IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 16, 2019, accepted February 6, 2019, date of publication February 14, 2019, date of current version March 1, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2899471*

# An Insight Into Test Case Optimization: Ideas and Trends With Future Perspectives

**NEHA GUPTA[1], ARUN SHARMA[1], AND MANOJ KUMAR PACHARIYA[2]**
[1]Department of Information Technology, Indira Gandhi Delhi Technical University for Women, New Delhi 110006, India
[2]Department of Computer Science and Applications, Makhanlal Chaturvedi National University of Journalism and Communication, Bhopal 462011, India

Corresponding author: Neha Gupta (nehagupta_4321@yahoo.co.in)

**ABSTRACT** Software engineering comprises several disciplines. Testing, the subject of this paper, is an important phase which is still largely unpredictable, expensive, and ad hoc. Software testing has already been framed as an optimization problem, i.e., to solve problems in software testing various optimization techniques can be used. To adequately test the software, various rules called test adequacy criteria are used. In this paper, a review of optimization techniques used in domains, test case generation, selection, minimization, and prioritization of testing, has been presented. The review is conducted by selecting quality literature from journals, conferences, workshops, and symposium. The state-of-the-art issues in software testing have been addressed by analyzing the reviewed literature on the basis of the domain of testing, test adequacy criteria, and optimization technique used. After analysis of the considered literature, authors were able to conclude the limitations of existing work, list of traditional and recently proposed adequacy criteria as well as combinations of adequacy criteria for multi-objective optimization and optimization techniques which may be used in the optimization of software testing.

**INDEX TERMS** Optimization techniques, test case generation, selection, minimization, prioritization, multi-objective optimization.

## I. INTRODUCTION

Field of software testing is a significant phase in software life cycle process which ensures software quality by testing a program with intention of finding software bugs. Unfortunately, the problem of finding all faults or proving their absence in a program is essentially unsolvable because input space is vast and only a few test inputs can be sampled. Beizer [1] stated that 50% of the total software project's effort is used for testing. Testing is therefore always a trade-off between the cost for further testing and undiscovered faults in a program.

To reduce cost, time and effort for developing software, optimization techniques can be used. The use of optimization algorithms is not new to the world of software. Work on optimization for testing was started by Miller and Spooner [2] in 1967. They generated test data by using various optimization techniques. Xanthakis *et al.* [3] applied a meta-heuristic algorithm for test data generation. Optimization techniques can be of various types such as conventional techniques (combinatorial optimization, greedy, linear programming); intelligent techniques (dynamic programming, evolutionary and stochastic algorithms); and hybrid techniques. For optimization, various techniques such as local search, hill climbing, fuzzy logic, machine learning, simulated annealing, genetic programming, evolutionary and meta-heuristic algorithms can be used. Hybrid algorithms can also be used which combines good features of two or more algorithms and may solve the same problem in a better optimized way. A lot of hybrid algorithms have been proposed such as PSO-LA [44], ACO and chaos optimization algorithm [45], CS-TS algorithm [46], cuckoo and genetic algorithm [47], PSO-Bat algorithm [57], tabu-firefly hybrid [58], firefly-harmony algorithm [60], bat-harmony algorithm [61] etc. which have been used in various phases of software engineering or some other optimization problem. As testing is diverse and complex, it has been divided into distinct but related topics, such as test case generation, selection, minimization, and prioritization.

It requires various input parameters such as test criteria specifications, source code and definitions of data structure for generating tests. It's not an easy task to generate realistic test cases because of the complexity of the input parameters.

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

Thus, optimization techniques can be used at this level also to improve test data generation. A lot of work [7]–[13] etc. has already been done in this direction.

Test execution needs lot of time and resources. Eliminating redundant or unnecessary tests by minimization and selection of test suites can save both time and resources. In test suite minimization redundant tests are eliminated whereas in selection subset of tests are identified which is required to re-test the software alterations. Further optimization techniques have been used in research works such as [14]–[18] for test case selection and [23]–[28] for test suite minimization.

Next is test case prioritization in which tests are scheduled in an order for execution to increase the prompt fault detection, which will make debugging of the software faster. Optimization work has been done in test case prioritization also, such as Ahmed *et al.* [31]; Sun *et al.* [32]; Li *et al.* [33]; Joseph and Radhamani [34]; Tyagi and Malhotra [35]; Epitropakis *et al.* [36].

For optimization in software testing, another important aspect is selecting the adequacy criterion based on which optimization has to be done. Adequacy criteria are the test obligations or rules applied to test suites to determine whether all software bugs have been treated or not. Young [40] had stated, ''A test suite adheres to an adequacy criterion when all the test cases in it succeed or pass''. Adequacy criteria may be of various types such as control flow, data flow, and fault-based adequacy criteria. Data-flow based includes all definitions criterion and all uses criterion. Cyclomatic number criterion, statement, branch, path, loop and relational operator coverage are included in control flow-based criteria. Fault-based criterion includes mutant coverage or mutant killing score.

Selecting adequacy criteria in testing is important because even after testing, the extent to which errors remain in software depends upon criteria used to test. According to Young [40], for distinguishing adequacy criteria, empirical approach and analytical approach can be used. In an empirical approach, a controlled study is done which determines the effectiveness of testing methods on basis of kind of software being tested and kind of organization in which software is developed. The analytical approach defines one adequacy criterion stronger than another and subsumes relationship between adequacy criteria is used. For e.g. branch coverage criterion subsumes statement coverage as every test suite sustaining branch coverage should also sustain statement coverage. Weiss [43] has compared criteria on the bases of satisfiability, correctness, and cost. For him there are only two essential bases for comparing criteria, first is their efficiency at exposing errors, and second is their cost of use. Thus, selecting the correct adequacy criteria for optimization is important.

Aim of this work is to present a detailed and exhaustive review on optimization techniques in the field of software testing. Focus should be on current state and future aspects; thus, literature is considered from the year 2007-2018 as lots of surveys [62]–[68] in various domains of software testing are available for the period even before 2007. Instead of considering a specific domain of testing and optimization technique, all domains i.e. test case generation, selection, minimization, and prioritization and all optimization techniques are considered. Optimization work in which multi-criteria or hybrid optimization technique is used, are considered for study as software testing is based on many factors such as code coverage, requirement coverage etc. Thus, considering one criterion would not be enough and hybrid techniques are considered because they have good features of many techniques.

*Research Questions Addressed in the Study:*

- **Research Question 1:** What are the traditional adequacy criteria and their combinations that are widely used in literature and what are the recent new criteria that can be used in combination with other criteria?
- **Research Question 2:** What are the traditional optimization techniques which have been widely used in literature and new techniques that have not been used in all the domains of software testing i.e. test case generation, selection, minimization and prioritization?
- **Research Question 3:** What are the various gaps in the reviewed literature?

Further, paper consists of the following Sections: Section 2 presents the research methodology. Section 3 presents a literature review of previous works on successful application of optimization techniques in the four domains of testing: test case generation, selection, minimization and prioritization. The analysis of existing work, as well as answers to the considered research question 1 and 2 are presented in Section 4. In Section 5, research question 3 is answered in the form of challenges and opportunities. The conclusion is in Section 6.

## II. RESEARCH METHODOLOGY

A systematic review is a rigorous procedure in which identification, selection of appropriate research and analysis of data are needed to be done. Review process employed in the present work for literature selection is represented in Figure 1. For selecting suitable research work, following keywords are used: ''optimization techniques'', ''test case generation'', ''test case selection'', ''test case minimization and prioritization'' and ''test adequacy criteria''. To answer the considered research questions, available research work is divided into three categories: optimization technique used in software testing; software test adequacy criteria and optimization techniques not widely used in software testing. Research work belonging to these categories serves as the database for conducting review. First category focuses on work where optimization technique is used, in considered domains of testing. Second category focus on finding non-traditional adequacy criteria and their related works. Third category focuses on finding optimization techniques that have not been used or very less used and has a future scope in software testing. Figure 1 represents how from these databases;

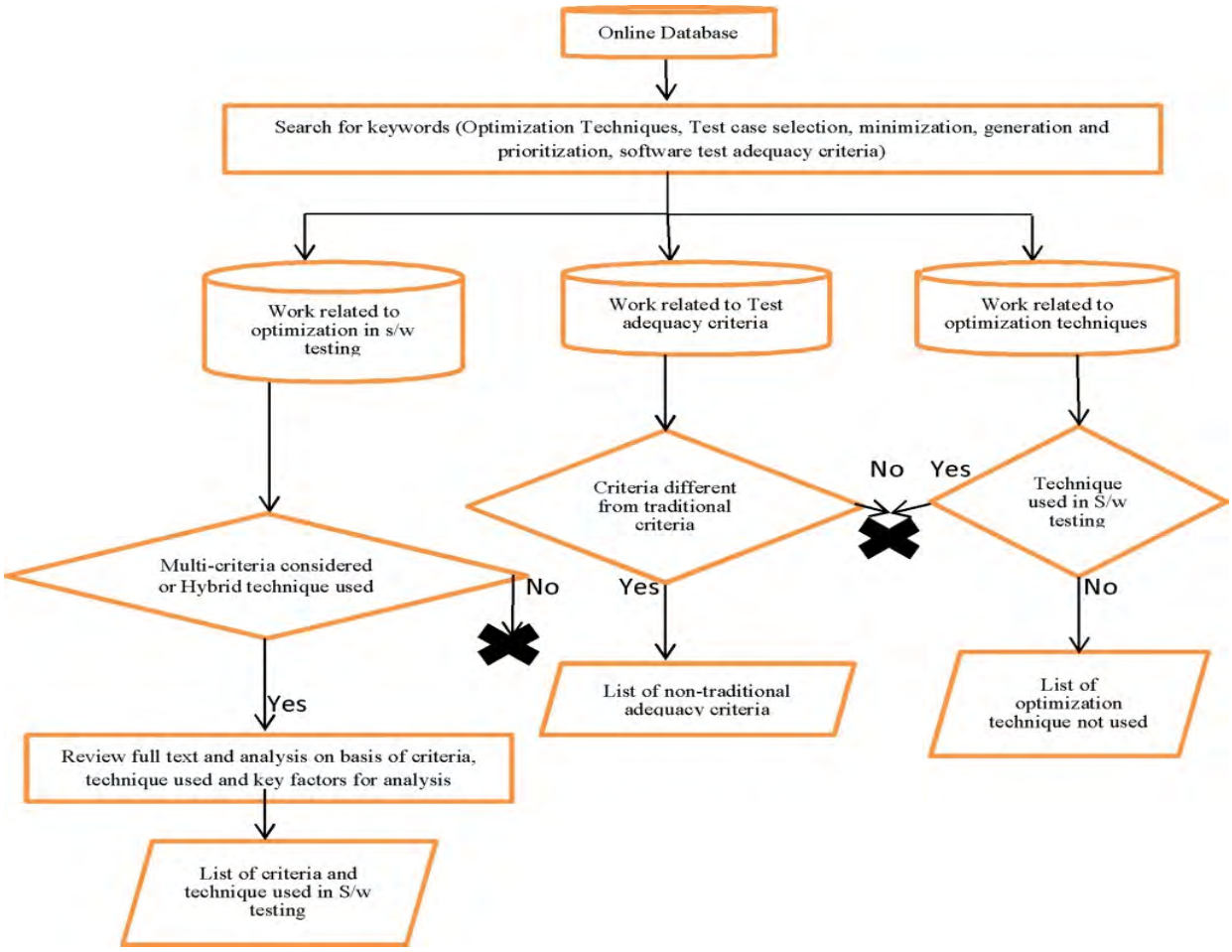



**FIGURE 1.** Systematic literature review.

**TABLE 1.** Search strings used to gather literature.

| Search String | Search Result | Included for review | Excluded |
|---|---|---|---|
| Optimization Technique AND (Test case generation OR Selection OR Minimization OR Prioritization) | 232 | 36 | 196 |
| Software Test adequacy criteria | 45 | 13 | 32 |
| Optimization Techniques | 32 | 7 | 25 |

the desired result has been achieved. For including and excluding research work from these categories, some filter conditions are used which have been discussed in following text. Table 1 represents the search string used for creating database as per the categories discussed above, no. of search result, no. of included and excluded research work after applying inclusion and exclusion criteria.

To limit the selection of work those are relevant to the considered research questions, the following inclusion and exclusion criteria are used.

**Filter conditions for study related to optimization in software testing.**

- Studies from the year 2007-2018 are included.
- Studies published in conference, symposium, workshops, notes and journals mentioned in Table 2 are included.
- Studies in which multi-criteria optimization is done or a hybrid technique is used for optimization are included.
- Studies with duplicate research work in different resources are removed.
  - Studies in which a proposal is given but not implemented are removed.
- Studies in which a proposal is not validated using examples or dataset are removed.

**Filter conditions for non-traditional adequacy criteria**

- Studies, where proposed adequacy criteria are new in concept, different from traditional adequacy criteria[1] and are scarcely used in literature as well as being possibly good criteria to use are considered.

[1]Traditional Adequacy criteria are criteria which have been commonly used in literature such as statement, branch, path, requirement coverage, execution time, fault detection rate etc.

**TABLE 2.** Publication platforms.

| | |
|---|---|
| Journals | IET Software;<br>Transactions on Software Engineering Methodology (TOSEM)-ACM;<br>Swarm and Evolutionary Computation; Journal of Systems and Software (JSS)-Elsevier;<br>Software: Practice and Experience (SPE); Software Testing, Verification and Reliability (STVR)-Wiley;<br>Transactions on Software Engineering-IEEE;<br>Empirical Software Engineering; Software Quality Journal-Springer |
| Conferences | International Conference on Software Testing, Verification, and Validation-IEEE (ICST); International Conference on Software Quality, Reliability and Security-IEEE (QRS); International Conference on Computer Theory and Application (ICCTA);<br>International Conference on Tools with Artificial Intelligence-IEEE (ICTAI);<br>International Conference on the Quality of Software Architectures-Springer (QOSA);<br>International Conference on Testing Software and Systems-Springer (ICTSS);<br>Conference on Genetic and Evolutionary Computation-ACM (GECCO) |
| Workshops | Workshop on Software Quality Assurance-IEEE (SQA);<br>Workshop on Automation of Software Testing-IEEE/ACM (AST);<br>Software Testing, Verification and Validation Workshops (ICSTW) |
| Symposiums | International Symposium on Search Based Software Engineering-Springer (SSBSE);<br>Symposium on Software Testing and Analysis-ACM (ISSTA);<br>International Symposium on Software Reliability Engineering- IEEE (ISSRE) |
| Notes | SIGSOFT Software Engineering Notes (ACM) |

**Filter conditions for optimization techniques not used in software testing**

- Studies, where proposed optimization technique is not used in software testing but gives better results in terms of convergence and accuracy compared to existing traditional techniques[2] are considered.

From a large pool of studies, only 36 studies for optimization in software testing are accepted because in them multi-objective optimization or hybrid technique is used. Analysis of each research work in the respective domain of testing has been done to answer the research questions addressed in the study. Selected literature is also analyzed and compared based on following key aspects: Method of problem formulation, type of optimization technique, level of validation, data source usage and feasibility in industrial or research projects. These aspects are included because while review, we found them to be important for an optimization work in software testing. Each aspect has some value and a weight assigned to it. Final marks are calculated by adding value of all the aspects. Research work which has the highest score is good in all terms. Analysis of recently proposed criteria is also done which is represented in tabular form in Section 4.

[2]Traditional techniques are optimization techniques which have been commonly used in literature such as genetic algorithm, PSO, ACO, Firefly, and NSGA etc.

**Key Aspects of evaluation:**

- **Method of formulating the software testing problem:** A problem can be formulated in two ways: single and multi-objective. A multi-objective approach is better as software testing optimization is based on many adequacy criteria such as code coverage, requirement coverage etc. Thus, multi-objective approach is given more value. The values assigned are given in Table 3.

**TABLE 3.** Problem Formulation Paradigms.

| FORMULATION | VALUE |
|---|---|
| Single objective approach | 1 |
| Multi-objective approach | 2 |

- **Type of optimization technique:** Various types of techniques can be used; conventional techniques are non-intelligent techniques such as linear programming, greedy etc. Intelligent techniques are dynamic programming, evolutionary and stochastic algorithms. Hybrid techniques can be a combination of any two types of technique. A technique which is the combination of both types will have good features of both. Thus, it is given maximum value of 3. The values assigned are given in Table 4.

**TABLE 4.** Types of Techniques for Test Case Optimization.

| OPTIMIZATION TECHNIQUE | VALUE |
|---|---|
| Conventional technique | 1 |
| Intelligent technique | 2 |
| Hybrid technique | 3 |

- **Validation:** Proposed test case optimization technique needs to be validated by applying it to programs. So, that the pros and cons of the proposed techniques can be known. Thus, validation is important, and if it is done then value of 1 is assigned. Values assigned are given in Table 5.

**TABLE 5.** Level of Validation of Test Case Optimization Techniques.

| VALIDATION DONE | VALUE |
|---|---|
| No | 0 |
| Yes | 1 |

- **Data source usage:** Data used for validation can be from a benchmark repository which consists of complex and large sized programs used for industrial purpose or from an academic source which consists of small and student made programs. Testing on complex and large programs is more meaningful. So, benchmark programs are given maximum value. Values assigned are given it Table 6.
- **Feasibility:** Is there a scope of using this work in industry or research projects. Values assigned are given in Table 7.

**TABLE 6.** Types of Data Source Used for Validation.

| DATA SOURCE USAGE | VALUE |
|---|---|
| Academic Programs | 1 |
| Benchmark repository | 2 |
| Both academic and benchmarked programs | 3 |

**TABLE 7.** Space for Future Research.

| FEASIBILITY | VALUE |
|---|---|
| Less scope | 1 |
| Possibility of extension of method | 2 |

## III. LITERATURE REVIEW

This section consists of study of various optimization techniques in the field of test case generation, selection, minimization, and prioritization as well as the recently proposed adequacy criteria.

### A. TEST CASE GENERATION

For the generation of test data, multi-objective search algorithm from pareto optimal and weighted fitness approach was proposed by Lakhotia *et al.* [4]. In pareto GA, a set of front lines is created by NSGA-II algorithm, which contains only non-dominating solutions. Performance of random search, weighted GA, and pareto GA is tested on five case studies. Results depict that weighted GA is best for maximum of the cases among all search methods.

Harman *et al.* [5] have used memory based, greedy based and CDG based test data reduction techniques with genetic algorithm to optimize oracle cost and branch coverage with a reduced number of tests. Check isbn, clip to circle and triangle are the programs on which study was conducted. Results show that they were able to reduce test cases and cyclomatic complexity without affecting branch coverage by all the techniques.

The genetic algorithm is among widely used algorithms. Rathore *et al.* [6] have modified this method by using tabu search in mutation step. The proposed algorithm is more effective as all nodes of the triangle classifier program are covered with a smaller number of test cases. Also, it covers node 8 which was uncovered in GA based algorithm.

Automated test data can be generated by path-oriented techniques which can be static and dynamic. Various approaches are there in dynamic techniques for optimizing testing problems. Srivastava *et al.* [7] used hybrid of cuckoo and tabu search for generating tests. Cuckoo search generates the solution and tabu list stores test cases and its information. This approach is used to generate test cases for triangle classifier problem. The output is generated test cases with nodes covered by them and time required for execution. Results depict that the proposed method uses only 467 iterations compared to 24233 iterations by normal cuckoo and tabu search.

Srivastava *et al.* [8] have modified firefly algorithm with guidance matrix for generating optimal test paths for testing. They have implemented the proposed method in their software testing tool, OFTSG, which gives data flow diagram and state diagram of a test case as result. They have compared the proposed algorithm with ACO and results show that with modified FA there is no recurrence in traversed path whereas by ACO repetition is there. Proposed technique converges faster than ACO.

Fraser and Arcuri [9] have developed whole test suites with smaller size using their EVOSUITE tool and compared with considering one goal at a time. It has proved to be effective compared to approach targeting single branches, with around 188 times the branch coverage after evaluating on a total of 1,741 classes. Test suites were smaller up to 62 percent.

Mutation based test generation is expensive as lot of mutants are involved. Fraser and Arcuri [10] have generated test cases using genetic algorithm based on mutation testing and branch coverage. They have removed equivalent mutants and optimized test suites to kill higher number of mutants. Results show that their approach helped in generating mutants-based test cases with less effort.

A black box testing approach for producing string test cases has been proposed by Shahbazi and Miller [11]. They have used GA and NSGA-II algorithms for generating string test cases based on test diversity and string length distribution. Results show proposed technique with considered string distance functions and optimization algorithms performs better than randomly generated test cases.

Existing approaches either consider one objective or combine all objectives into a single fitness function for test case generation, for e.g. whole test suite generation. Panichella *et al.* [12] have considered it as a multi-objective problem in perspective of coverage of mutants, statement and branches using Dynamic Many-Objective Sorting algorithm (DynaMOSA). It was evaluated for 346 Java classes and compared with the whole-suite approach and MOSA. The outcome shows that DynaMOSA outdoes WSA in 28% and 27% of the classes aimed at branch and mutation coverage respectively. DynaMOSA also outdoes MOSA.

Table 8 includes detail of literature for test case generation with adequacy criteria, optimization technique used, results and remarks.

### B. TEST CASE SELECTION

A Pareto efficient approach has been proposed by Shin and Harman [14] which takes multiple objectives by considering formulation which combines two criteria: coverage and cost and three criteria: coverage, cost and fault history for test case selection. These instance problems are solved by re-formulated greedy algorithm, NSGA-II and vNSGA-II. The study was done on print_tokens, print_tokens2, schedule, schedule2, and space. Mixed results are there. VNSGA-II had shown best performance for siemens programs, whereas the weighted-sum additional greedy algorithm had given good results with space.

Size-constrained regression test selection problem has been solved by Mirarab *et al.* [15]. They have considered

**TABLE 8.** Summary of Literature in domain of test case generation.

| PAPER | TECHNIQUES USED | CRITERIA COVERED | RESULT | REMARKS |
|---|---|---|---|---|
| [4] | Pareto GA<br>Weighted GA | Branch coverage<br>Dynamic memory Consumption | Weighted GA is best among Pareto, weighted GA and random techniques. | Weight balancing between the different objectives is obtained by 'trial and error'. Well-defined technique may be used. |
| [5] | Memory, greedy and CDG-Based reduction with genetic algorithm | Oracle cost<br>Branch coverage | Reduced number of test cases without affecting branch coverage by all the techniques. | It may be possible to achieve better result by using model like 'crowding distance' in the NSGA–II algorithm for extending the CDG–based approach. |
| [6] | GA and Tabu search | Path coverage | Suggested technique is more effective as a smaller number of tests covers all the nodes and it covers node 8 also which is not covered by GA algorithm. | Risk of stucking in local optima may be excluded by using random backtracking. |
| [7] | Cuckoo Search and Tabu Search (CSTS) | Path Coverage | CSTS performs better than GA, ACO, ABCs, for node coverage, number of Iterations and parameters. Iteration No. reduced to 467 from 24233. | CSTS is applied on triangle classifier problem. It may be applied on larger commercial problem. Though the strategy is laid out for string input, it still needs to be explored through experimental comparisons for string inputs. |
| [8] | Firefly Algorithm | Path coverage | FA converges faster than ACO and can remove redundant paths in a better way. | A very small graph is taken for study with a smaller number of nodes. Complex examples with a greater number of nodes may be taken. |
| [9] | Genetic Algorithm | Multiple Branch | Up to 188 times more branch coverage compared to single branch approach. 62 percent smaller test suites. | Though it's a multi criteria approach but whole-suite is generated by single objective formulation. |
| [10] | Genetic Algorithm | Branch coverage<br>Mutation coverage | Achieved high branch coverage and mutation score with less efforts to be done for mutation testing and thus improved fault detection capability | Equivalent mutants are removed to reduce time for mutation testing, but remaining mutants still need to be checked for equivalence. |
| [11] | NSGA-II<br>GA | Test diversity<br>String length distribution | Produced string test cases using GA and NSGA-II performs better in terms fault detection compared to test cases produced by random technique. | Other string distance function may be tried to improve used fitness function |
| [12] | Dynamic Many-Objective Sorting Algorithm | Statement coverage<br>Branch coverage<br>Strong mutation Coverage | DynaMOSA outperforms both MOSA and WSA in 28% and 27% of the classes for branch and mutation coverage respectively. | They are the first one to consider coverage only as a many-objective problem. |

two criteria for test suite selection and prioritization based on fault prediction of test cases: minimum sum of coverage (max-min criterion or $D_{min}$) and sum of coverage $D_{sum}$. Both criteria need to be maximized. To solve the problem, they have used linear programming and greedy approach. They have considered coverage data for 7 test cases and 3 software elements. $D_{sum}$ is maximum for selected case T1, T3, T5, T7. But $D_{min}$ can be increased if T5 is replaced by T4. They have given importance to both criteria and there is a twofold increase in $D_{min}$ for a slight decrease in $D_{Sum}$. Thus, these two criteria are optimized together to the best.

Lucia *et al.* [16] have optimized statement coverage and execution time for test case selection. They have aggregated population diversity in the attained Pareto fronts of NSGA-II. The study was conducted on four programs from the SIR and results show that enhanced NSGA-II has improved convergence speed and non-dominated solutions with higher diversity were obtained.

BMOPSOCDRLS and BMOPSO-CDRLS2 have been proposed by Souza *et al.* [17] for multi-objective test case selection. These algorithms are developed by combining BMOPSO-CDR with forward selection, backward elimination, and the 1-opt algorithm. These mechanisms use branch coverage and cost, for structural test case selection. For the experiment, randomly four test suits of the space program from the software-artifact infrastructure repository (SIR) were used and evaluated using four matrices: hyper-volume, generational distance, inverted generational distance, and coverage. They have compared the proposed algorithms with BMOPSO-CDR and NSGA-II. Results show that BMOPSO-CDRLS and BMOPSO-CDRLS2 are better.

Zeeshan and Ahsan [18] have optimized regression test suites using fuzzy logic with the following objectives: requirement coverage, execution time, fault detection rate, requirement failure impact. Results show that there is a 50 % reduction in execution time and size of test suites by using the proposed fuzzy technique.

**TABLE 9.** Summary of Research paper in domain of test case selection.

| PAPER | TECHNIQUES USED | CRITERIA COVERED | RESULT | REMARKS |
|---|---|---|---|---|
| [14] | Weighted additional greedy algorithm<br>NSGA-II<br>vNSGA-II | Coverage<br>Cost<br>Fault history | vNSGA-II has given good results for Siemens programs, whereas the weighted-sum additional greedy algorithm gives best result with space. | Authors are able to give empirical results which disclose that greedy algorithms accomplish sound for single objective formulation and are not always effective in the multi-objective problem. |
| [15] | Integer Linear Programming and Greedy approach | Minimum sum of coverage<br>Sum of coverage | Proposed approach is consistent and outperforms other techniques | Proposed a new criterion: maximizing the minimum sum of coverage. D Function can be used for other coverage also such as requirement coverage. |
| [16] | NSGA-II with Diversity | Statement coverage<br>Execution time | Convergence speed of the genetic algorithm was improved and non-dominated solutions with higher diversity are obtained | |
| [17] | BMOPSOCDRLS andBMOPSOCDRLS2 | Branch coverage Minimize cost | BMOPSO-CDRLS and BMOPSO-CDRLS2are better than BMOPSO-CDR and NSGA-II | Only two criteria are used. |
| [18] | Fuzzy Logic | Requirement coverage, and Execution time,<br>fault detection rate | Fuzzy Logic reduced execution time and size of test suites up to 50%. | Only two case studies considered.<br>Continues... |
| [19] | NSGA-II | Code coverage<br>Test diversity<br>Test execution time | 3 objective approach is more effective, sometimes up to 25% than the bi-objective alternative | More empirical studies are needed in field of test diversification as it is very new. Study may be extended by analyzing additional distance functions. |
| [20] | Hybrid of fuzzy entropy and Ant colony optimization | Statement, Branch coverage<br>Execution time | Ambiguous, obsolete test cases have been removed and test case reduced from approximately 2400 to 800. | No comparison is done with classification techniques such as K-mean classification, SVM etc. and optimization algorithm such as Bat, PSO. |
| [21] | Alternating Variable Method<br>SPEA2<br>Random Search | Time Difference<br>Mean Priority, Mean Probability and Mean Consequence | SPEA 2 outdoes all considered algorithms in terms of mean priority, probability and consequences | Evaluation metrics such as epsilon apart from fitness value and hyper volume may be tried. |
| [22] | ACO<br>Hybrid PSO | Fault detection<br>Execution time | Hybrid PSO performs better that ACO as it detects 84.2 % of faults only with 0.7% of test cases. | Evaluated only on one benchmark object flex |

A three-objective optimization approach for selection of test cases using NSGA-II has been proposed by Mondal *et al.* [19]. Their study shows that neither of the two techniques is dominating each other completely, rather they can be complementary. Test cases given by coverage-based and diversity-based approaches have little overlap thus they have used NSGA-II for solving a three-objective problem in which code coverage and diversity are maximized and execution time is minimized. They have conducted the study on five real-world SUTs (Apache Ant: 3 versions, JTopas: 1 version, and Space: 1 version). Results show that this approach is effective up to 25% than the bi-objective alternative and costs less.

Hybrid of fuzzy entropy and ant colony optimization has been used by Kumar *et al.* [20] for classification and selection of test cases. It consists of three stages. First multifaceted test cases are filtered and classified using fuzzy logic. Then combination of fuzzy entropy and backward search strategy is used to filter test cases. In third stage test cases are selected from the output of second stage using ant colony optimization with a forward search strategy. It's applied on test suite for Print_Tokens and Print_Tokens 2 code from Software Infrastructure Repository. The results show that the performance of algorithm increases as the stages progress.

A test case selection approach based on cost measure and effectiveness measure has been proposed by Pradhan *et al.* [21]. For optimization they have used Alternating Variable Method, SPEA2 and Random Search. Results are in favor of SPEA2 algorithm.

Agrawal and Kaur [22] have used ACO and Hybrid PSO for test case selection based on fault coverage and execution time. Results show that hybrid PSO performs better compared to ACO, as it is able to detect 84.2 % of faults with only 0.7 % of tests whereas ACO required 11% of tests.

In Table 9, details of literature in the field of test case selection have been provided in terms of adequacy criteria, optimization technique, its results and remarks.

### C. TEST SUITE MINIMIZATION

Regression testing is a complicated process which includes multiple constraint and criteria to be tested. Yoo and Harman [23] produced better results in test suite

minimization in regression testing by a hybrid multi-objective genetic and greedy approach. Test suite minimization should be done with maximum fault detection and minimum cost. In this work five subjects flex, grep, gzipped and space are considered. Code coverage tells how adequate tests are and execution time tells about the cost for testing. This approach has helped in making a more effective testing decision.

Mala and Mohan [24] have incorporated two heuristics RemoveSharp and LocalOPt in hybrid genetic algorithm which has improved quality and reduced number of test cases. HGA is compared with GA and bacterial foraging algorithm. Test cases are reduced up to 80.6% by HGA.

In most of the work, code coverage is used as criteria for optimization. Functional requirement coverage is also an important criterion. So, Souza *et al.* [25] have proposed, a binary multi-objective PSO and BMOPSO-CDR which combines binary PSO with crowding distance and roulette wheel, to maximize requirement coverage and minimize cost. Proposed algorithms were implemented on test suites from mobile devices background. Results were evaluated with five metrics: hyper volume, spacing, maximum spread, coverage and coverage difference. BMOPSO-CDR was better compared to both BMOPSO and random algorithms.

Suri and Mangal [26] reduced the test cases suite by a hybrid of bee colony optimization and genetic algorithm. They presented a tool HBG_TCS that implements the proposed approach. Comparison of the approach is done with the ant colony optimization.

General purpose computing on graphical processing units is expensive to use in SBSE. Yoo *et al.* [27] are the first one to use GPGPU in SBSE; GPGPU has an advanced architecture with the feature of parallelism which provides a good platform for improving SBSE scalability. They have presented GPGPU-based test suite minimization technique based on requirement coverage and computation cost. The approach was evaluated on NSGA-II, SPEA2 and two archive algorithm. GPGPU-based optimization can attain a speed of around 25.09 times in comparison to CPU executed algorithm.

A combinatorial and cuckoo search-based optimization technique has been proposed by Ahmed *et al.* [28] to generate tests on basis of functionality coverage. Their proposed technique performs better than TS, GA and PSO algorithms.

Zheng *et al.* [29] have used various optimization techniques such as greedy, NSGA-II and decomposition based multi-objective evolutionary algorithm (MOEA/D) for minimizing test suites according to execution cost, statement, branch and MC/DC coverage. Results show MOEA/D is the most robust algorithm among all. Ahmed [30] presented a technique using combinatorial optimization and CS to reduce test cases in configuration-aware structural testing. First, optimized test suite is generated by using combinatorial optimization and then filtered using mutation testing. The efficacy of technique is measured by doing an empirical study on a software system.

### D. TEST SUITE PRIORITIZATION

An approach for prioritizing tests has been presented by Ahmed *et al.* [31]. The approach uses genetic algorithm to optimize multi-criteria fitness function consisting of condition, multiple condition, and statement coverage. It uses multiple control flow coverage metrics. The average percentage of fault detected metric was used to compare the proposed technique with other related work. Results of the proposed technique are close to previous work.

Sun *et al.* [32] have prioritized test cases for GUI applications. They have optimized fault detection capability using event and statement coverage using additional greedy algorithm. Proposed strategy is better in terms of performance compared to single-objective strategies.

GPU-based parallel MOEAs are proposed by Li *et al.* [33] to increase execution efficiency of test case prioritization. Parallel crossover calculation outlines centered on ordinal and sequential illustrations have also been used. The study was done on eight benchmarks which show a speed-up of 120 times.

Technique formed by the integration of artificial bee colony, fuzzy C-Means, and PSO, is proposed by Joseph and Radhamani [34]. ABC-FCM iterates till the number of paths and faults covered is maximized. Then PSO is given individuals obtained from the ABC-FCM as the input for further optimization. Proposed method is better as it improved quality and reduced complexity of test cases compared with other optimization techniques.

A 3-phase approach has been proposed by Tyagi and Malhotra [35] for prioritizing tests. In first phase, they have removed redundant test cases. Then reduced test set covering all faults is formed using MOPSO. In the third phase, a ratio of fault coverage to the execution time is calculated to prioritize the test cases. The MOPSO approach outdid no reverse and random ordering techniques in terms of average percentage of fault detected.

Epitropakis *et al.* [36] have proposed prioritization technique considering coverage of code and changed code. For evaluation, they have used NSGA-II, two archive evolutionary algorithm, and additional greedy algorithm. Five subjects taken from the SIR are flex, grep, gzip, make, and the sixth subject program is MySQL. They have also introduced a coverage compaction algorithm that reduces coverage data size, and execution time of all the algorithms used.

Marchetto *et al.* [37] have used NSGA-II for maximizing the number of discovered faults by optimizing development cost, code, and requirement coverage. It consists of: recovering traceability links, computing metrics and estimating maintainability indexes. MOTCP+ is the proposed prototype which has been compared with random prioritization, code coverage, and additional code coverage-based prioritization on 21 java application. Based on median of APFD values, MOTCP+ outperforms mostly all techniques. APFD in each MOTCP+, AddCodeCov and NSGAIIdim2 is 80%, 66.6% and 62% respective.

A resource aware test prioritization technique has been proposed by Wang *et al.* [38] for software developed for Videoconferencing Systems at Cisco. They have used weighted GA, NSGA-II, SPEA-II, PSO and hybrid of GA and differential evolution for optimizing total execution time, test resource usage and fault detection capability. Random weighted genetic algorithm is best among all algorithms for optimizing all criteria together.

A multi-objective approach named graphite for optimizing Code coverage, Test dissimilarity and execution time has been proposed by Azizi and Do [39]. Graphite performs better for considered time constraint (25%, 50%, and 75% in terms of APFD compared to other greedy based techniques.

Details of literature for test case minimization and prioritization in terms of adequacy criteria, optimization techniques, result and a remark has been provided in Table 10.

### E. RECENTLY PROPOSED ADEQUACY CRITERIA

- **Behavioral adequacy:** Fraser and Walkinshaw [41] have proposed behavioral adequacy criterion which considers, ''test set is considered suitable if its tests cover all characteristics of computation done by the program''. It states that the program's input and output have a relationship between them which is considered as the behavior of a program. Thus, from a program's behavior according to the test sets, it may be possible to deduce a model which will help in guessing the outputs for inputs that haven't been encountered.
- **Fuzzy Entropy value of test cases:** It's been proposed by Kumar *et al.* [20] which measures ambiguousness of test cases. It can be applied in the fitness, classification, and selection of test cases to measure the ambiguity. FEFI system based on statement, branch coverage, fault detection capability and execution time has been built which measures ambiguity in the suitability of test case using fuzzy entropy.
- **Distinguishing mutation adequacy:** Distinguishing mutation adequacy criterion has been proposed by Shin *et al.* [42] which contain the idea of diversity in mutant killing, i.e. mutants can be differentiated from each other by the set of tests that kill them. Two mutant's mx and my generated from program $p_o$ are distinguished by a test t only if they hold the following condition:

$$d(t, p_o, m_x) \neq d(t, p_o, m_y), \text{ for a differentiator d}$$

- **Maximizing the minimum sum of coverage:** This criterion is proposed by Mirarab *et al.* [15] with the aim of maximizing the minimum coverage through all software elements. $D_{min}$ is the function proposed by them with the following equation.

$$D_{min}(S) = \min_{a_m \in A} W_m d_m(S) \quad (1)$$

where $d_m(S)$ denotes fault detection capability of S when applied to am and $w_m$ is used to assign weights to software elements

- **Operational coverage:** A profile-based testing criterion called operational coverage has been proposed by Miranda and Bertolino [75]. It identifies various entities such as branch, function etc. in code and then group them conferring to usage frequency in reference to the frequently used operations of software. Then, operational coverage is calculated by using formula:

$$operational\ coverage = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}.100\% \quad (2)$$

where, n is the number of important groups; $x_i$ denotes the rate of entities covered from group $i$. $w_i$ is weight assigned to group $i$.

## IV. ANALYSIS

This section consists of analysis of reviewed literature and answer the considered research question.

### A. ANALYSIS FOR TEST CASE GENERATION

Analysis of literature in test case generation in terms of aspects defined in Section 2 has been presented in Table 11. Paper having highest score indicates the more acceptability and suitability in terms of usage. After evaluation of work for test case generation, it may be concluded:

1) Though multi-criteria optimization is done in most of the research work, only Panichella *et al.* [12] have used multi-objective algorithm DynaMOSA for optimization.

2) Mostly structural coverage-based adequacy criteria branch and path coverage are considered for optimization. There is a lack of research work considering functional testing-based adequacy criterion for test case generation. Gay *et al.* [48] have also concentrated on the disadvantage of creating tests based on only coverage. They have evaluated the efficiency of test suites based on four coverage criteria (condition, decision, modified condition/decision and observable MC/DC). Counter example-based and random test generation approaches were used on seven case examples and concluded that using coverage criterion alone for fault finding is almost equivalent to producing random test suites of equal sizes. Thus, target should not be structural coverage. Instead, it should be used as a supplement with randomly generated test suites. With this approach, there is an increase of up to 13.5 percentage more fault detection.

3) It may also be concluded that half of the studies in test case generation considered academic programs as subjects.

### B. ANALYSIS FOR TEST CASE SELECTION

Literature in test case selection has been analyzed according to key aspects defined in Section 2 and the analysis is available in Table 12.

For test case selection, it may be concluded that:

(1) The most common techniques used for optimization are genetic and greedy algorithms with their variants.

(2) Adequacy criterion of almost all types: code coverage, requirement coverage and fault coverage have been used for optimization, but mutation-based testing is scarcely used in test case selection.

**TABLE 10. Summary of Research paper in domain of test case minimization and prioritization.**

| PAPER | TECHNIQUES USED | CRITERIA COVERED | RESULT | REMARKS |
|---|---|---|---|---|
| [23] | Hybrid of genetic algorithm and greedy approach | Code coverage<br>Fault coverage<br>Execution time | Approach helped in making more effective testing decision. | Greedy approach is used which is well suited for single objective problem whereas it's a multi objective problem. |
| [24] | Hybrid of GA and local search | Mutation score<br>Path coverage | Quality of the test cases is improved, and no. of test cases is reduced up to 80.6% | GA is widely used technique. We can improve some new algorithm with local search technique. |
| [25] | BMOPSO) and BMOPSO-CDR | Requirement Coverage<br>Execution Time | BMOPSO-CDR outperformed both BMOPSO and random algorithms for both suites | Source of dataset is not mentioned. |
| [26] | Hybrid on bee colony optimization and genetic algorithm | Fault coverage<br>Execution Time | Better results compared to ACO | Dataset is not reliable |
| [27] | GPGPU<br>NSGA-II<br>SPEA<br>Two Archive algorithm | Requirement coverage<br>Computation cost | GPGPU-based optimization achieved speed–up around 25.09X compared to CPU executed algorithms | GPGPU can be costly. |
| [28] | Combinatorial optimization and Cuckoo Search | Fault detection | No. of test suites is reduced with increased fault detection capability. | Only 1 program is considered for case study. Validation should be done on more case studies. |
| [29] | Greedy technique<br>NSGA-II<br>MOEA/D | Cost<br>Statement coverage<br>Branch coverage<br>MC/DC coverage | Results show that greedy technique is worst in performance. MOEA/D with fixed constant performed better than NSGA-II. | Constant c is assigned fixed value of 0.3. Results with other values need to be explored. |
| [30] | Combinatorial optimization and Cuckoo Search | Functionality coverage | Proposed approach works better than TS, PSO, and GA | Validated on a self-made application. Should be tried on benchmark program |
| **DOMAIN: TEST CASE PRIORITIZATION** | | | | |
| [31] | Genetic Algorithm | Condition Coverage,<br>Multiple Condition Coverage<br>Statement Coverage | The results of proposed technique are close to previous work | No improvements. |
| [32] | Additional Greedy Algorithm | Event coverage,<br>Statement coverage | Proposed strategy performs better than single-objective strategies | New technique may be used as greedy approach is mostly suitable for single objective approach whereas it is multi-objective problem. |
| [33] | GPU<br>NSGA-II | Statement Coverage<br>Execution time | Speedup of up to 120 times is achieved by using GPU. | Small real-world applications used for case study. Approach may be applied on larger real-world applications. |
| [34] | ABC<br>Fuzzy C-Mean<br>PSO | Faults coverage<br>Statement coverage | ABC-FCM provides better results in terms of statement and fault coverage compared with ACO, ABC and PSO. | Source of test data not mentioned properly |
| [35] | MOPSO | Fault coverage<br>Execution Time | MOPSO outdid ordering techniques in terms of APFD. | |
| [36] | NSGA-II,<br>TAEA,<br>Additional greedy algorithm | Average % of coverage achieved, % coverage of changed code, % of past fault coverage. | Coverage compaction algorithm has reduced coverage data size, and execution time of all the algorithms used. | Case study done only on 6 programs. |
| [37] | NSGA-II | Code coverage,<br>Additional code coverage, Requirement coverage and cost | In terms of APFD values MOTCP+ outperforms mostly all techniques as APFD in each MOTCP+, AddCodeCov and NSGA-II is 80%, 66.6% and 62% respectively. | For recovering traceability links between requirements and code, Vector space modeling technique can be considered for future work. |
| [38] | Weighted GA,<br>NSGA-II,<br>SPEA-II,<br>PSO<br>GA and differential evolution hybrid | Total execution time, test resource usage and fault detection capability. | Random weighted genetic algorithm performs best among considered algorithms. | Analysis done only on one case study. |
| [39] | Greedy graph-based technique | Code coverage, Test dissimilarity<br>Test execution time | Proposed technique graphite performs better in terms of APFD compared to other greedy based techniques for prioritization | Diversity can be measured with other distance functions such as Hamming and edit distance |

Continues...

(3) In most of the experiments, the tests are conducted on small scale software, which are mostly freely available or are from Software Infrastructure Repository.

### C. ANALYSIS FOR TEST CASE MINIMIZATION AND PRIORITIZATION

Table 13 and Table 14 include the value of aspects based on which papers have been evaluated for minimization and

**TABLE 11.** Final score calculation for literature in test case generation.

| Aspects | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] |
|---|---|---|---|---|---|---|---|---|---|
| Method of problem formulation | 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| Type of optimization technique | 2 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 3 |
| Validation | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Data source usage | 1 | 3 | 1 | 1 | 1 | 2 | 2 | 2 | 3 |
| Feasibility | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 1 | 2 |
| Total | 8 | 11 | 7 | 7 | 6 | 8 | 9 | 8 | 11 |

**TABLE 12.** Final score calculation for literature in test case selection.

| Aspects | [14] | [15] | [16] | [17] | [18] | [19] | [20] | [21] | [22] |
|---|---|---|---|---|---|---|---|---|---|
| Method of problem formulation | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Type of optimization technique | 3 | 3 | 2 | 3 | 2 | 2 | 3 | 3 | 3 |
| Validation | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Data source usage | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 2 |
| Feasibility | 1 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 1 |
| Total | 9 | 10 | 9 | 10 | 9 | 9 | 10 | 10 | 9 |

**TABLE 13.** Final score calculation for literature in test case minimization.

| Aspects | [23] | [24] | [25] | [26] | [27] | [28] | [29] | [30] |
|---|---|---|---|---|---|---|---|---|
| Method of problem formulation | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 1 |
| Type of optimization technique | 3 | 3 | 3 | 3 | 2 | 3 | 2 | 3 |
| Validation | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Data source usage | 2 | 3 | 1 | 1 | 2 | 1 | 2 | 1 |
| Feasibility | 2 | 2 | 1 | 1 | 2 | 2 | 2 | 2 |
| Total | 10 | 11 | 8 | 8 | 9 | 8 | 9 | 8 |

**TABLE 14.** Final score calculation for literature in test case prioritization.

| Aspects | [31] | [32] | [33] | [34] | [35] | [36] | [37] | [38] | [39] |
|---|---|---|---|---|---|---|---|---|---|
| Method of problem formulation | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Type of optimization technique | 2 | 1 | 2 | 3 | 2 | 2 | 2 | 3 | 3 |
| Validation | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Data source usage | 1 | 1 | 2 | 1 | 1 | 2 | 2 | 2 | 2 |
| Feasibility | 1 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 1 |
| Total | 7 | 6 | 9 | 8 | 7 | 9 | 9 | 9 | 9 |

prioritization respectively. For minimization and prioritization of test cases it may be concluded:

1) Genetic algorithms and greedy algorithm are again the most commonly used technique;

2) For test case minimization and prioritization all types I.e. Structural, Functional, Mutation, and Fault based criteria have been used. Among non-functional criteria execution time or cost is mainly considered, usability, security criteria are left out while optimizing test cases.

3) General purpose computing on graphical processing units (GPGPU) based technique has been used in both the domains for optimization. Marchetto *et al.* [37] proposed technique for multi-objective prioritization in which they have considered cost, code and requirement coverage together. They have used Information Retrieval technique LSI (Latent Semantic Analysis) for recovering the traceability links between requirements and code. Further, research in this direction, considering both requirements and code coverage together in optimization may be a good possibility.

### D. ANALYSIS FOR RESEARCH QUESTION 1

For software testing optimization, selecting adequacy criteria is one of the key steps. One of the research questions is to find out the traditional and non-traditional adequacy criteria as well as their combinations which have been used or possible combinations which can be used together for multi-criteria optimization. Considered literature is analyzed and is represented in Figure 2. Test adequacy criteria or obligations are of various types such as Structural (statement, branch, condition, path, multiple condition), Functional (requirement coverage), Model-based, Fault-based (mutation analysis, fault detection rate) and Non-functional such as execution time, security, usability and safety. From the study, it can be concluded that structural criteria such as statement, branch and multiple condition coverage; fault-based criteria

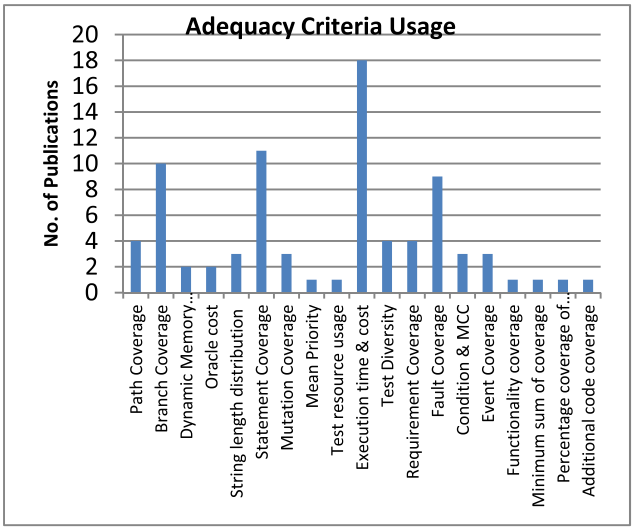



**FIGURE 2.** No. of publications where considered adequacy criteria is used.

such as mutation analysis and fault coverage; non-functional criterion execution cost or time have mostly been considered for optimization in various testing domains. Functional and model-based criteria, as well as non-functional properties such as security and usability, are less considered for optimization. Test diversity, dynamic memory consumption, mean priority and test resource usage are some of the least used criteria for testing. Data flow criteria for optimization are choice of fewer researchers compared to control flow criteria.

From review, it may also be seen that code coverage has been the choice of many researchers in all the four considered domains: test case generation, selection, minimization and prioritization. Zhu *et al.* [72] have suggested that, code coverage is not boundlessly appropriate as it may happen that some part of code is infeasible I.e. dead or not coverable. Thus, no test will be able to cover that part of code. Almulla *et al.* [71] have also analyzed nine faults of guava project from which they were able to detect only three faults on basis of branch coverage-based test suites. They found that coverage is not enough for detecting all faults. Other factors such as Input values, data types of input, complex data types and order of function call play an important role in triggering and detecting fault. Thus, code coverage should not be the main criterion for testing.

For the combination of adequacy criteria which have been used together Table 8, Table 9 and Table 10 should be considered for domain test case generation, selection, minimization, and prioritization respectively. Among all combinations, the most used combinations are statement and branch coverage, branch and mutation coverage, code coverage and execution time, code coverage and fault detection capability, requirement coverage and execution time. Combinations which are least used are code coverage and test diversity, event coverage and fault coverage, path and fault coverage, code coverage and requirement coverage.

During literature various recently proposed adequacy criteria were identified such as behavioral adequacy [41], fuzzy entropy of test cases [20], maximizing minimum sum of coverage [15], diversity aware mutation adequacy criteria [42], and operational coverage [75] which has already been discussed in Section 4.5.

Branch, path and mutation coverage are the popular adequacy criteria. But when these criteria are used, a gap between the syntax of the code and observable program behavior is easily visible. These criteria mislead about the level to which program behavior has been inspected. Almulla *et al.* [71] have also suggested that input values, data types of input are important factors to be looked for detecting faults as they may be the cause for triggering the fault. Behavioral adequacy criterion [41] fills this gap. It takes care of both code coverage and input. Authors stated that if a test set is behavioral adequate then it would be possible to know the behavior of the program and it could foresee the output for input that has not been considered. The proposed approach can further be improved by using better coverage criteria such as multiple condition coverage or studied in terms of black box coverage also.

Diversity aware mutation criterion [42] has improved mutation testing, by distinguishing mutants based on their kill information. Though it takes more computation cost but in terms of fault detection it's better than traditional kill-only mutation adequacy criterion. Thus, it subsumes the existing kill-only criteria. With branch coverage-based test cases the results are more improved. Thus, if fault detection and software quality is important and there is more budget for testing then stronger diversity aware criterion can be used.

Ambiguity based classification of test cases for test case selection has been done only by Kumar *et al.* [20]. They have used fuzzy entropy-based technique to select low ambiguity test cases and then tested the software under test. As classification of tests for selection is a new method, a lot of existing classification techniques such as neural network, K-mean classification etc. can be used. Results show that with fewer test cases they were able to get better results.

Operational coverage criterion [75] considers various entities such as statement, function, branch etc. and assigns weight to them based on their importance with usage profile. It acts as a good discontinuing rule for operational profile based testing. Results show that it performs better than traditional coverage in terms of fault-finding capability with fewer test cases for test case selection.

Maximizing minimum sum of coverage [15] considers idea that test cases should be selected such that it is maximizing the minimum coverage of element in a program. If three elements are there in a program, then the selected test cases should be such that the minimum coverage from among these three elements should be the maximum achieved minimum coverage by the selected test cases. It provides even coverage through all the software elements, and thus fault detection capability is also increased. It can be studied not only in terms code but other coverage also such as requirement coverage.

**TABLE 15.** Analysis of recently proposed criteria identified in literature.

| Criterion | Domain of Criterion | Usability in testing Domain | No. of Research paper | Optimization Technique Used | Validation | Future work |
|---|---|---|---|---|---|---|
| Behavioural Adequacy [41] | Inference based adequacy + code coverage | Test case generation | 3 [41, 51, 52] | Genetic Algorithm | Validated on 18 small programs from known source | • Adapt to more complex system<br>• Use other syntactic coverage criteria<br>• Defining it in terms of black box testing. |
| Fuzzy Entropy of test cases [20] | Ambiguousness of tests | Test case selection | 3 [20, 53,54] | Ant Colony Optimization | Validated on Print_Tokens and Print_Tokens2 | • Comparison with other classification techniques such as K-mean, SVM etc.<br>• Use other approach apart from ACO |
| Maximizing Minimum sum of coverage [15] | Coverage based criterion | Test case selection and Prioritization | 1 [15] | Integer Linear Programming and greedy approach | 5 Java programs from SIR Repository | • D Function can be used for other coverage also such as requirement coverage. |
| Diversity Aware Mutation Adequacy [42] | Mutation Testing | Test case selection and prioritization | 4 [42, 55,56, 69] | Greedy, NSGA-II | Validated on programs from Defects4j Repository | • Optimization techniques can be used for better results |
| Operational coverage [75] | Operational Profile based testing | Test case selection | 2 [75, 76] | None | Validated on grep, gzip and sed | • Optimization techniques may be used |

These criteria may also be optimized and used in combination with other criteria such as diversity aware mutation adequacy may be used with code coverage or execution time or cost. Maximizing minimum sum of coverage may be used for requirement coverage or any other coverage-based criteria. Behavioral criterion is considered in terms of code coverage, other syntactic coverage-based criteria can also be tried. It may also be further improved by using support vector machine or state machine inference algorithms. Analysis of these criteria is represented in Table 15, which is done based on domain or category for it, No. of publications for given criterion, Usability in domains of testing, Optimization technique used, Validation and Future aspects.

Thus, adequacy criteria are the basis for software testing and there is a need to choose them wisely to improve testing and find maximum faults.

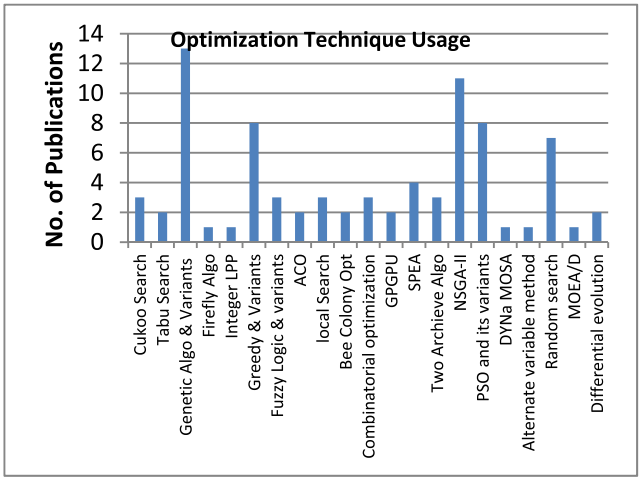



**FIGURE 3.** No. of publications where considered optimization technique is used.

### *E. ANALYSIS FOR RESEARCH QUESTION 2*

Second research question is to find out optimization techniques that have already been used and which can be used in software testing. Considered literature is analyzed and number of publications in which optimization technique is used are counted and represented in Figure 3.

Software testing is an NP complete problem and for solving it meta-heuristic algorithms such as PSO is a better choice compared to deterministic algorithm such as linear programming. From review it may be concluded that optimization techniques such as greedy technique, fuzzy logic, PSO, tabu search, genetic algorithm, random search etc. all have been used either separately or in combination with other techniques in software testing. Genetic algorithm & its variants are the first, greedy and PSO with their variants are the second choice of researchers. In case of multi-objective algorithms, NSGA-II was the choice of many researchers. In many cases to evaluate proposed technique, it was compared with random search. These techniques have their own pros and cons.

So, it's more beneficial to use a hybrid technique which has good quality of many techniques. Researchers have

used hybrid techniques also. After analysis of literature, few algorithms such as PSO-Bat algorithm (PSO-BA) [57], Bat-Harmony algorithm (BA-HS) [61], Firefly-harmony algorithm (HS-FA) [60], Tabu-firefly hybrid (TSFF I and TSFF II) [58] and multi-objective bat algorithm (MOBA) [59], Dragonfly Algorithm [73] have been identified and to the best of our knowledge they are scarcely or not used in software testing.

PSO-BAT algorithm [57] is communication strategy between Particle Swarm Optimization (PSO) and Bat Algorithm (BA) where PSO's worst particles will be switched with the top individuals in BA and vice-versa. PSO-BA has additional convergence and accuracy of up to 3% and 47% respectively compared to BA and PSO.

Bat algorithm may get caught into local optima. So, in HS pitch tuning operation is added which acts as a mutation operator while updating bat for next iteration and BA-HS hybrid is developed. According to Wang and Guo [61], ''It can speed up the global convergence rate without losing the strong robustness of the basic BA.'' Results suggest that HSBA produces improved solutions in comparison to the traditional population-based algorithms.

To overcome trapping of fireflies in local optima in HSFA [60], exploration and exploitation of HS and FA respectively are completely used. Thus, convergence speed of HS-FA has increased. Evaluation of benchmark test problems shows that HS-FA finds better values compared to other algorithms by making use of the useful knowledge more efficiently.

TSFF I and TSFF II are meta-heuristic algorithms proposed by Rohaninejad *et al.* [58] to solve job shop scheduling problem which take less computational time compared to GA algorithm. These algorithms also reduce the excess cost of procedures by effective use of the operation's slack.

Bat Algorithm is more desirable compared to other meta-heuristic algorithms as it can automatically zoom into a region where favorable solutions are found. It converges quickly and takes less time by swapping from exploration to exploitation. MOBA algorithm [59] also converges exponentially as well can deal with diverse Pareto optimal sets and higher nonlinear problems with complex constraints.

Meta-heuristic algorithm called dragonfly (DA) has been developed by Seyedali [73] which is based on swarming behaviors of dragonflies. It can be used to solve discrete, single and multi-objective problems as its various variants such as binary DA and multi-objective DA are also available. Its performance has been evaluated on various unimodal and multimodal test functions. Results show that DA algorithm performs better than PSO and GA. A diversity element in dragonfly algorithm has been introduced by Sugave *et al.* [74] and they have used it for test suite minimization.

In Table 16, we have summarized the advantages of PSOBA, BA-HS, HS-FA, TSFF I, TSFF II and MOBA meta-heuristic algorithms. These algorithms perform better in terms of convergence and accuracy than standard PSO, GA, Firefly algorithms which are widely used meta-heuristic algorithms in software testing. Thus, suggested algorithms are good candidate for resolving NP complete optimization problems in software testing.

**TABLE 16.** Advantages of suggested algorithms.

| Algorithm | Benefits |
|---|---|
| **PSO-BAT** | PSO-BA has more convergence and accuracy of up to 3% and 47% respectively compared to BA and PSO |
| **BAT-HARMONY** | Does not trap in local optima and has fast global convergence rate. |
| **FIREFLY-HARMONY** | HS-FA converges quickly compared to HS and FA |
| **TABU-FIREFLY** | TSFF I and TSFF II are much faster than the GA algorithm |
| **MOBA** | MOBA algorithm converges almost exponentially and can solve nonlinear problems with complex constraints. |
| **Dragonfly Algorithm** | It has higher exploration rate and is better in performance compared to PSO and GA |

**TABLE 17.** Adequacy Criteria Identified in Literature.

| Adequacy criteria | A | B | C | D |
|---|---|---|---|---|
| Statement coverage | ✓ | ✓ | ✓ | ✓ |
| Branch coverage | ✓ | ✓ | ✓ | ✓ |
| Path coverage | ✓ | | ✓ | ✓ |
| Multiple condition coverage | ✓ | ✓ | | ✓ |
| Oracle cost | ✓ | | ✓ | ✓ |
| Multiple branch coverage | ✓ | | | |
| Dynamic memory consumption | ✓ | | | |
| Fault coverage | | ✓ | ✓ | ✓ |
| Average % of coverage of changed code | | | | ✓ |
| Mutation Score | ✓ | ✓ | ✓ | ✓ |
| Execution time and cost | | ✓ | ✓ | ✓ |
| Test resource usage | | | | ✓ |
| Requirement coverage | ✓ | ✓ | ✓ | ✓ |
| Event coverage | ✓ | ✓ | | ✓ |
| String length distribution | ✓ | ✓ | | ✓ |
| Average % of coverage achieved | | | | ✓ |
| Test diversity | ✓ | ✓ | | ✓ |
| Mean priority | | ✓ | | |
| Additional coverage | | | | ✓ |
| Functionality coverage | | | ✓ | ✓ |
| **Recent Adequacy criteria** | | | | |
| **Minimum sum of coverage** | | ✓ | | |
| **Fuzzy entropy of test cases** | | ✓ | | |
| **Diversity Aware Mutation testing** | | ✓ | | ✓ |
| **Behavioral coverage** | ✓ | | | |
| **Operational coverage** | | ✓ | | |

Conclusion from analysis can be summed up in Table 17 and Table 18. These tables depict the adequacy criteria and optimization techniques encountered during the review respectively. A, B, C, D represents the domains of testing:

**TABLE 18.** Techniques identified in literature for Test Case Optimization.

| Optimization Technique | A | B | C | D |
|---|---|---|---|---|
| Genetic algorithm & its variants | ✓ | ✓ | ✓ | ✓ |
| Tabu search | ✓ | | | ✓ |
| Cuckoo search | ✓ | | ✓ | |
| Differential evolution | | ✓ | | ✓ |
| Firefly algorithm | ✓ | ✓ | | |
| Greedy & its variants | ✓ | ✓ | ✓ | ✓ |
| NSGA-II | ✓ | ✓ | ✓ | ✓ |
| PSO and its variant | ✓ | ✓ | ✓ | ✓ |
| Integer Linear programming | | ✓ | | |
| ACO | | ✓ | ✓ | |
| BCO | | | ✓ | ✓ |
| SPEA | | | ✓ | ✓ |
| Two archive algorithm | | | ✓ | ✓ |
| Local search | ✓ | ✓ | ✓ | |
| Alternate variable method | | ✓ | | |
| GPGPU | | | ✓ | ✓ |
| Fuzzy & its variants | | ✓ | ✓ | ✓ |
| Many objective sorting algorithm | ✓ | | | |
| Random search | ✓ | ✓ | ✓ | ✓ |
| Combinatorial optimization | | ✓ | ✓ | |
| **Optimization techniques not used in software testing** | | | | |
| **Multi objective bat algorithm** | - | - | - | - |
| **Dragonfly algorithm** | - | - | ✓ | - |
| **PSO-BAT** | - | - | - | - |
| **BAT-HARMONY** | - | - | - | - |
| **FIREFLY-HARMONY** | - | - | - | - |
| **TABU-FIREFLY** | - | - | - | - |

test generation, selection, minimization and prioritization. The domains in which they have been used are ticked (✓). So, from these tables research question 1 and 2 can be answered very well.

## V. CHALLENGES AND OPPORTUNITIES

Research question 3 is to find limitations in existing work. From analysis of Literature Review, it can be determined that there are many gaps in the existing work. Limitation of research work considered in the review has been provided as remarks in Table 8, 9 and 10. General limitations observed in literature review which provide us with further opportunities are:

### A. PARADIGM SHIFT OF CONSIDERING NONFUNCTIONAL CRITERIA WITH FUNCTIONAL CRITERIA IN SOFTWARE TESTING

To date, researchers have mostly focused on testing software for functional correctness. But apart from functional correctness, there are other properties such as runtime, energy or battery use, quality of service, security, usability, performance of system etc. which tester needs to test. Sometimes it's difficult to present non-functional properties in the form of fitness function, but if they can be then techniques can be used to further optimize them. Work has been done in this direction also, but the number of publications is less compared to publications for SBST considering functional properties. Afzal *et al.* [49] have presented a review of optimization of non-functional properties of software and considered research work from 1996-2007. They were able to find only 35 research work in total: execution cost (15), service quality (2), safety (4), security (7) and usability (7). Thus, a lot of work can still be done in this direction.

### B. DANGER OF CODE COVERAGE DIRECTED TEST CASE OPTIMIZATION

Test adequacy in literature as well as in industry is widely considered in terms of syntactic concept i.e. Code coverage which consists of all control and data flow coverage criteria. Gay *et al.* [48] have also concentrated on the disadvantage of generating tests based on only coverage. They have emphasized that target should not be structural coverage. Instead, it should be used as a supplement with randomly generated test suites. A study on industrial work, NoiseGen has been done by Nardo *et al.* [70] for coverage-based test case selection, minimization and prioritization. For prioritization, results of their study show that industry studies produce less optimistic outcomes about the efficacy of coverage-based techniques. For minimization they concluded that with coverage-based minimization there is reduction in execution cost but with reduction in fault detecting capability also. Thus, it may be used only for noncritical systems. Hemmati [50] has also conducted study on five projects of Defects4j repository to know fault detection capability of control and data flow coverage criteria and concluded that: statement coverage is able to detect 10% of the faults; control-flow coverage combined together detects 28% of fault; basic du-pair coverage can find 79% of unobserved faults by control-flow coverage; about 15% of the faults might not be discovered by any of the coverage criteria considered. Thus, it's not necessary that if 100% coverage is achieved then all the faults will be detected. From literature review it may be concluded that code coverage is used often. So, researchers should use other criteria such as fault coverage, mutation testing and behavioral coverage with code coverage to achieve better fault detection.

### C. FINDING LINKS BETWEEN VARIOUS SOFTWARE ARTIFACTS

Testing is done on basis of various adequacy criteria which are based on software pieces e.g. source code and requirements. Optimization techniques in software testing generally consider one aspect either coverage of code or requirements and assume that all software artifacts are equally relevant. This will not help in revealing the faults related to coding and faults related to the misinterpretation of requirements simultaneously. So, finding links among many software artifacts such as code, requirements, test cases etc. is important because it will give information about part of code related to a requirement and which test cases are covering that code and requirement and will help in discovering both

types of faults with minimum test cases. Thus, it will cover structural, functional and application requirements; and cost dimensions. It will help in regression testing also as if it's known prior to changes in software about the linkage between code, requirements and tests then test cases can be selected that are appropriate for a fragment of the application or for implemented changes. Various data gathering methods such as Latent Semantic Indexing, vector space modeling can be used to find links between software artifacts.

### D. TESTING IN A WAY THAT HELPS DEBUGGING

Optimizing tests only with the aim of testing will not help in proper software development. After testing debugging is done which also requires a lot of efforts. For debugging, engineers need to locate the statements which are causing fault. Quality of test cases should be such that the fault localization efficiency of the test suites is not conceded. Many existing testing methods prevent adequate assimilation with the use of suitable test suites to allocate precedence to test cases which are used for fault localization. In present scenario, fault detection and fault localization are treated separately. Thus, if test cases are generated, selected or prioritized along these terms that they will help in early fault localization also then it will take less time for debugging.

### E. USE OF RECENTLY PROPOSED ADEQUACY CRITERIA AND BETTER ALGORITHMS

New adequacy criteria such as Behavioral adequacy, distinguishing mutation adequacy and maximizing minimum sum of coverage have been reviewed in this paper. They have not been used with optimization techniques or in multi-criteria optimization in software testing. A new combination of these and other criteria may be made and tried in software testing. Various hybrid algorithms such as PSO-Bat algorithm, Bat-Harmony algorithm, Firefly-harmony algorithm, Tabu-firefly hybrid and multi-objective bat algorithm etc. which have been discussed in the review are not used in search-based software testing. They may provide better results compared to other algorithms in less computation time

## VI. CONCLUSION

This paper presents a review on optimization techniques used in all domains of software testing. Main aim was to identify the adequacy criteria and optimization techniques which have been used a lot and which have further scope to be used. This aim has been fulfilled by answering the considered research questions. After analysis of the literature, it may be concluded that all types of optimization techniques are widely used in all domains of testing, but mostly intelligent and hybrid techniques are used in the time period of 2007-2018. There is a paradigm shift from single objective to multi-objective optimization as many objectives or criterion in software testing like coverage of client requirements and code, number of defects detected, test efforts/cost, mutant killing scores have been considered together. Though researchers have started considering multiple criteria together, they aggregate all criteria in a single objective function by assigning a weight for each criterion and then use single objective optimization technique. Multi-objective algorithms may give better results by finding pareto optimal solutions. In multi-objective algorithms, non-simulated genetic algorithm was the choice of most of researchers. Various gaps have been identified which can be considered as suggestions for future development in field of optimization in software testing. Some appropriate areas for future works are:

1) Most studies have focused on structural coverage, and there is limited evidence of the application of optimization techniques to other testing phases and types of coverage. Thus, a combination of criteria such as code coverage and requirement coverage should be tried.

2) Using optimization techniques with recently identified criteria such as behavioral adequacy, distinguishing mutation adequacy, max-min criterion and operational coverage etc.

3) Using new, hybrid and multi-objective algorithms such as PSO-Bat Algorithm, Bat-Harmony Algorithm, Firefly-harmony Algorithm, Tabu-firefly hybrid, multi-objective bat algorithm (MOBA) and Dragonfly algorithm etc. which have not been used in the field of software testing.

## REFERENCES

[1] B. Beizer, *Software Testing Techniques*, 2nd ed. New York, NY, USA: Van Nostrand, 1990.

[2] W. Miller and D. L. Spooner, "Automatic generation of floating-point test data," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 3, pp. 223–226, Sep. 1976.

[3] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, and K. Karapoulios, "Application of genetic algorithms to software testing," in *Proc. 5th Int. Conf. Softw. Eng. Appl.*, 1992, pp. 625–636.

[4] K. Lakhotia, M. Harman, and P. McMinn, "A multi-objective approach to search-based test data generation," in *Proc. 9th Annu. Conf. Genet. Evol. Comput.*, 2007, pp. 1098–1105.

[5] M. Harman, S. G. Kim, K. Lakhotia, P. McMinn, and Y. Shin, "Optimizing for the number of tests generated in search based test data generation with an application to the oracle cost problem," in *Proc. 3rd Int. Conf. Softw. Test., Verification, Validation Workshops*, Apr. 2010, pp. 182–191.

[6] A. Rathore, A. Bohara, R. Gupta, T. S. Prashanth, and P. R. Srivastava, "Application of genetic algorithm and tabu search in software testing," in *Proc. 4th Annu. ACM Bangalore Conf.*, Mar. 2011, pp. 23–26.

[7] P. R. Srivastava, R. Khandelwal, S. Khandelwal, S. Kumar, and S. Ranganatha, "Automated test data generation using cuckoo search and tabu search (CSTS) algorithm," *J. Intell. Syst.*, vol. 21, no. 2, pp. 195–224, Jul. 2012.

[8] P. R. Srivatsava, B. Mallikarjun, and X. S. Yang, "Optimal test sequence generation using firefly algorithm," *Swarm Evol. Comput.*, vol. 8, pp. 44–53, Feb. 2013.

[9] G. Fraser and A. Arcuri, "Whole test suite generation," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 276–291, Feb. 2013.

[10] G. Fraser and A. Arcuri, "Achieving scalable mutation-based generation of whole test suites," *Empirical Softw. Eng.*, vol. 20, no. 3, pp. 783–812, 2015.

[11] A. Shahbazi and J. Miller, "Black-box string test case generation through a multi-objective optimization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 4, pp. 361–378, Apr. 2016.

[12] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 122–158, Feb. 2018.

[13] G. Fraser and A. Arcuri, "A large-scale evaluation of automated unit test generation using evosuite," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 2, pp. 8–49, 2014.

[14] Y. Shin and M. Harman, "Pareto efficient multi-objective test case selection," in *Proc. Int. Symp. Softw. Test. Anal.*, 2007, pp. 140–150.

[15] S. Mirarab, S. Akhlaghi, and L. Tahvildari, "Size-constrained regression test case selection using multicriteria optimization," *IEEE Trans. Softw. Eng.*, vol. 38, no. 4, pp. 936–956, Jul. 2012.

[16] A. De Lucia, M. Di Penta, R. Oliveto, and A. Panichella, "On the role of diversity measures for multi-objective test case selection," in *Proc. 7th Int. Workshop Autom. Softw. Test*, Jun. 2012, pp. 145–151.

[17] L. S. de Souza, R. B. C. Prudêncio, and F. D. A. Barros, "A hybrid binary multi-objective particle swarm optimization with local search for test case selection," in *Proc. IEEE Brazilian Conf. Intell. Syst. (BRACIS)*, Oct. 2014, pp. 414–419.

[18] A. Zeeshan and A. Ahsan, "Exploration and analysis of regression test suite optimization," *ACM SIGSOFT Softw. Eng. Notes*, vol. 39, no. 1, pp. 1–5, 2014.

[19] D. Mondal, H. Hemmati, and S. Durocher, "Exploring test suite diversification and code coverage in multi-objective test case selection," in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2015, pp. 1–10.

[20] M. Kumar, A. Sharma, and R. Kumar, "An empirical evaluation of a three-tier conduit framework for multifaceted test case classification and selection using fuzzy-ant colony optimisation approach," *Softw., Pract. Exper.*, vol. 45, no. 7, pp. 949–971, 2015.

[21] D. Pradhan, S. Wang, S. Ali, and T. Yue, "Search-based cost-effective test case selection within a time budget: An empirical study," in *Proc. Genet. Evol. Comput. Conf.*, 2016, pp. 1085–1092.

[22] A. P. Agrawal and A. Kaur, "A comprehensive comparison of ant colony and hybrid particle swarm optimization algorithms through test case selection," in *Data Engineering and Intelligent Computing* (Advances in Intelligent Systems and Computing), vol. 542, S. Satapathy, V. Bhateja, K. Raju, and B. Janakiramaiah, Eds. Singapore: Springer, 2018.

[23] 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.

[24] D. J. Mala and V. Mohan, "Quality improvement and optimization of test cases: A hybrid genetic algorithm based approach," *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, pp. 1–14, May 2010.

[25] L. S. de Souza, P. B. C. de Miranda, R. B. C. Prudencio, and F. A. Barros, "A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort," in *Proc. 23rd IEEE Int. Conf. Tools Artif. Intell.*, Nov. 2011, pp. 245–252.

[26] B. Suri and I. Mangal, "Analyzing test case selection using proposed hybrid technique based on BCO and genetic algorithm and a comparison with ACO," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 2, no. 4, pp. 206–211, 2012.

[27] 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.

[28] B. S. Ahmed, T. S. Abdulsamad, and M. Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the cuckoo search algorithm," *Inf. Softw. Technol.*, vol. 66, pp. 13–29, Oct. 2015.

[29] W. Zheng, R. M. Hierons, M. Li, X. H. Liu, and V. Vinciotti, "Multi-objective optimisation for regression testing," *Inf. Sci.*, vol. 334, pp. 1–16, Mar. 2016.

[30] B. S. Ahmed, "Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing," *Int. J. Eng. Sci. Technol.*, vol. 19, no. 2, pp. 737–753, 2016.

[31] A. A. Ahmed, M. Shaheen, and E. Kosba, "Software testing suite prioritization using multi-criteria fitness function," in *Proc. 23rd IEEE Int. Conf. Comput. Theory Appl. (ICCTA)*, Oct. 2012, pp. 160–166.

[32] W. Sun, Z. Gao, W. Yang, C. Fang, and Z. Chen, "Multi-objective test case prioritization for GUI applications," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, 2013, pp. 1074–1079.

[33] Z. Li, Y. Bian, R. Zhao, and J. Cheng, "A fine-grained parallel multi-objective test case prioritization on GPU," in *Search Based Software Engineering* (Lecture Notes in Computer Science), vol. 8084, G. Ruhe and Y. Zhang, Eds. Berlin, Germany: Springer, 2013.

[34] A. Joseph and G. Radhamani, "Fuzzy C means (FCM) clustering based hybrid swarm intelligence algorithm for test case optimization," *Res. J. Appl. Sci., Eng. Technol.*, vol. 8, no. 1, pp. 76–82, 2014.

[35] M. Tyagi and S. Malhotra, "Test case prioritization using multi objective particle swarm optimizer," in *Proc. Int. Conf. Signal Propag. Comput. Technol. (ICSPCT)*, Jul. 2014, pp. 390–395.

[36] M. G. Epitropakis, S. Yoo, M. Harman, and E. K. Burke, "Empirical evaluation of Pareto efficient multi-objective regression test case prioritization," in *Proc. Int. Symp. Softw. Test. Anal.*, 2015, pp. 234–245.

[37] A. Marchetto, M. M. Islam, W. Asghar, A. Susi, and G. Scanniello, "A multi-objective technique to prioritize test cases," *IEEE Trans. Softw. Eng.*, vol. 42, no. 10, pp. 918–940, Oct. 2016.

[38] S. Wang, S. Ali, T. Yue, Ø. Bakkeli, and M. Liaaen, "Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search," in *Proc. 38th Int. Conf. Softw. Eng. Companion*, 2016, pp. 182–191.

[39] M. Azizi and H. Do, "Graphite: A greedy graph-based technique for regression test case prioritization," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2018, pp. 245–251.

[40] M. Pezze and M. Young, *Software Testing and Analysis: Process, Principles, and Techniques*. Hoboken, NJ, USA: Wiley, 2008.

[41] G. Fraser and N. Walkinshaw, "Assessing and generating test sets in terms of behavioural adequacy," *Softw. Test., Verification Rel.*, vol. 25, no. 8, pp. 749–780, 2015.

[42] D. Shin, S. Yoo, and D.-H. Bae, "Diversity-aware mutation adequacy criterion for improving fault detection capability," in *Proc. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2016, pp. 122–131.

[43] S. N. Weiss, "Methods of comparing test data adequacy criteria," in *Proc. 14th Annu. Int., Comput. Softw. Appl. Conf. (COMPSAC)*, Oct. 1990, pp. 1–6.

[44] F. S. Gharehchopogh, I. Maleki, N. Ghoyunchizad, and E. Mostafaee, "A novel hybrid artificial immune system with genetic algorithm for software cost estimation," *Magn. Res. Rep.*, vol. 2, no. 6, pp. 506–517, 2014.

[45] Z. A. Dizaji and F. S. Gharehchopogh, "A hybrid of ant colony optimization and chaos optimization algorithms approach for software cost estimation," *Indian J. Sci. Technol.*, vol. 8, no. 2, pp. 128–133, 2015.

[46] I. Maleki, L. Ebrahimi, and F. S. Gharehchopogh, "A hybrid approach of firefly and genetic algorithms in software cost estimation," *Magn. Res. Rep.*, vol. 2, no. 6, pp. 372–388, 2014.

[47] A. Martens, D. Ardagna, H. Koziolek, R. Mirandola, and R. Reussner, "A hybrid approach for multi-attribute QoS optimisation in component based software systems," in *Proc. Int. Conf. Qual. Softw. Archit.*, 2010, pp. 84–101.

[48] G. Gay, M. Staats, M. Whalen, and M. P. E. Heimdahl, "The risks of coverage-directed test case generation," *IEEE Trans. Softw. Eng.*, vol. 41, no. 8, pp. 803–819, Aug. 2015.

[49] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Inf. Softw. Technol.*, vol. 51, no. 6, pp. 957–976, 2009.

[50] H. Hemmati, "How effective are code coverage criteria?" in *Proc. Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Aug. 2015, pp. 151–156.

[51] N. Walkinshaw, "Assessing test adequacy for black-box systems without specifications," in *Proc. Int. Conf. Test. Softw. Syst.*, 2011, pp. 209–224.

[52] G. Fraser and N. Walkinshaw, "Behaviourally adequate software testing," in *Proc. 5th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2012, pp. 300–309.

[53] M. Kumar, A. Sharma, and R. Kumar, "Fuzzy entropy-based framework for multi-faceted test case classification and selection: An empirical study," *IET Softw.*, vol. 8, no. 3, pp. 103–112, 2013.

[54] M. Kumar, A. Sharma, and R. Kumar, "Multi faceted measurement framework for test case classification and fitness evaluation using fuzzy logic based approach," *Chiang Mai J. Sci.*, vol. 39, no. 3, pp. 486–497, 2012.

[55] D. Shin and D.-H. Bae, "A theoretical framework for understanding mutation-based testing methods," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2016, pp. 299–308.

[56] D. Shin, Y. Shin, and D.-H. Bae, "A theoretical and empirical study of diversity-aware mutation adequacy criterion," *IEEE Trans. Softw. Eng.*, vol. 44, no. 10, pp. 914–931, Oct. 2018.

[57] T. S. Pan, T. K. Dao, T. T. Nguyen, and S. C. Chu, "Hybrid particle swarm optimization with bat algorithm," in *Genetic and Evolutionary Computing* (Advances in Intelligent Systems and Computing), vol. 329, H. Sun, C. Y. Yang, C. W. Lin, J. S. Pan, V. Snasel, and A. Abraham, Eds. Cham, Switzerland: Springer, 2015.

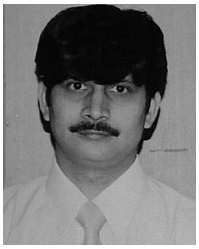
[58] M. Rohaninejad, A. S. Kheirkhah, B. V. Nouri, and P. Fattahi, "Two hybrid tabu search–firefly algorithms for the capacitated job shop scheduling problem with sequence-dependent setup cost," *Int. J. Comput. Integr. Manuf.*, vol. 28, no. 5, pp. 470–487, 2015.

[59] X.-S. Yang, "Bat algorithm for multi-objective optimization," *Int. J. Bio-Inspired Comput.*, vol. 3, no. 5, pp. 267–274, Sep. 2011.

[60] G. Lihong, W. Gaige, and W. Heqi, "An effective hybrid firefly algorithm with harmony search for global numerical optimization," *Sci. World J.*, Vol. 2013, pp. 1–9, Sep. 2013.
[61] G. Wang and L. Guo, "A novel hybrid bat algorithm with harmony search for global numerical optimization," *J. Appl. Math.*, vol. 2013, pp. 1–21, 2013.
[62] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Proc. Future Softw. Eng. (FOSE)*, May 2007, pp. 85–103.
[63] M. Kumar, A. Sharma, and R. Kumar, "Optimization of test cases using soft computing techniques: A critical review," *WSEAS Trans. Inf. Sci. Appl.*, vol. 8, no. 11, pp. 440–452, 2011.
[64] 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.
[65] A. Saswat *et al.*, "An orchestrated survey of methodologies for automated software test case generation," *J. Syst. Softw.*, vol. 86, no. 8, pp. 1978–2001, 2013.
[66] E. N. Narciso, M. E. Delamaro, and F. L. S. Nunes, "Test case selection: A systematic literature review," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 24, no. 4, pp. 653–676, 2014.
[67] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *Proc. Int. Conf. Softw. Test., Verification Validation (ICST)*, May 2015, pp. 8–23.
[68] S. M. Mohi-Aldeen, S. Deris, and R. Mohamad, "Systematic mapping study in automatic test case generation," in *New Trends in Software Methodologies, Tools and Techniques*, Vol. 265. Amsterdam, The Netherlands: IOS Press, 2014, pp. 703–720.
[69] D. Shin, S. Yoo, M. Papadakis, and D.-H. Bae, "Empirical evaluation of mutation-based test prioritization techniques," *Softw., Test. Verification Rel.*, vol. 28, no. 8, pp. 1–27, 2017.
[70] D. Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system," *Softw. Test., Verification Rel.*, vol. 25, no. 4, pp. 371–396, 2015.
[71] H. Almulla, A. Salahirad, and G. Gay, "Using search-based test generation to discover real faults in Guava," in *Proc. Int. Symp. Search Based Softw. Eng.*, 2017, pp. 153–160.
[72] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, 1997.
[73] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Comput. Appl.*, vol. 27, no. 4, pp. 1053–1073, 2016.
[74] S. R. Sugave, S. H. Patil, and B. E. Reddy, "DDF: Diversity dragonfly algorithm for cost-aware test suite minimization approach for software testing," in *Proc. Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, Jun. 2017, pp. 701–707.
[75] B. Miranda and A. Bertolino, "An assessment of operational coverage as both an adequacy and a selection criterion for operational profile based testing," *Softw. Qual. J.*, vol. 26, no. 4, pp. 1571–1594, 2018.
[76] B. Miranda and A. Bertolino, "Does code coverage provide a good stopping rule for operational profile based testing?" in *Proc. 11th IEEE/ACM Int. Workshop Autom. Softw. Test (AST)*, May 2016, pp. 22–28.

**NEHA GUPTA** received the B.Tech. degree in computer science engineering from Maharshi Dayanand University, India, in 2011, and the M.Tech. degree in software engineering from Delhi Technological University, India, in 2014. She is currently pursuing the Ph.D. degree with the Department of Information Technology, Indira Gandhi Delhi Technical University for Women, India. Her research interests include optimization techniques, software testing, and cost estimation.

**ARUN SHARMA** received the Ph.D. degree from Thapar University, Patiala, in 2009. He is currently an Associate Professor of IT with the Indira Gandhi Delhi Technical University for Women, New Delhi. Under his guidance, five students have completed their Ph.D. degree. He has published more than 60 papers in SCI/SCIE/SCOPUS and other international journals and conferences including IEEE, ACM, Springer, Elsevier, Wiley, and several others. His areas of interests include software engineering, autonomous systems, soft computing, and big data.



**MANOJ KUMAR PACHARIYA** received the master's degree from UP Technical University and the Ph.D. degree from Thapar University, Patiala, India. He is currently an Associate Professor with the Department of Computer Science and Applications, Makhanlal Chaturvedi National University of Journalism and Communication, Bhopal, India. He has published several research papers in international journals indexed in SCI and Scopus journals. His research interest includes software engineering focusing on software testing, software quality, test case generation, test case optimization, quality metrics, multi-objective optimization, component-based software development, search-based software development, data mining, association rule mining, soft computing, evolutionary computing, and nature inspired computing techniques. He is a member of several professional bodies including CSI, IEEE, ACM, SCRS, and International Association of Engineers Society. He is an Editorial Board Member of various international journals. He is also a Reviewer for various national, international conferences, and international journals.