Raspect*: Xie et al. (Xie et al., 2006) extended *Rostra* (Xie et al., 2004) to detect the redundant unit test cases for aspect-oriented programs, called *Raspect* that preserves structural coverage using similar states for the considered objects. The proposed tool automatically detects similar test cases, which do not exercise the new behavior of the program under test. Consequently, it requires less time for manual inspection of the computed solution. However, *Raspect* performance degrades significantly in the case of non-availability of program's algebraic specification, which is used for asserting the program behavior.
- *RUTE-J*: Andrews et al. (Andrews et al., 2006) proposed a tool, called *RUTE-J* to test a small portion of the program under test such as methods or classes. It uses delta debugging technique (Burger and Zeller, 2011) to isolate the failure-inducing inputs of the program under test. Furthermore, it effectively manages test's storage and retrieval operations. However, *RUTE-J* tool interaction experience is highly required as the tester has to enter some technical inputs.
- *Randoop*: Pacheco and Ernst (Pacheco and Ernst, 2007) presented a deterministic execution feedback-directed tool, called *Randoop* that determines and eliminates the test inputs that either throw exceptions or create similar objects. However, *Randoop* lacks in creating a parameter graph instead it used previously generated test sequences of a component set, which are useful to create similar objects or throw exceptions. Consequently, *Randoop* determines a small number of redundant test cases.
- *GenRed*: Jayagarl et al. (Jaygarl et al., 2010) proposed a new feedback-directed randomized tool, called *GenRed* that discards generated redundant method sequences. The authors empirically evaluated the performance of *GenRed* and *Randoop* (Pacheco and Ernst, 2007) using a total testing time criterion. Although, *GenRed* showed good performance compared to *Randoop* in terms of size of the RS and achieved code coverage, however, the authors did not report required time mechanism necessary to understand the core mechanism of controlling *GenRed* working.

- *Jtop*: Zhang et al. (Zhang et al., 2009) focused on managing JUnit test cases and proposed an Eclipse IDE plug-in, called *Jtop*, which does not require any code coverage information for TSR. The authors define and substitute '*relevant relation*' notion with traditional coverage-based information for TSR. Consequently, it enables *Jtop* to determine a set of related test cases covering particular elements of the program under test. Although, authors evaluated the performance of *Jtop* using a thorough set of test data, but they did not discuss about the achieved results.
- *TOBIAS*: Dedeau et al. (Dadeau et al., 2007) presented a semi-automated combinatorial testing based tool, called TOBIAS. The proposed tool captures the test engineers' knowledge to write test patterns, which requires significant manual effort. After that, test patterns are unfolded that might results into a set of millions of abstract and redundant test cases. Therefore, to better cope with the impact of combinatorial unfolding, TOBIAS employs a generic filtering and selection mechanisms.
- *TEMSA*: Wang et al. (Wang et al., 2015) presented a tool called TEst Minimization using Search Algorithms (TEMSA). The proposed tool supports efficient software product line testing using feature pairwise coverage criterion. The test engineer needs to select the test minimization objective with highest importance. Then, TEMSA recommends most suitable weight-based search algorithm based on a given importance value of the test minimization objective. After that, test engineer might start running the test minimization process. Finally, TEMSA generates the minimized test suite.
- *Open-SourceRed*: Kauffman and Kapfhammer (Kauffman and Kapfhammer, 2012) proposed an open-source framework supported by a prototype tool, called *Open-SourceRed*, which contains two open-source components: (i) Proteja and (ii) Modificare. *Proteja* accepts two main inputs including, Java Program and JUnit test suite, and generates a test-coverage report in a binary matrix format based on statement, method, and class coverage criteria. In contrast, *Modificare* takes test-coverage and timing information to perform TSR and also produces a modified suite file.

### 3.1.2. Comparison of Randomized Unit Testing based TSR Tools

A significant number of existing TSR tools (53%, 10/19) focused on randomized unit testing. However, there are two key limitations associated with current tools in the category of randomized unit testing that need further research to design more effective tools. The limitations are: (i) test oracle problem, which demands manual effort to evaluate the test results, and (ii) generating infeasible test cases, since execution does not represent the actual scenario. Currently, tools are focusing on different programming domains, including structured (Horgan and London, 1992), object-oriented (Zhang et al., 2009, Xie et al., 2006, Andrews et al., 2006, Jaygarl et al., 2010, Kauffman and Kapfhammer, 2012, Dadeau et al., 2007), and aspect-oriented (Xie et al., 2004). However, there is no generalized tool, which is applicable and effective to test programs developed using various programming domains.

A variety of tools for randomized unit testing has been proposed, where the majority was based on code coverage information of the program under test to determine the reduced suite (Horgan and London, 1992, Xie et al., 2004, Xie et al., 2006, Andrews et al., 2006, Pacheco and Ernst, 2007, Jaygarl et al., 2010, Dadeau et al., 2007). However, there are a number of drawbacks of using code coverage information: (i) instrumented version of source code needs to be run to collect coverage information, (ii) coverage storage and management cost proportionally increases with respect to the program size, and (iii) previously collected coverage information becomes inconsistent due to software evolution. Consequently, current coverage-based TSR tools might be difficult to test real-world applications due to extra cost in terms of coverage collection, storage, and management. In this situation, compared to *ATAC* (Horgan and London, 1992), *Rostra* (Xie et al., 2004), *Raspect* (Xie et al., 2006), *RUTE-J* (Andrews et al., 2006), *Randoop* (Pacheco and Ernst, 2007), *GenRed* (Jaygarl et al., 2010), *TOBIAS* (Dadeau et al., 2007), *TEMSA* (Wang et al., 2015) and *Open-SourceRed* (Kauffman and Kapfhammer, 2012), *Jtop* (Zhang et al., 2009) is an effective tool, since it does not require any code coverage information. Furthermore, *Jtop* automatically extracts static call graph to determine a test case covering particular program elements; however, it usually contains over approximations. In other words, over approximation results significant call relationships, which would never occur in real program execution.

In contrast to other randomized unit testing tools, both *Rostra* (Xie et al., 2004) and *Raspect* (Xie et al., 2006) dynamically monitor the test executions to detect and remove the redundant test cases based on equivalent object's states. To achieve this, they used bounded exhaustive generation with state-space exploration. Consequently, they required less time for test synthesis as redundant test cases are automatically detected. However, both *Rostra* (Xie et al., 2004) and *Raspect* (Xie et al., 2006) requires significant resources in terms of memory to keep track of all object's states.

Among all, two TSR tools including, *Randoop* (Pacheco and Ernst, 2007) and *GenRed* (Jaygarl et al., 2010), are based on feedback-directed mechanism, which enables a limited access to the objects. For instance, if *Randoop* (Pacheco and Ernst, 2007) is unable to locate an object of the correct type required to invoke a method in the existing set of sequences, then it will never be able to invoke such method. Moreover, *Randoop* (Pacheco and Ernst, 2007) is easy to use but lacks in supporting multi-threaded programs. Ultimately, *Randoop* (Pacheco and Ernst, 2007) is unable to concurrently execute the methods. Thus, in this situation, *GenRed* (Jaygarl et al., 2010) outperformed *Randoop* (Pacheco and Ernst, 2007) by using on-demand generation to create necessary test inputs. Furthermore, *GenRed* (Jaygarl et al., 2010) assigns high priority to the methods with lower coverage to invoke method in the current set of method sequences.

Some randomized unit testing supported TSR tools including, *ATAC* (Horgan and London, 1992), *RUTE-J* (Andrews et al., 2006), *Randoop* (Pacheco and Ernst, 2007), *GenRed* (Jaygarl et al., 2010), *TOBIAS* (Dadeau et al., 2007), and *Open-SourceRed* (Kauffman and Kapfhammer, 2012), require high human interventions during TSR process for their correct functioning, which ultimately needs additional testing time. For instance, *ATAC* (Horgan and London, 1992) requires high human interaction to create test data, test script, and evaluate test results. Similarly, *TOBIAS* (Dadeau et al., 2007) requires significant effort of test engineer to write test patterns based on their knowledge. In this situation, *Raspect* (Xie et al., 2006) requires less manual inspection, since it automatically detects redundant test cases using program algebraic specifications. In contrast, some tools such as *RUTE-J* (Andrews et al., 2006), *Randoop* (Pacheco and Ernst, 2007), *GenRed* (Jaygarl et al., 2010), and *Open-SourceRed* (Kauffman and Kapfhammer, 2012) require high user experience regarding tool interaction in order to enter some technical inputs. For example, *RUTE-J* (Andrews et al., 2006) heavily depends on user interaction experience as it has no intelligence to automatically check the correctness of the run-time entered values, including method parameters range and weighting scheme. Similarly, *Randoop* (Pacheco and Ernst, 2007) requires the test engineer's interaction to set the time limits for determining the RS. In this situation, *Jtop* (Zhang et al., 2009) outperformed other proposed tools since it does not require any user experience. *Jtop* automatically extracts the static call graph in order to generate non-redundant test cases. Future tools can further improve the capability of *Jtop* by constructing a dynamic call graph (Lee et al., 2007), which extracts an exact record of program execution frequencies of call relationship based on the profile information. Ultimately, it is beneficial in optimizing overall program under test behavior. However, some cost may be required in terms of non-invasive capture tools to adequately capture the execution traces for generating profile information.

## 3.2. User Session Testing

Web applications have been widely accepted as a cost-effective communication medium for business organizations over past few years. Due to complexities of web applications, including frequent user interaction and heterogeneous components, traditional testing techniques and theories cannot be directly used to ensure the quality of web applications (Li and Xing, 2011). User session testing is increasingly adopted for web application's testing (Di Lucca and Fasolino, 2006). It mainly focuses on selecting user session data such as sequence of user actions with the web application, which are recorded in web server logs (Elbaum et al., 2005, Akhunzada et al., 2015b). Collected user session data may contain significant redundancy, since common scenarios of application execution are achieved by the users.

### 3.2.1. User Session Testing based TSR Tools

To the best of our knowledge, researchers have proposed two TSR tools namely, *USbRed* (Sampath et al., 2007) and *CPUT* (Sampath et al., 2011), to support user session testing for web applications by determining and eliminating the duplicate user session data.

- *USbRed*: Sampath et al. (Sampath et al., 2007) proposed a framework supported by *USbRed* tool to incrementally reduce a set of recorded user session data. It mainly focuses on maintaining coverage and fault detection capabilities in terms of all base requests and use case representation. The authors applied *concept analysis* to determine the minimal test suite by reducing a set of user sessions. Concept analysis is a clustering technique that groups the common user sessions based on similar distinct attributes (Di Lucca and Fasolino, 2006). The authors reported that *USbRed* along with three proposed TSR heuristics effectively reduces the user- session test data along with high coverage and fault detection of base requests.
- *CPUT*: Sampath et al. (Sampath et al., 2011) presented a general tool designed to test web applications, called *CPUT* that supports the user by easily collecting and reducing the user-session based test cases. The authors conducted an experimental study to show the efficacy of *CPUT* using a small-size web application and reported that the tool achieved nearly 20% of TSR. However, they did not mention the achieved fault-detection rate.

### 3.2.2. Comparison of User Session Testing based TSR Tools

User session testing based TSR tools, including *USbRed* (Sampath et al., 2007) and *CPUT* (Sampath et al., 2011), offers an effective and cheap mechanism for selecting representative user session data by capturing execution traces as created by real users. However, they need non-invasive capture tools to adequately capture the execution traces. On the other hand, they provide an external (or black-box) point of view, which is cost-effective in comparison to code coverage-based techniques and useful to expose functionality-related faults. Furthermore, proposed tools significantly decreased the testing cost in terms of finding the inputs as they generate a large pool of test cases without analyzing the implementation details. However, they achieved limited code coverage as specific input is required to exercise certain program paths. Ultimately, they never ensure exercising all program paths.

*USbRed* (Sampath et al., 2007) generates a large session data covering commonly used scenarios only, but incapable to explore the user session data in a systematic manner. Consequently, it lacks in covering rarest scenarios, which are useful in detecting critical faults. In contrast, *CPUT* (Sampath et al., 2011) ensures t-way such as 2-way combinatorial coverage of inter-window interactions to select a rare pair, which is effective to determine different faults using common execution traces. Combinatorial explosion is one main issue that can negatively affect the performance of *CPUT* (Sampath et al., 2011). Clustering of similar user sessions may be a viable solution to avoid the combinatorial explosion. On the other hand, to capture HTTP request along with related data, *CPUT* (Sampath et al., 2011) is supported by a generalized logger, which can be easily deployed on publicly available *Apache* server running on Linux or Windows platform. Conversely, *USbRed* (Sampath et al., 2007) is supported by an uncommon *Resin* web server. Both *USbRed* (Sampath et al., 2007) and *CPUT* (Sampath et al., 2011) use a different format of test cases. *USbRed* (Sampath et al., 2007) uses ordered HTTP requests as a test case. While, *CPUT* (Sampath et al., 2011) uses XML format based test cases, which can be easily parsed and processed by open-source XML parser and replay tools, respectively.

There are two main issues of *USbRed* (Sampath et al., 2007) and *CPUT* (Sampath et al., 2011) tools: (i) captured user session data becomes invalid due to small modification in web applications (e.g., change in page name, links, and options) and (ii) generated all user session data cannot be executed due to time constraints. Future tools should need to consider most recent user sessions to solve the test-suite size problem by finding minimal test inputs. Moreover, future tools can employ genetic algorithm (Harman and Jones, 2001) or software agent-based technology (Jennings, 2000), which has the high potential to provide efficient and effective solution for testing dynamic behavior of web applications.

## 3.3. Retargeted Compilers Testing

Processors need to be redesigned to provide better solutions for embedded software, which consequently require constructing new compilers for redesigned processors (Woo et al., 2007). The retargeting compiler is an efficient approach that supports the development of a new compiler for a processor by reusing and adjusting the existing compilers rather than building it from scratch (Boujarwah and Saleh, 1997). Generally, a large number of test cases are generated using source code by employing source-language grammar coverage criteria. Consequently, it demands automation support to reduce the test-suite size.

### 3.3.1. Retargeted Compilers Testing based TSR Tools

To the best of our knowledge, researchers have proposed two tools, including *RTL* (Woo et al., 2007) and *PLOOSE* (Chae et al., 2011) that focus on testing the back-end of a retargeted compiler for reducing the test suite, since the back-end relies on the targeted processor.

- *RTL*: Woo et al. (Woo et al., 2007) proposed a high-level abstraction based TSR framework for retargeted compilers by using the intermediate representations of test input programs. Notice that intermediate representation is coded in Register Transfer Language (RTL). The proposed framework consists of two tools, namely *Test Generator* and *Test Filter*, to filter test cases based on the RTL rule coverage criterion. The authors experimentally evaluated the effectiveness of C-covering and RTL-covering test suites. They concluded that RTL achieved a significant test-size reduction for the considered study.
- *PLOOSE*: Chae et al. (Chae et al., 2011) extended RTL-based framework by proposing a new tool, called *PLOOSE* that determines minimal tests based on intermediate representation of test inputs. It contains two main components: (i) *Test Suite Generator*, generates a number of test cases based on the given C grammar-coverage criteria, and (ii) *Test Suite Reducer*, converts compatible tests into RTL code using a translator, then finds and eliminates similar tests using RTL coverage. The authors conducted an empirical study and reported that *PLOOSE* achieved over 90% on average test-size reduction

### 3.3.2. Comparison of Retargeted Compiler Testing based TSR Tools

*RTL* (Woo et al., 2007) and *PLOOSE* (Chae et al., 2011) primarily focused on testing the back-end of a retargeted compiler using intermediate representation to determine the RS. Researchers have mentioned two main advantageous of intermediate code-based testing for retargeted compilers: (i) determining more efficient test cases compared to the traditional code-based testing and (ii) adequately testing retargeted compiler in case of time limitations (Chae et al., 2011). *RTL* determines the RS based on grammar-based test generation approaches and retains redundant test cases in initial test optimization stage. However, *RTL* requires significant time in terms of test generation, which ultimately do not scale for large-size embedded applications. Future tools can enhance the performance of *RTL* by avoiding such redundant test cases, which would ultimately need to be eliminated. In contrast, *PLOOSE* considers additional grammar-coverage criteria to generate and minimize test suites. However, its *Test Suite Reducer* component needs to be enhanced in order to support new TSR techniques and intermediate languages.

## 3.4. Integer Linear Programming

All previously discussed tools except TEMSA (Wang et al., 2015) focused on solving single-objective TSR problem (i.e. to reduce the test-suite size by maintaining high coverage using certain criteria of the SUT), which consequently produces a sub-optimal solution. However, more than one objective such as minimal test execution time and maximal fault-detection is required to determine an optimal solution. Integer Linear Programming (ILP) is commonly used technique to determine the possible “Best” solution for a defined mathematical model for a set of objectives and constraints (Williams, 2013, Akhunzada et al., 2015c).

### 3.4.1. Integer Linear Programming based TSR Tools

To the best of our knowledge, researchers have proposed two tools namely, *MINTS* (Hsu and Orso, 2009) and *EDTSO* (Li et al., 2013), which are based on encoding TSR problem as an ILP problem.

- *MINTS*: Hsu and Orso (Hsu and Orso, 2009) proposed a framework to support a wide range of TSR problems by handling various objectives using ILP-based approach. The proposed framework is supported by a tool, called *MINTS* that provides the flexibility of specifying multi-objective TSR problems. Based on the defined encoding mechanism, related objectives are given directly to one or more supporting ILP solvers. The proposed tool is freely available and due to its modular structure it can be easily plug-in different ILP solvers. The authors experimentally evaluated the performance of the tool and concluded that *MINTS* is effective and practical to test-suite size problem.
- *EDTSO*: Li et al. (Li et al., 2013) proposed a novel approach in order to optimize the energy usage of test suites. The proposed approach is supported by a prototype tool, called *EDTSO*, which focuses on an energy-efficient reduced test suite. *EDTSO* is effective for testing Android applications having limited energy budget by automatically producing energy-efficient RS. The authors performed a pair-wise comparison of the energy usage of the RS, including variability, potential impact, and effectiveness as determined by the proposed approach and traditional approach. They reported that *EDTSO* maintained coverage along with minimal energy consumption ranging from 5% to 48% compared to traditional TSR techniques. Later, Li et al. (Li et al., 2014) extended *EDTSO* by integrating it with existing test workflows and also by utilizing other resource constraints such as test execution time.

### 3.4.2. Comparison of Integer Linear Programming based TSR Tools

Integer Linear Programming (ILP) based TSR tools, including *MINTS* (Hsu and Orso, 2009) and *EDTSO* (Li et al., 2013), determined an optimal solution using multiple ILP solvers by considering defined objectives and constraints. In addition, improved execution time is achieved using a wide range of ILP solvers by running multiple solvers in parallel fashion and then simplifying the generated solutions in a sequential manner. *MINTS* provides any set of test-related data according to the need of a testing environment. Appropriate selection of test data greatly affects the outcome of *MINTS*, which heavily depends on the tester's expertise. Furthermore, non-availability of historical testing information makes *MINTS* impractical to compute the optimal solution. Moreover, both *MINTS* and *EDTSO* need to execute each test case in order to collect profiles such as time or memory complexity and test-code structural coverage by instrumenting the targeted program. Certainly, high execution cost is required for instrumentation, which might be a key concern while employing these TSR tools.

## 3.5. Automated Fault-Localization

Fault-localization (also known as fault isolation or root cause analysis) is the most expensive and time consuming activity in debugging (Gong et al., 2015). It focuses on finding the exact location of SUT's faults to improve the fault-detection process (DiGiuseppe and Jones, 2014).

### 3.5.1. Automated Fault-Localization based TSR Tools

To the best of our knowledge, researchers have proposed three tools including *SrTC* (Zhang et al., 2010), *JINSI* (Burger and Zeller, 2011) and GZoltar (Campos et al., 2012), to improve fault-localization effectiveness with the distribution uniformity of test cases.

- *SrTC*: Zhang et al. (Zhang et al., 2010) presented very first time the concept of relative redundancy for TSR and focused to balance the uneven distribution in the RS. The authors proposed a framework supported by *SrTC* tool, which intentionally retained a small number of redundant test cases to improve fault-localization effectiveness. *SrTC* contains three modules: (i) *Reduction Processor*, (ii) *w Processor*, and (iii) *Evaluation*, which uses and produces five types of data: (1) *T*, (2) *REP*, (3) *TIE*, (4) *CAN*, and (5) *REL-REP*. To judge the efficacy of *SrTC*, the authors conducted an experimental study by using *NanoXM* (Do et al., 2004) as a subject program. According to the reported results, the average value of reduction was 24.25%. However, the fault-localization effectiveness was improved significantly.

- *JINSI*: Burger and Zeller (Burger and Zeller, 2011) proposed a *JINSI* tool that uses passing and failing execution information in order to determine faulty program entities. First of all, *JINSI* records and replays interactions of a given objects set and effectively reproduces the failure inducing method calls. After that, *JINSI* filters the sequence of failure inducing method calls by combining event slicing and delta debugging techniques, which result in a minimized object interaction. Eventually, the minimal set of unit tests is generated using the reduced interaction information. The authors experimentally evaluated the performance of the *JINSI* tool by applying it on six different Java subject programs. Finally, they reported that *JINSI* achieved an average of 0.22% search space reduction of the original Java program and produced a test driver containing eight to twelve failure-reproducing interactions.
- *GZoltar*: Campos et al. (Campos et al., 2012) presented a GZoltar toolset that is a spectrum-based fault localization plug-in for the Eclipse Integrated Development Environment (IDE). The proposed tool supports a constraint-based approach (Campos, 2012) to perform TSR, while ensuring the same code coverage compared to the original test suite. Moreover, GZoltar facilitate the user to prioritize the RS based on the test execution time and cardinality. It also automatically produces different visual representations of the diagnostic report, which ultimately facilitate the users in detecting the most suspicious parts of the program's source code. As a result, the fault localization process can be improved using minimal possible time. GZoltar provides the infrastructure to instruments the source code of the SUT in order to automatically generate runtime data. Next, the test cases are executed to collect the code coverage data and execution traces. Finally, the collected coverage information is passed to constraint solver in order to determine minimal RS.

### 3.5.2. Comparison of Automated Fault-Localization based TSR Tools

The researchers have concluded that faults usually reside in execution paths of failed test cases (Dandan et al., 2013). Thus, the statements in failed test cases' paths might be more beneficial in improving fault-localization effectiveness. However, *SrTC* (Zhang et al., 2010) only consider the coverage information of test cases instead of concrete path information. Consequently, *SrTC* (Zhang et al., 2010) may remove the test cases which are relevant to fault-localization requirements and ultimately achieves less fault-localization effectiveness. In contrast, *JINSI* (Burger and Zeller, 2011) performs better in terms of improving fault-localization effectiveness, since it minimizes failing test cases based on call/return traces. Ultimately, *JINSI* (Burger and Zeller, 2011) has the high potential to determine the significant number of faults than *SrTC* (Zhang et al., 2010). In comparison to *SrTC* (Zhang et al., 2010) and *JINSI* (Burger and Zeller, 2011), *GZoltar* (Campos et al., 2012) can speed-up the fault-localization process, since it provides different visual representations of the generated diagnostic report.

## 4. Thematic Taxonomy based Comparison of Test Suite Reduction (TSR) Tools

This section compares existing Test Suite Reduction (TSR) tools based on the devised thematic taxonomy (presented in Section 2) to highlight the commonalities and differences among the reported tools (depicted in Table 1 and 2). We critically analyzed current TSR tools based on several parameters: (i) approach type, (ii) testing paradigm, (iii) optimization type, (iv) coverage source, (v) execution platform, (vi) computational mode, vii) license type, (viii) evaluation, (ix) customizability, and (x) support.

Current state-of-the-art TSR tools can be classified into coverage-based, search-based, similarity-based, and Integer Linear Programming (ILP) based approaches. Majority of the existing tools (Horgan and London, 1992, Xie et al., 2004, Xie et al., 2006, Andrews et al., 2006, Pacheco and Ernst, 2007, Jaygarl et al., 2010, Kauffman and Kapfhammer, 2012, Sampath et al., 2011, Woo et al., 2007, Chae et al., 2011, Zhang et al., 2010, Burger and Zeller, 2011, Campos et al., 2012, Dadeau et al., 2007) consider coverage of the SUT as a base to determine the reduced suite. However, Open-SourceRed (Kauffman and Kapfhammer, 2012) and TEMSA (Wang et al., 2015) follows search-based techniques to find the diverse subset of a test suite. Similarly, *MINTS* (Hsu and Orso, 2009) and *EDTSO* (Li et al., 2013) employ ILP-based approach to determine the global optimal solution for TSR problem. In contrast, *USbRed* (Sampath et al., 2007) employs similarity-based technique to find the most different test cases from a test suite. In comparison to search-based approaches, coverage-based approaches require extra computational time to determine the coverage of a program using certain coverage criteria such as statement, branch, or path coverage. Moreover, coverage-based approaches may produce optimal or near-optimal solution using greedy algorithms (Khan et al., 2013). In contrast, search-based approaches employ different types of search algorithms, such as Genetic Algorithm, to determine a global optimal solution based on the defined fitness function. However, search-based approaches can be very time-consuming especially if the fitness function is evaluated in terms of increased coverage provided by the different possible solutions.

The attributes of *testing paradigm* parameters state the type of programming languages in which the targeted SUT is developed. The TSR tools are broadly classified into structured, object-oriented, and aspect-oriented implementation platforms. The tools in (Horgan and London, 1992, Woo et al., 2007, Chae et al., 2011), (Xie et al., 2004, Andrews et al., 2006, Pacheco and Ernst, 2007, Jaygarl et al., 2010, Zhang et al., 2009, Kauffman and Kapfhammer, 2012, Sampath et al., 2007, Sampath et al., 2011, Hsu and Orso, 2009, Li et al., 2013, Zhang et al., 2010, Burger and Zeller, 2011), and (Xie et al., 2006), consider structured, object-oriented, and aspect-oriented programming platforms, to minimize the test-suite size, respectively. The TSR tools can be categorized into single-objective and multi-objective tools. The single-objective based *optimization type* focuses on cost such as loss in fault-detection capability or effectiveness in terms of size of the RS. Alternatively, *MINTS* (Hsu and Orso, 2009) and *EDTSO* (Li et al., 2013) focus on the multi-objective optimization type using ILP-based approach, which attempts to balance the tradeoffs between both cost and effectiveness measures. Multi-objective is preferable to the tester as it generates different possible solutions of considered objectives and constraints, which consequently helps the tester in the decision-making process. However, in comparison, single-objective takes less computation time, since it only considers either cost or effectiveness measure.

The *coverage source* provides a base to determine the representative test cases. The tools in (Horgan and London, 1992, Andrews et al., 2006, Pacheco and Ernst, 2007, Jaygarl et al., 2010, Zhang et al., 2009, Sampath et al., 2007, Sampath et al., 2011, Burger and Zeller, 2011, Zhang et al., 2010, Kauffman and Kapfhammer, 2012, Woo et al., 2007) and (Xie et al., 2004, Xie et al., 2006) consider source code and test execution profile, respectively. In comparison to source code, test execution profiles are expensive as they can be collected by executing the entire test suite prior to initiating the test reduction process. The *execution platform* considers a single server and multiple server attributes to support the reduction process. Among others, *MINTS* (Hsu and Orso, 2009) and *EDTSO* (Li et al., 2013) use multiple servers attribute to concurrently solve the TSR problem. As a result, the computational time taken by *MINTS* (Hsu and Orso, 2009) and *EDTSO* (Li et al., 2013) significantly reduced at the cost of extra hardware resources. The *computational mode* exploits offline (Horgan and London, 1992, Xie et al., 2004, Xie et al., 2006, Andrews et al., 2006, Pacheco and Ernst, 2007, Jaygarl et al., 2010, Zhang et al., 2009, Burger and Zeller, 2011, Zhang et al., 2010, Kauffman and Kapfhammer, 2012, Woo et al., 2007) and online (Sampath et al., 2007, Sampath et al., 2011) attributes to generate the representative solution. In comparison to offline, online computational mode supports real-time testing, especially in the case of web-based applications.

The attributes of *license type* parameters define whether the TSR tool is freely available or a commercial product. The tools in (Horgan and London, 1992, Woo et al., 2007) and (Sampath et al., 2011) are available as free and proprietary, respectively. Analysis of TSR tools reveals that most of the existing tools are products of multiple academic research projects. While commercial support for such tools is rather scarce. In contrast, most of the freely available open source tools provide full *customizability* (ATAC (Horgan and London, 1992), RUTE-J (Andrews et al., 2006), Randoop (Pacheco and Ernst, 2007), Jtop (Zhang et al., 2009), Open-SourceRed (Kauffman and Kapfhammer, 2012)) so that desired alterations can be performed to adapt and enhance these tools according to the diverse set of requirments. While numerous other tools allow only limited (GZoltar (Campos et al., 2012), MINTS (Hsu and Orso, 2009), JINSI (Burger and Zeller, 2011)) or no customizability (TEMSA [34], Rostra [27], GenRed [31], USbRed [36], CPUT [37], EDTSO [41]). It is really hard to find the available *support* for numerous tools. A significant hurdle in this regard is language barrier as some research groups have their web sites and support in languages other than English (RTL (Woo et al., 2007), PLOOSE (Chae et al., 2011), TOBIAS (Dadeau et al., 2007)). This makes it hard to find the available support for respective tools. Moreover some proprietary academic research tools (USbRed (Sampath et al., 2007), CPUT (Sampath et al., 2011)) were only accessible through private search spaces, making it difficult to access executables and supporting materials. While considering *evaluation* of these tools, very few tools have been externally evaluated. This fact points to a significant research gap regarding empirical evaluation of existing TSR tools. Most of the freely available TSR tools are open source and we have provided download links for source codes and support for these tools (ATAC (Horgan and London, 1992), RUTE-J (Andrews et al., 2006), Randoop (Pacheco and Ernst, 2007), Jtop (Zhang et al., 2009), Open-SourceRed (Kauffman and Kapfhammer, 2012), MINTS (Hsu and Orso, 2009)). This aggregation of information will help researchers in acquiring useful information for empirical evaluation and extension of existing tools.

Table 1: Taxonomy based Comparison of TSR Frameworks/Tools

| **Tool Name** | **Approach Type** | **Testing Paradigm** | **Optimization Type** | **Coverage Source** | **Execution Platform** | **Computational Mode** | **License** |
|---|---|---|---|---|---|---|---|
| ATAC (Horgan and London, 1992) | Coverage-based | Structured | Single-objective | Source Code | Single Server | Offline | Free |
| Rostra (Xie et al., 2004) | Coverage-based | Object-oriented | Single-objective | Execution Profile | Single Server | Offline | Research |
| Raspect (Xie et al., 2006) | Coverage-based | Aspect-oriented | Single-objective | Execution Profile | Single Server | Offline | Research |
| RUTE-J (Andrews et al., 2006) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Free |
| Randoop (Pacheco and Ernst, 2007) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Free |
| GenRed (Jaygarl et al., 2010) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Research |
| Jtop (Zhang et al., 2009) | - | Object-oriented | Single-objective | - | Single Server | Offline | Free |
| TOBIAS (Dadeau et al., 2007) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Research |
| TEMSA (Wang et al., 2015) | Search-based | Object-oriented | Multi-objective | Feature Model | Single Server | Offline | Research |
| Open-SourceRed (Kauffman and Kapfhammer, 2012) | Coverage-based, Search-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Free |
| USbRed (Sampath et al., 2007) | Similarity-based | Object-oriented | Single-objective | Source Code | Single Server | Online | Research |
| CPUT (Sampath et al., 2011) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Online | Research |
| RTL (Woo et al., 2007) | Coverage-based | Structured | Single-objective | Source Code | Single Server | Offline | Research |
| PLOOSE (Chae et al., 2011) | Coverage-based | Structured | Single-objective | Source Code | Single Server | Offline | Research |
| MINTS (Hsu and Orso, 2009) | ILP-based | Object-oriented | Multi-objective | Source Code | Multiple Server | Offline | Free |
| EDTSO (Li et al., 2013) | ILP-based | Object-oriented | Multi-objective | Source Code | Multiple Server | Offline | Research |
| SrTC (Zhang et al., 2010) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Research |
| JINSI (Burger and Zeller, 2011) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Research |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| GZoltar (Campos et al., 2012) | Coverage-based | Object-oriented | Single-objective | Source Code | Single Server | Offline | Research / Commercial |

Table 2: Taxonomy based Comparison of TSR Frameworks/Tools

| **Tool Name** | **Evaluation** | | **Customizability** | | **Support** | | | |
|---|---|---|---|---|---|---|---|---|
| | **Internal** | **External** | **Full** | **Partial** | **Written Documentation** | **Executable** | **Web Site** | **Web link (Source Code, exe files, packages, Help and documentation)** |
| ATAC (Horgan and London, 1992) | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ | http://invisible-island.net/atac/atac.html |
| Rostra (Xie et al., 2004) | ✓ | ✓ | × | × | × | × | × | http://taoxie.cs.illinois.edu/research.htm |
| Raspect (Xie et al., 2006) | ✓ | × | × | × | × | × | × | http://taoxie.cs.illinois.edu/publications.htm |
| RUTE-J (Andrews et al., 2006) | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ | http://staff.unak.is/andy/MScTesting0708/Assignments/RUTEJstuff/rutej-1.2.zip<br>http://staff.unak.is/andy/MScTesting0708/assignments.htm |
| Randoop (Pacheco and Ernst, 2007) | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ | http://mernst.github.io/randoop/<br>http://randoop.codeplex.com/ |
| GenRed (Jaygarl et al., 2010) | ✓ | × | × | × | × | × | × | N/A |
| Jtop (Zhang et al., 2009) | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ | https://code.google.com/p/pku-jtop/ |
| TOBIAS (Dadeau et al., 2007) | ✓ | ✓ | N/A | N/A | ✓ | N/A | ✓ | http://www.irisa.fr/en/cote<br>Supporting information in French |
| TEMSA (Wang et al., 2015) | ✓ | × | × | × | ✓ | ✓ | ✓ | http://zen-tools.com/TEMSA/ |
| Open-SourceRed (Kauffman and Kapfhammer, 2012) | ✓ | × | ✓ | × | ✓ | ✓ | ✓ | https://github.com/kauffmj/modificare |
| USbRed (Sampath et al., 2007) | ✓ | × | × | × | × | × | ✓ | http://hiper.cis.udel.edu/doku.php/projects/webapps<br>http://hiper.cis.udel.edu/udsacl/doku.php/research/home (Pvt research space) |
| CPUT (Sampath et al., 2011) | ✓ | × | × | × | × | × | ✓ | http://userpages.umbc.edu/~sampath/CPUT_Web.html |
| RTL (Woo et al., 2007) | ✓ | × | N/A | N/A | N/A | N/A | × | http://oos.cse.pusan.ac.kr<br>Supporting information in Korean |
| PLOOSE (Chae et al., 2011) | ✓ | × | N/A | N/A | N/A | N/A | ✓ | http://oos.cse.pusan.ac.kr/ploose/<br>Supporting information in Korean |
| MINTS (Hsu and Orso, 2009) | ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ | http://www.cc.gatech.edu/~orso/software/mints/<br>http://www.cc.gatech.edu/~orso/software/mints/mints.tgz |
| EDTSO (Li et al., 2013) | ✓ | × | × | × | × | × | × | http://www-scf.usc.edu/~dingli/ |
| SrTC (Zhang et al., 2010) | ✓ | × | N/A | N/A | N/A | N/A | N/A | N/A |
| JINSI (Burger and Zeller, 2011) | ✓ | ✓ | × | ✓ | ✓ | ✓ | × | https://www.st.cs.uni-saarland.de/dd/ |
| GZoltar (Campos et al., 2012) | ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ | http://www.gzoltar.com/<br>http://www.gzoltar.com/lib/ |

## 5. Open Research Issues

This section presents potential research issues related to automation support for TSR. The highlighted research issues will assist the researchers in improving limitations of existing TSR tools. Furthermore, it would help in proposing such tools that can efficiently meet the challenges of increasingly popular applications, including software product lines (Bosch, 2002), mobile applications (Zhang and Adipat, 2005), service-oriented architectures (Erl, 2005), web applications (Ali et al., 2007), and cloud-based applications (Orso and Rothermel, 2014).

### 5.1. Multi-Objective TSR Optimization Support

Naturally, TSR problem is a multi-objective optimization problem focusing on various defined cost and effectiveness related challenges. This challenge becomes more complicated due to the involvement of many user-defined conflicting objectives and constraints for TSR optimization. Consequently, the tester needs to define a minimal threshold for cost-effectiveness measure, which would be finally compared with the obtained results. However, finding an optimal threshold depends on different factors, which varies from system to system. For example, in the case of web-based systems, some loss in FDE can be acceptable compared to safety critical systems, which requires 100% FDE due to high impact on human life. The proposed tool should be able to define an optimal threshold value based on the given factors. Extensive system profiling enables a tool to accurately estimate the desirable threshold. Current TSR tools (82%, 16/19) mainly focused on the single-objective optimization type such as to determine the minimal RS, which is impractical for a testing scenario containing multiple objectives and constraints for optimization (Harman, 2011). Conversely, researchers have to develop multi-objective supported tools that can efficiently meet the testing challenges of future applications.

### 5.2. Hyper-Heuristic Software Testing

The “hyper-heuristic software testing” opens a new way for TSR, which involves combining different regression testing activities such as reduction, selection, and prioritization, and employ various types of search algorithms such as *Genetic Algorithm* and *Hill Climbing* to generate a best-possible TSR optimization. Therefore, a TSR tool can utilize the hyper-heuristic software testing opportunity to produce an optimal solution by focusing on multiple regression activities rather than applying single activity. For example, the TSR tool can first determine the RS (i.e. reduction) and then finds the high-value (i.e. prioritization) test cases to get the right execution order of reduced tests (Khan et al., 2009). The tool can employ search algorithms to perform both reduction and prioritization. Similarly, the other possible scenarios (e.g., reduction and then prioritization, and prioritization and then reduction) can also be supported by the proposed tools.

### 5.3. Multiple Server-based TSR Optimization

The use of multiple servers is beneficial in improving the computational time, especially for large complex systems having an enormous pool of test cases (Sprenkle et al., 2005). So, in this situation, the tester can achieve efficient computational time at the cost of extra required servers. This could be feasible in a situation, where divide-and-conquer strategy is acceptable. For instance, first the optimization problem is divided into many sub-problems, and then each sub-problem is independently solved using a single server. Thus, the proposed tool can act as a controller to manage multiple servers. It first determines the sub-problems and assigns each sub-problem to the available server for optimization. Next, the tool collects the computed solutions from each server and merges into a single solution. After that, the tool removes the duplication from the obtained solutions (from multiple servers) to generate a final optimal solution. However, resource affordability plays a vital role in the adoption of concurrent processing. This could be possible if low-cost servers are available or the tester has sufficient computational resources. One feasible solution is to employ Cloud Computing (Shiraz et al., 2015) for achieving low-cost computational resources without compromising the quality of the obtained optimized solution.

### 5.4. Automated Evaluation of Solution Quality

Researchers can determine efficacy of obtained test reduction solution based on the proposed measures of cost (i.e. loss in FDE and tool execution time) and effectiveness (i.e. size of reduced suite and achieved coverage of the SUT). There may be a high possibility of miscalculations in determining the efficacy of an optimized solution due to human involvement in the test result's evaluation process. Since, cost-effectiveness measures are well-defined; they can be easily implemented and embedded into the TSR tool. Consequently, it helps to generate the RS along with associated statistical analysis (using defined cost-effectiveness measures) that provides confidence to the tester about the obtained solution. Similarly, the statistical report can assist the tester in the decision-making process by providing a multi-facet view of the targeted TSR optimization problem.

### 5.5. Hybrid Solution and Agile Software Development

Hybrid solutions for optimization problems show significant improvements in existing results by combining the strong points of two different TSR approaches (Yoo and Harman, 2010). Future tools can use the strengths of existing TSR approaches such as coverage-based and search-based to find a more effective solution for TSR problem (Harman and McMinn, 2010). For example, the future tool first selects a set of diverse test cases by employing a search-based approach. Next, the adequacy of computed solution in terms of attained coverage is computed by using the coverage-based approach. Consequently, the integration of search-based and coverage-based approaches helps to provide an optimal solution of TSR problem with respect to high coverage of SUT. Notice that achieved coverage of the computed solution acts as a quality indicator, which is beneficial in deciding whether the test optimization process should be continued or stopped.

The TSR tool is highly beneficial to support agile software-development process (Stober and Hansmann, 2010). One of the important practices of agile software development is on the notion “testing early and testing often". In this scenario, test cases signify instant feedback to the tester, whether the tested portion of a program is free of error or introduces new regression errors. Such tool would be also practical for web application testing using online mode of computation.

## 6. Conclusion and Future Remarks

Test Suite Reduction (TSR) remains a predominant research area for past two decades, which has resulted in various types of TSR tools supporting different testing domains. This paper has critically evaluated current state-of-the-art TSR frameworks and tools. Furthermore, we provided a global view on automation support perspective of TSR by analyzing the reported tools based on a set of comparison criteria. A thematic taxonomy was devised to classify the existing literature. Moreover, several potential research issues are presented that need further research to develop cost-effective TSR tools.

Current state-of-the-art TSR frameworks and tools have mainly focused on coverage-based approaches (74%, 14/19) that exploit the coverage of a system under test to determine the reduced suite. However, achieving high coverage of a software system seldom guarantees to detect all possible faults. Alternatively, search-based approaches have the better potential to expose real faults by finding the diversity among prescribed test cases. Conversely, Integer Linear Programming-based approaches ensure Pareto-optimal solution for TSR. In contrast, similarity-based approaches generate a desired number of test cases having minimal similarity without considering coverage requirements of the system under test. On the other hand, existing TSR tools mainly targeted to solve the single-objective TSR optimization problems (82%, 16/19), which is impractical for a testing scenario containing multiple objectives and constraints. Therefore, it is crucial to pay more attention on multi-objective optimization problems in order to find better cost-effective solutions according to the nature of the application under test. Furthermore, existing TSR tools determine the reduced test suite in a sequential manner. Nevertheless, concurrent computation using multiple resources augments the desired computational time.

The consideration of multi-objective optimization, hyper-heuristic software testing, concurrent processing, automated result evaluation, agile development support, and hybrid solutions can augment the capabilities of existing

TSR tools. Recently, similarity-based approaches are gaining high attention of researchers to provide an optimal solution to TSR problem. Hence, future research should focus on devising similarity-based automation support for TSR. In future, we plan to develop a multi-server based TSR tool to efficiently solve the multi-objective TSR optimization problems.

**Acknowledgements**

This work is carried out within the framework of the research project supported by High Impact Research Grant with reference UM.C/625/1/HIR/MOHE/FCSIT/13, funded by the Ministry of Education, Malaysia. S.U.R. Khan would like to acknowledge the financial support of the Bright Sparks Program at University of Malaya.

## References

AKHUNZADA, A., GANI, A., ANUAR, N. B., ABDELAZIZ, A., KHAN, M. K., HAYAT, A. & KHAN, S. U. 2015a. Secure and dependable software defined networks. *Journal of Network and Computer Applications*.

AKHUNZADA, A., GANI, A., HUSSAIN, S. & KHAN, A. A. A formal framework for web service broker to compose QoS measures. SAI Intelligent Systems Conference (IntelliSys), 2015, 2015b. IEEE, 532-536.

AKHUNZADA, A., GANI, A., HUSSAIN, S. & KHAN, A. A. Towards experiencing the pair programming as a practice of the Rational Unified Process (RUP). SAI Intelligent Systems Conference (IntelliSys), 2015, 2015c. IEEE, 537-542.

AKHUNZADA, A., SOOKHAK, M., ANUAR, N. B., GANI, A., AHMED, E., SHIRAZ, M., FURNELL, S., HAYAT, A. & KHAN, M. K. 2015d. Man-At-The-End attacks: Analysis, taxonomy, human aspects, motivation and future directions. *Journal of Network and Computer Applications*, 48**,** 44-57.

ALI, S., BRIAND, L. C., REHMAN, M. J.-U., ASGHAR, H., IQBAL, M. Z. Z. & NADEEM, A. 2007. A state-based approach to integration testing based on UML models. *Information and Software Technology,* 49**,** 1087-1106.

ANDREWS, J. H., HALDAR, S., LEI, Y. & LI, F. C. H. Tool support for randomized unit testing. Proceedings of the First ACM International Workshop on Random Testing, 2006. 36-45.

ANDREWS, J. H., MENZIES, T. & LI, F. C. 2011. Genetic algorithms for randomized unit testing. *IEEE Transactions on Software Engineering,* 37**,** 80-94.

BALLER, H., LITY, S., LOCHAU, M. & SCHAEFER, I. Multi-objective test suite optimization for incremental product family testing. Proceedings of the Seventh IEEE International Conference on Software Testing, Verification and Validation (ICST'14), 2014 2014. 303-312.

BEIZER, B. 2002. *Software testing techniques*, Dreamtech Press.

BERTOLINO, A. Software testing research: Achievements, challenges, dreams. Proceedings of the IEEE Conference on Future of Software Engineering (FOSE'07), 2007 2007. 85-103.

BLACK, J., MELACHRINOUDIS, E. & KAELI, D. Bi-criteria models for all-uses test suite reduction. Proceedings of the 26th IEEE International Conference on Software Engineering (ICSE'04), 2004. 106-115.

BOSCH, J. 2002. Maturity and evolution in software product lines: Approaches, artefacts and organization. *In:* CHASTEK, G. J. (ed.) *Software Product Lines.* Springer Berlin Heidelberg.

BOUJARWAH, A. S. & SALEH, K. 1997. Compiler test case generation methods: a survey and assessment. *Information and software technology,* 39**,** 617-625.

BURGER, M. & ZELLER, A. Minimizing reproduction of software failures. Proceedings of the 2011 ACM International Symposium on Software Testing and Analysis (ISSTA'11), 2011. 221-231.

CAMPOS, J., RIBOIRA, A., PEREZ, A. & ABREU, R. Gzoltar: an eclipse plug-in for testing and debugging. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE'12), 2012. ACM, 378-381.

CAMPOS, J. C. D. 2012. *Regression testing with GZoltar techniques for test suite minimization, selection, and prioritization.* Master Master's Thesis, University of Porto.

CHAE, H. S., WOO, G., KIM, T. Y., BAE, J. H. & KIM, W.-Y. 2011. An automated approach to reducing test suites for testing retargeted C compilers for embedded systems. *Journal of Systems and Software,* 84**,** 2053-2064.

CHANG, V. 2014a. The business intelligence as a service in the cloud. *Future Generation Computer Systems,* 37**,** 512-534.

CHANG, V. 2014b. Cloud Computing for brain segmentation–a perspective from the technology and evaluations. *International Journal of Big Data Intelligence,* 1**,** 192-204.

CHANG, V. 2015a. A Cybernetics Social Cloud. *Journal of Systems and Software*.

CHANG, V. 2015b. An overview, examples and impacts offered by Emerging Services and Analytics in Cloud Computing. *International Journal of Information Management*.

CHANG, V. & RAMACHANDRAN, M. 2016. Towards achieving Data Security with the Cloud Computing Adoption Framework.

CHANG, V. & WILLS, G. 2016. A model to compare cloud and non-cloud storage of Big Data. *Future Generation Computer Systems,* 57**,** 56-76.

CHEN, T. Y., KUO, F.-C., MERKEL, R. G. & TSE, T. 2010. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software,* 83**,** 60-66.

COUTINHO, A., CARTAXO, E. G. & MACHADO, P. D. Test suite reduction based on similarity of test cases. 7st Brazilian Workshop on Systematic and Automated Software Testing—CBSoft 2013, 2013. Brası´lia, DF, Brazil.

DADEAU, F., LEDRU, Y. & DU BOUSQUET, L. Directed random reduction of combinatorial test suites. Proceedings of the 2nd International Workshop on Random Testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), 2007. ACM, 18-25.

DANDAN, G., TIANTIAN, W., XIAOHONG, S. & PEIJUN, M. 2013. A test-suite reduction approach to improving fault-localization effectiveness. *Computer Languages, Systems & Structures,* 39**,** 95-108.

DEB, K., PRATAP, A., AGARWAL, S. & MEYARIVAN, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation,* 6**,** 182-197.

DI LUCCA, G. A. & FASOLINO, A. R. 2006. Testing web-based applications: The state of the art and future trends. *Information and Software Technology,* 48**,** 1172-1186.

DIGIUSEPPE, N. & JONES, J. A. 2014. Fault density, fault types, and spectra-based fault localization. *Empirical Software Engineering***,** 1-40.

DO, H., ELBAUM, S. & ROTHERMEL, G. Infrastructure support for controlled experimentation with software testing and regression testing techniques. Proceedings of the IEEE International Symposium on Empirical Software Engineering (ISESE'04) 2004. 60-70.

ELBAUM, S., ROTHERMEL, G., KARRE, S. & II, M. F. 2005. Leveraging user-session data to support web application testing. *IEEE Transactions on Software Engineering,* 31**,** 187-202.

ELBERZHAGER, F., ROSBACH, A., MÜNCH, J. & ESCHBACH, R. 2012. Reducing test effort: a systematic mapping study on existing approaches. *Information and Software Technology,* 54**,** 1092-1106.

ERL, T. 2005. *Service-oriented architecture: concepts, technology, and design*, Pearson Education India.

GONG, D., SU, X., WANG, T., MA, P. & YU, W. 2015. State dependency probabilistic model for fault localization. *Information and Software Technology,* 57**,** 430-445.

HAO, D., ZHANG, L., WU, X., MEI, H. & ROTHERMEL, G. On-demand test suite reduction. Proceedings of the IEEE International Conference on Software Engineering (ICSE'12), 2012. 738-748.

HARMAN, M. 2011. Software engineering meets evolutionary computation. *Computer,* 44**,** 31-39.

HARMAN, M. & JONES, B. F. 2001. Search-based software engineering. *Information and Software Technology,* 43**,** 833-839.

HARMAN, M. & MCMINN, P. 2010. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Transactions on Software Engineering,* 36**,** 226-247.

HARROLD, M. J., GUPTA, R. & SOFFA, M. L. 1993. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology (TOSEM),* 2**,** 270-285.

HAUG, M., OLSEN, E. W. & CONSOLINI, L. 2001. *Software quality approaches: testing, verification, and validation*, Springer.

HORGAN, J. R. & LONDON, S. A data flow coverage testing tool for C. Proceedings of the Second IEEE Symposium on Assessment of Quality Software Development Tools 1992. 2-10.

HSU, H.-Y. & ORSO, A. MINTS: a general framework and tool for supporting test-suite minimization. Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE'09) 2009. 419-429.

JACCARD, P. 1901. Etude comparative de la distribution florale dans une portion des Alpes et du Jura. *Bulletin of the Waldensian Society of Natural Sciences,* 37.

JAYGARL, H., LU, K.-S. & CHANG, C. K. GenRed: a tool for generating and reducing object-oriented test cases. Proceedings of the 34th Annual IEEE Computer Software and Applications Conference (COMPSAC'10) 2010. 127-136.

JENNINGS, N. R. 2000. On agent-based software engineering. *Artificial intelligence,* 117**,** 277-296.

KAUFFMAN, J. M. & KAPFHAMMER, G. M. A framework to support research in and encourage industrial adoption of regression testing techniques. Proceedings of the Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST'12), 2012. 907-908.

KHAN, S. U. R., NADEEM, A. & AWAIS, A. TestFilter: a statement-coverage based test case reduction technique. Proceedings of the 10th IEEE International Multi-Topic Conference (INMIC'06), 2006. 275-280.

KHAN, S. U. R., PECK, L. S., PARIZI, R. M. & ELAHI, M. An analysis of the code coverage-based greedy algorithms for test suite reduction. Proceedings of the Second International Conference on Informatics Engineering & Information Science (ICIEIS'13), 2013. The Society of Digital Information and Wireless Communication, 370-377.

KHAN, S. U. R., REHMAN, I. U. & MALIK, S. U. R. The impact of test case reduction and prioritization on software testing effectiveness. Proceedings of the Sixth IEEE International Conference on Emerging Technologies (ICET'09) 2009. 416-421.

KUHN, D. R. & OKUN, V. Pseudo-exhaustive testing for software. Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop (SEW'06) 2006. 153-158.

LEE, B., RESNICK, K., BOND, M. D. & MCKINLEY, K. S. Correcting the dynamic call graph using control-flow constraints. *In:* KRISHNAMURTHI, S. & ODERSKY, M., eds. Proceedings of the 16th International Conference (CC'07) held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'07) 2007. Springer Berlin Heidelberg, 80-95.

LEE, J. & CHUNG, C. 2000. An optimal representative set selection method. *Information and Software Technology,* 42**,** 17-25.

LEON, D. & PODGURSKI, A. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. Proceedings of the 14th IEEE International Symposium on Software Reliability Engineering (ISSRE'03) 2003. 442-453.

LI, D., JIN, Y., SAHIN, C., CLAUSE, J. & HALFOND, W. G. Integrated energy-directed test suite optimization. Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA'14), 2014. 339-350.

LI, D., SAHIN, C., CLAUSE, J. & HALFOND, W. G. Energy-directed test suite optimization. Proceedings of the Second IEEE International Workshop on Green and Sustainable Software (GREENS'13), 2013. San Francisco, CA, USA, 62-69.

LI, J.-H. & XING, D.-D. 2011. User session data based web applications test with cluster analysis. *In:* SHEN, G. & HUANG, X. (eds.) *Advanced Research on Computer Science and Information Engineering.* Springer Berlin Heidelberg.

LIN, Y.-D., CHOU, C.-H., LAI, Y.-C., HUANG, T.-Y., CHUNG, S., HUNG, J.-T. & LIN, F. C. 2012. Test coverage optimization for large code problems. *Journal of Systems and Software,* 85**,** 16-27.

MICHAEL, R. G. & DAVID, S. J. 1979. Computers and intractability: a guide to the theory of NP-completeness. *WH Freeman & Co., San Francisco*.

MYERS, G. J., SANDLER, C. & BADGETT, T. 2011. *The art of software testing*, John Wiley & Sons.

NACHMANSON, L., VEANES, M., SCHULTE, W., TILLMANN, N. & GRIESKAMP, W. 2004. Optimal strategies for testing nondeterministic systems. *ACM SIGSOFT Software Engineering Notes,* 29**,** 55-64.

ORSO, A. & ROTHERMEL, G. Software testing: a research travelogue (2000–2014). Proceedings of the IEEE International Conference on Software Engineering (ICSE'14)-Track on the Future of Software Engineering, 2014. 117-132.

PACHECO, C. & ERNST, M. D. Randoop: feedback-directed random testing for Java. Proceedings of the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion, 2007. 815-816.

RAMLER, R. & WOLFMAIER, K. Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. Proceedings of the 2006 International Workshop on Automation of Software Test, 2006. ACM, 85-91.

ROTHERMEL, G., UNTCH, R. H., CHU, C. & HARROLD, M. J. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering,* 27**,** 929-948.

SAMPATH, S., BRYCE, R. C., JAIN, S. & MANCHESTER, S. A tool for combination-based prioritization and reduction of user-session-based test suites. Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM'11) 2011. 574-577.

SAMPATH, S., SPRENKLE, S., GIBSON, E., POLLOCK, L. & GREENWALD, A. S. 2007. Applying concept analysis to user-session-based testing of web applications. *IEEE Transactions on Software Engineering,* 33**,** 643-658.

SHIRAZ, M., GANI, A., SHAMIM, A., KHAN, S. & AHMAD, R. W. 2015. Energy efficient computational offloading framework for mobile cloud computing. *Journal of Grid Computing,* 13**,** 1-18.

SPRENKLE, S., SAMPATH, S., GIBSON, E., POLLOCK, L. & SOUTER, A. An empirical comparison of test suite reduction techniques for user-session-based testing of web applications. Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005. 587-596.

STOBER, T. & HANSMANN, U. 2010. *Agile software development-best practices for large software development projects*, Springer-Verlag Berlin Heidelberg.

TASSEY, G. 2002. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project.*

WANG, S., ALI, S. & GOTLIEB, A. 2015. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software,* 103**,** 370-391.

WILLIAMS, H. P. 2013. *Model building in mathematical programming*, John Wiley & Sons.

WOO, G., CHAE, H. S. & JANG, H. 2007. An intermediate representation approach to reducing test suites for retargeted compilers. *In:* ABDENNADHER, N. & KORDON, F. (eds.) *Reliable Software Technologies–Ada Europe 2007.* Springer Berlin Heidelberg.

XIE, T., MARINOV, D. & NOTKIN, D. Rostra: A framework for detecting redundant object-oriented unit tests. Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE'04), 2004. 196-205.

XIE, T., ZHAO, J., MARINOV, D. & NOTKIN, D. Detecting redundant unit tests for AspectJ programs. Proceedings of 17th IEEE International Symposium on Software Reliability Engineering (ISSRE'06) 2006. 179-190.

YOO, S. & HARMAN, M. Pareto efficient multi-objective test case selection. Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'07), 2007. ACM, 140-150.

YOO, S. & HARMAN, M. 2010. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software,* 83**,** 689-701.

YOO, S. & HARMAN, M. 2012. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability,* 22**,** 67-120.

ZHANG, D. & ADIPAT, B. 2005. Challenges, methodologies, and issues in the usability testing of mobile applications. *International Journal of Human-Computer Interaction*, 18**,** 293-308.

ZHANG, L., ZHOU, J., HAO, D., ZHANG, L. & MEI, H. Jtop: Managing JUnit test cases in absence of coverage information. Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'09), 2009. IEEE Computer Society, 677-679.

ZHANG, X., GU, Q., CHEN, X., QI, J. & CHEN, D. A study of relative redundancy in test-suite reduction while retaining or improving fault-localization effectiveness. Proceedings of the ACM Symposium on Applied Computing (SAC'10), 2010. Sierre, Switzerland: ACM, 2229-2236.