Computers, Materials & Continua
DOI: 10.32604/cmc.2023.028625
Article

Tech Science Press

An Optimized Test Case Minimization Technique Using Genetic Algorithm for Regression Testing

Rubab Sheikh[1], Muhammad Imran Babar[2,*], Rawish Butt[3], Abdelzahir Abdelmaboud[4] and Taiseer Abdalla Elfadil Eisa[4]

[1]Bahria University, Islamabad, 46000, Pakistan
[2]Department of Computer Science, FAST-National University of Computer and Emerging Sciences, Islamabad, Pakistan
[3]Department of Computer Science, SZABIST, Islamabad, 46000, Pakistan
[4]Department of Information Systems, College of Science and Arts, King Khalid University, Mahayil Asir, Saudi Arabia
*Corresponding Author: Muhammad Imran Babar. Email: imran.babar@nu.edu.pk
Received: 14 February 2022; Accepted: 04 November 2022

Abstract: Regression testing is a widely used approach to confirm the correct functionality of the software in incremental development. The use of test cases makes it easier to test the ripple effect of changed requirements. Rigorous testing may help in meeting the quality criteria that is based on the conformance to the requirements as given by the intended stakeholders. However, a minimized and prioritized set of test cases may reduce the efforts and time required for testing while focusing on the timely delivery of the software application. In this research, a technique named *TestReduce* has been presented to get a minimal set of test cases based on high priority to ensure that the web application meets the required quality criteria. A new technique *TestReduce* is proposed with a blend of genetic algorithm to find an optimized and minimal set of test cases. The ultimate objective associated with this study is to provide a technique that may solve the minimization problem of regression test cases in the case of linked requirements. In this research, the 100-Dollar prioritization approach is used to define the priority of the new requirements.

Keywords: Test case minimization; regression testing; testreduce; genetic algorithm; 100-dollar prioritization

1 Introduction

Software testing activity has much importance in software development to deliver a good quality product. Testing requires high resource utilization to conform to the end-user requirements. According to an estimate, half of the software cost is incurred by software testing actions to meet the high quality of software [1]. To make a system more reliable and maintainable numerous testing techniques are presented in the literature like unit testing, black-box testing, gray box testing, white box testing, penetration testing, stress testing, regression testing, integration testing, system testing, performance testing, Graphical User Interface (GUI) testing, security testing, and compatibility testing. In this decade, agile and most used development processes follow an iterative, incremental approach to develop software where functionalities are added in short cycles. The incremental process increases the probability of fault injection when changes are made to in development cycles [2]. Regression Testing is a testing process used to confirm that modification or any change in the existing system does not disturb the performance of the existing software to ensure the system's reliability [3]. There are two methods to generate the test cases as manual and automated. Manual testing leads to high cost and waste of resources [4]. However, the automated test case generation may produce efficient results in terms of reduced time and cost [5]. As the developing software evolves from one cycle to another, the size of the test suite increases rapidly. In regression testing, it becomes hard to execute all the Test Cases (TCs) repeatedly to monitor the behavior of modified components. The execution of all test cases may result in higher time consumptions [6]. To save time and cost, only critical TCs are selected by software testers to analyze the performance of the application. The techniques to consider in regression analysis are TC selection, TC Prioritization (TCP) and Test Suite Minimization (TSM) or reduction [3]. The research focuses on TSM with optimal solutions to validate web-based applications with the use of Genetic Algorithm (GA). Nature-inspired algorithms became an active research area in the domain of software testing. GA has been discussed in many automatic test data generation [7,8] stress testing [9,10] integration testing [11], unit testing of classes in object-oriented programming [12] and high volume testing [13]. There are many techniques for test case generation of model-based web applications. However, there are few studies in the domain of TCP for web applications [14]. This research focuses on the TSM technique for regression testing in web applications based on incremental development.

The underlying study is organized as follows: Section 2 discusses the background of Regressing Testing and Test Case Prioritization. Section 3 is about related work in the domain of software testing. Section 4 presents the proposed technique and its implementation details. Section 5 is about experimentation and results. Section 6 concludes the underlying research.

2 Background

Software testing is significant in the quality of any software application. Due to a diversity of hardware platforms, operating systems, and programming languages, software testing has become challenging in software development [3]. Software plays an essential role in our lives ranging from simple calculations to complex systems in every field of life as health, entertainment, and education. Hence, software failure can lead to several problems like high financial losses, environmental vulnerabilities, and can also take human lives [15]. Software quality is associated with robust software testing that is considered as one of the indispensable activities to produce high-quality products with a low defect rate [16,17]. Moreover, "Software testing is the process of exercising or evaluating a system or system components by manual or automated means to verify that it satisfies specified requirements" [18].

Regression testing turns into a difficult task to execute all test cases when the test suite becomes voluminous. Regression testing incorporates a few steps to minimize the testing efforts as testers first prioritize the test cases before performing regression testing to test high priority requirements first. The process in which ordering of the test cases is carried out is called as Test Case Prioritization (TCP). The description of the general TCP criteria is presented in [3]. However, the most commonly used criteria or metric for the TCP is structural coverage [6,19,20]. The structural coverage results in the detection of maximum faults in a short period of time. In this research, 100-dollar technique is applied for TCP. The concept of cash is applied in 100-dollar and the participants of the technique assign a value to the list of requirements by assuming an imaginary amount of 100 dollars [21]. Many researchers attempted to use nature-inspired algorithms such as Genetic Algorithm (GA), Differential Evolution, Ant Colony Optimization (ACO), and Neural Network (NN) to get an optimum result for the regression TSM problems [22]. The authors in [22] attempted to prioritize TCs to uncover maximum faults in the system. An optimized technique was proposed by using ACO. Results showed a reduction of cost, effort, and time in performing regression testing [23]. Another nature-inspired algorithm cuckoo search was used in addition to mutation testing in [24]. A technique for TSM was presented in configuration-aware structural testing. Primarily, an optimized test suite was produced through combinatorial optimization. An adaptive mechanism was then used for further optimization [24].

TSM challenges have been handled in the literature using integer linear programming with a focus on certain criteria. These techniques fall short of computing optimum solutions, particularly when there is an overlap among test cases in terms of several criteria, such as code coverage and the collection of discovered errors. Recently, nonlinear formulas have been presented to handle such problems. However, these formulations need much more processing resources than linear ones [25]. Furthermore, they are susceptible to flaws that may result in sub-optimal solutions. The test case reduction and prioritization reduce the efficiency of test case scheduling while increasing the usage of time and resources. As a result, a research proposed a unique African Buffalo-based Convolution Neural Slicing (AB-CNS) technique for lowering time and resource usage during regression testing. Initially, the suggested model minimizes test cases and prioritizes them using the AB-CNS fitness function [26].

The TSM technique is used to select the smaller set of TCs that can uncover maximum faults in the system while minimizing cost and effort of testing at the same time. TCM is used to minimize the large set of TCs from the suite depending on specified criteria. In which multi-criteria TSM problem can be solved by a tool called Nemo, Nemo was considered as optimum solution for the multi-criteria test-suite minimization (MCTSM) problems, in which nonlinear problems can be turned into linear ones and then solve the problems by using Nemo. It can be used to achieve the best solution for MCTSM problems while considering up-to-date solvers. The study focuses on a practice to chain fault localization and detection. However, Outcomes showed the proposed solution can decrease the size of the test suite while considering both goals of fault localization and detection [27]. Scope-aided testing is a tactic presented in [28]. The key objective of the study was to upsurge the production by testing the reused code to discover faults. The empirical evolution of the study showed an average rate of fault detection can be improved by test suite prioritization while considering such faults that are in scope. In test suite and TSM the size of test suite can be considered by covering a reasonable number of faults.

3 Related Work

Nature-inspired algorithms are used in various computing or software engineering fields like cost estimation, cryptography, warehousing, or data mining problems. Genetic Algorithm (GA) is a search-based algorithm on the concept of biology, where the finest individuals are chosen for reproduction [29]. Numerous search algorithms especially GAs got interest of researchers to discover predictive functions for the relation. Many researchers explored the phenomenon of GA in contrast to regression test case prioritization [30,31]. Software testing is costly and takes more time, the use of GA produced better test data set. GA has been proven a successful solution to generate software test data automatically with the comparison of random testing GA tests all branches [9]. Test sets were generated to execute a wide range of branches in different programs comprising a scheme of five different measures mainly including a quadratic equation solver, linear and binary search procedures, remainder, and a triangle classifier. An experimental study was presented for the TCP technique using GA in [29]. This technique separated the test case discovered by the customer and among the rest of the test cases. It showed that when compared with arbitrarily prioritized TCS, a large set of faults can be detected. This experiment reveals that GA provides an effective and time-efficient solution by applying structural-based criteria to prioritize TCs.

GA was used to generate the test data, produce the optimal solution, and solve many problems during testing. To improve the performance it established the maintenance during searching [32]. The combination of GAs and coupling measurement was used to improve the optimal integration test orders in [11]. GA was used to reduce the cost-effectiveness that depends on coupling measurement. Coupling measurements were consumed to determine the intricacy of stubs. GA helped to access the best solution in a stable and efficient method. The usage method of GA was to create the best test arrangements that depend on coupling measurement. The system-level TCP scheme was presented in [31] to discover faults at an earlier stage of testing which can improve quality using GA. Multiple prioritization factors were discussed to strategize the proposed technique. Prioritization factors may seem concrete such as TC length fault proneness, coverage of code, and flow of data, or non-concrete such as the severity of faults. A TSM model in a Component-Based System was discussed in [24]. In the proposed model, a blend of GA and soft computing techniques was added into equal class partitioning in the generation of test suite to optimize the fitness values. A method of breeding software test cases in contrast to GA is presented in the process of the software testing cycle [8]. A fitness function was used for the fossil records of the organisms therefore it produces interesting search behaviors like novelty and severity. To analyze the particular fossil record and an overall search process, the study utilized several visualization techniques. GAs are used in breeding software test cases as a major aspect of a product testing cycle. An automatic method for software test case production is proposed in [12]. Test cases were produced for unit class testing under a general scenario. The researcher produced TCs by chromosomes as all statistics were contained that is which objects of a class to create, which sets of input should be used, and which methods should be utilized for those inputs. The experiment revealed that using a GA for test case optimization was extremely powerful in achieving optimal coverage in a reasonable time. The resultant test suites were relatively compact in nature. A mixed technique is presented with GA and Bee Colony Optimization (BCO) for test case reduction [9]. GA conforms the optimization problems and produce solution. It provides optimum results in minimum time span.

A methodology for automatic production of test information, utilizing state graphs and GA is debated in [7]. Exploratory proof demonstrated that the test information acquired can cover problematic ways and a structure that can be utilized to conform the requirements in testing. The Tabu Search algorithm is a novel technique, it has been used to find solutions for the real-time problems in the automatic test generation with GA during software testing for effective solutions [33]. The authors in [34] proposed weight-based GAs in order to effectively minimize the size of test suite, while achieving high pairwise feature coverage and the ability to detect a fault in the software test cases in the software manufacturing process. The testing activity was performed based on three objectives; Test Minimization Percentage (TMP), Fault Detection Capability (FDC) and Feature Pairwise Coverage (FPC) to build a genetic function for testing. They also proposed three weight-based GAs and it was very effective in producing considerable results. Their results showed that the Weight-Based GA for Multi-objective Optimization achieved the given threshold while others failed or hardly made up to the criteria.

Different search algorithms such as Greedy Algorithms, GAs, Hill Climbing, Additional Greedy Algorithm, and 2-Optimal Algorithm have been discussed in the literature to find a solution for

TC problems during regression testing. GA works better as compared to other search algorithms. Regression testing is very costly, but it is significant to a software development process. Test case prioritization intent to better perform regression testing by sequencing the TCs. Greedy Algorithms have been focused on TCP for regression testing however five search techniques have been used in [35]. GAs are also used in the development of functional testing. GAs are simple to cover a vast scope of development issues. GA finds better results in a very short duration. The maximum faults can be discovered during software testing. To conclude it is considered that most researchers find the GA technique useful that can find a better solution in a short period of time [36]. Regression testing may result in a decrease in reliability of the software due to reduction of some key use cases [37]. RT process is highly costly and time consuming and take significant time in case of larger test suits [38,39]. Some of the techniques are not suitable for RT as they are not suitable for larger test suits [40]. The other reported issues are related to the path coverage as the presented techniques are not suitable in terms of path coverage [41]. Fulfilment of the requirements is an important issue that is not handled by the existing techniques and focus on code check only [42]. We have proposed a technique TestReduce to ensure that the key use cases have not been eliminated after application of TestReduce.

4 Proposed Technique

This section presents proposed technique for regression testing for optimal test suite minimization with the use of Genetic Algorithm (GA). Details of each step involved in the proposed technique are presented in the subsequent sections.

4.1 TestReduce

TestReduce is based on GA to reduce regression TCs. The purpose of TestReduce is to optimize the solution that may best fit to find out a specific set of TCs to initiate regression testing. The selected set will help to test the quality of the corrected code and to analyze the ripple effect. The different steps involved in the TestReduce are mentioned in this section.

Step 1: Requirement Prioritization

A change in software requirements is directly associated with the regression. Hence, any change in the requirements will ultimately result in regression analysis. A better way can be paved for regression test case prioritization by giving due consideration to a changed scenario in terms of requirements. The changes in requirements can be categorized into two main categories. The first category of requirements is named as independent requirements that do not have any relationship or association with other requirements. The second category of the requirements is termed as associated requirements that show a certain extent of relationship or association with other requirements. This research does not focus on independent requirements. Hence, the key objective of this study is to provide a way that may solve the prioritization problem of regression test cases in the case of linked requirements.

Requirement's prioritization is taken as a first step to define the priority of the regression test cases. Hence, 100-Dollar prioritization approach was used to define the priority of new requirements. Therefore, if the change in requirements is higher, then ultimately the number of requirements will be higher in number. However, the prioritized value of a requirement is represented by variable 'p'.

Step 2: Requirements Associations

There are multiple words that are used for linked requirements like dependency, association, and relationship. However, from now onward the word association will be used for dependent requirements to maintain consistency. In the case of regression analysis, there is a need to redefine the priorities of the new requirements due to a given change in the standards and procedures of the developing environment. Moreover, the exploration of associations, with existing and new requirements, is significant to define the priority of test cases in a correct manner. The requirement associations, directly or indirectly, lead towards the phenomenon of traceability. In the phenomenon of traceability, the main goal is to trace a relation with the predecessor and successor. Hence, this association may result in defining the test case priorities more rigorously. The independent requirements that have no association with other requirements are easy to test as they are not going to affect any associated functionality. Fig. 1 describes the association among new and previously defined requirements.

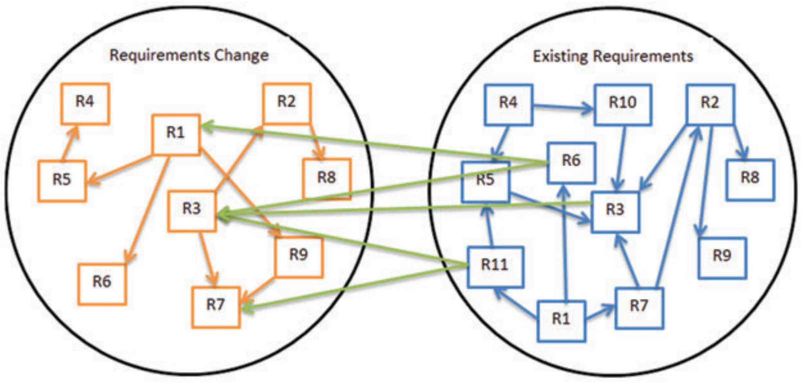

Figure 1: Associations of new requirements with existing requirements

- *Degree of Association or Affectivity of Requirements*
  The degree of association or affectivity of requirements among new and existing requirements is concerned with modules and views of the system. The degree of association is represented by 'd_a'. The value of 'd_a' may vary based on the associations. Its value may be zero or 'n'. Let suppose, there are n number of newly added requirements or rectifications based on the reported bugs in the sprint.

- *Degree of Association among New Requirements*
  The internal association of new requirements are extremely important. As it shows the worth of a requirement to be tested on a priority basis based on its inner degree of association with peer requirements. The inner degree of association among newly added requirements is represented by 'd_i'.

Step 3: *Rectification of the Bugs in Modules*

Moreover, the degree of bugs that are rectified in a particular module or web page adds more worth in defining priorities of the test cases that may help in minimization of test cases. The degree of bugs rectification is denoted by '*dr*'.

Step 4: Degree of Association of Rectified Modules

Step 4 is about the degree of association among rectified modules with other modules of the system. If the module is independent, then it can be tested with less effort as it does not affect other modules during the integration of system modules. However, in the case of association with other modules the rectification results in the ripple effect. To mitigate the ripple effect, the degree of association of rectified modules with other modules is highly vital and is considered as an input variable in this technique. It is denoted by 'dm'.

Step 5: Derived Objective Function

Objective function has key importance as test case prioritization is based on its value. The objective function used in this research is a problem of combinatorics and is shown in Eq. (1).

$$f\left(p,\ d_{a,}\ d_{i,},\ d_r, d_m\right) = (p + d_a + \ d_i + \ d_r + d_m) - 5 \tag{1}$$

Step 6: Application of Genetic Algorithm

GA is a search-based approach used to solve optimization problems to find out the most optimal set of TCs for a system. GA comprises steps like seed population to generate the required number of individuals. In the next step, score or fitness values are assigned to the generated individuals. In the next step, parents are selected for crossover to produce offspring. The newly born off-springs are assigned scores or fitness values. After finishing crossover, the next step is based on the selection of offspring for mutation. The result of mutation is a new offspring or mutated offspring. The next step is scoring or assigning fitness values to mutated offspring. After mutation, there comes the step of survival of the fittest. The new individuals are seeded to the population and if the convergence criterion is met then GA ends in an optimized result otherwise the generated offspring is fed back to the population. This process continues until the GA is converged to an optimal solution. The chromosomes in this research are based on values of ‘p’, ‘da’, ‘di’, ‘dr’, and ‘dm’.

- *Initialization of population*
  First, many more independent solutions are incidentally generated to form the initialization of the population. The size of the population depends upon the nature of the problems and these problems have many suitable solutions.

- *Fitness Function*
  In which independents are decided for mating from a population dependent on their fitness. Fitness is characterized as a trademark and capacity of an independent to endure and recreate in a situation. Determination produces the new population from the former one, in this manner beginning another age. The fitness estimation of every chromosome in the present age is dictated by a suitable assessment. Along these lines, the fitness esteem is utilized to choose a set of better chromosomes from the population for the later breeding.

- *Crossover*
  After the fitness function, the crossover activity is connected to the chromosomes chosen from the population. The crossover includes swapping of the succession of bits or qualities in the string between two independents. This procedure of swapping is done and rehashed every time with various parent independent until the later breeding has ideal independents.

- *Mutation*
  After the crossover procedure, the change activity is connected to an incidental selected subset of the population. Mutation prompts an adjustment of chromosomes in little better approaches to present great qualities. The primary point of change is to get a decent variety of population.

5 Experimentation and Results

In this research for experimental purposes Learning Management System (LMS) is used as a case study. The test cases are derived from existing requirements and evolved requirements of the LMS. There was a total of 55 test cases. The test cases from T1 to T39 are from existing requirements and test cases from T40-T55 are from evolved requirements. Then 100-dollar prioritization technique was applied by the three experts from industry to define the priority of test cases. The higher priority of the test cases was defined by using 100-dollar technique. The higher priority is denoted by 'p'. Requirement Prioritization values are shown in Fig. 2. It is the prioritization of existing requirements that contains old test cases. Requirement prioritization values of the new requirements that contain new test cases are shown in Fig. 3.

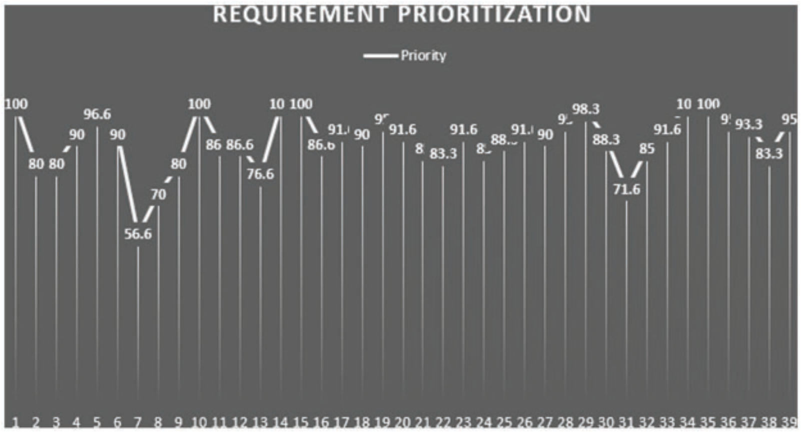

Figure 2: Requirement prioritization of old test cases

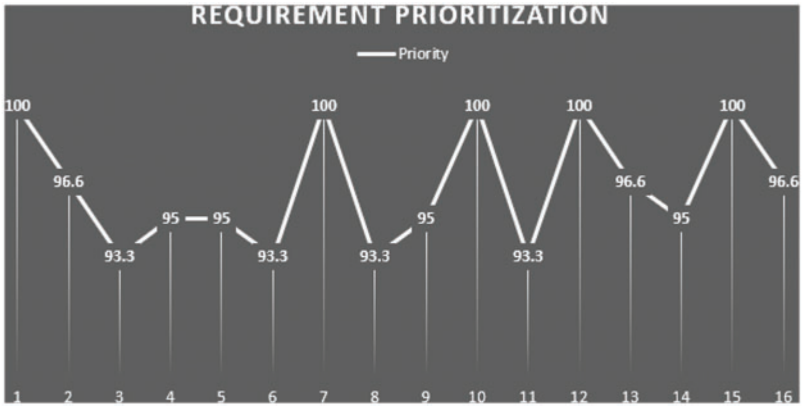

Figure 3: Requirement prioritization of new test cases

In the second step, the association among old and new test cases was defined. First one is the degree of association that is used to define the association among new and old requirements that have all test cases. It is denoted by 'da'. The highest degree of association is shown in Fig. 4. Second is the degree of association among new requirements that are used to define the association among new requirements only that have new test cases. It is denoted by 'di' and is shown in Fig. 5.

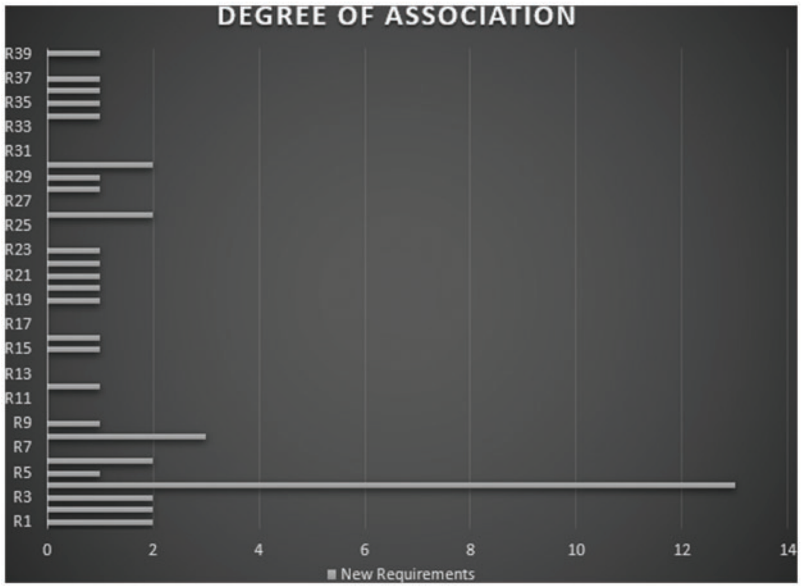

Figure 4: Degree of association

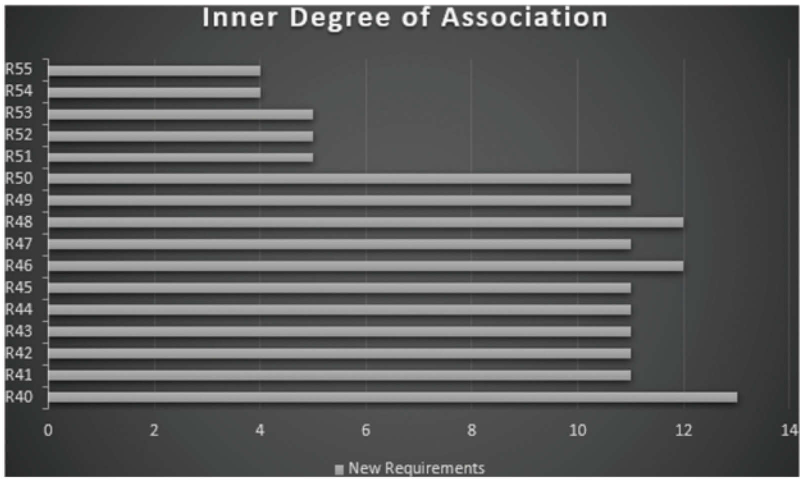

Figure 5: Degree of association among new requirements

In the third step, the degree of rectification among existing and evolved requirements is defined that contains all test cases. The rectification of old test cases is associated with the non-rectification of new test cases. This association changes the non-rectification of new test cases into the rectification. It is denoted by 'dr' and is shown in Fig. 6. In the fourth step, the degree of association of rectified modules is defined among existing and evolved modules that contains all test cases. It is denoted by 'dm' and is shown in Fig. 7.

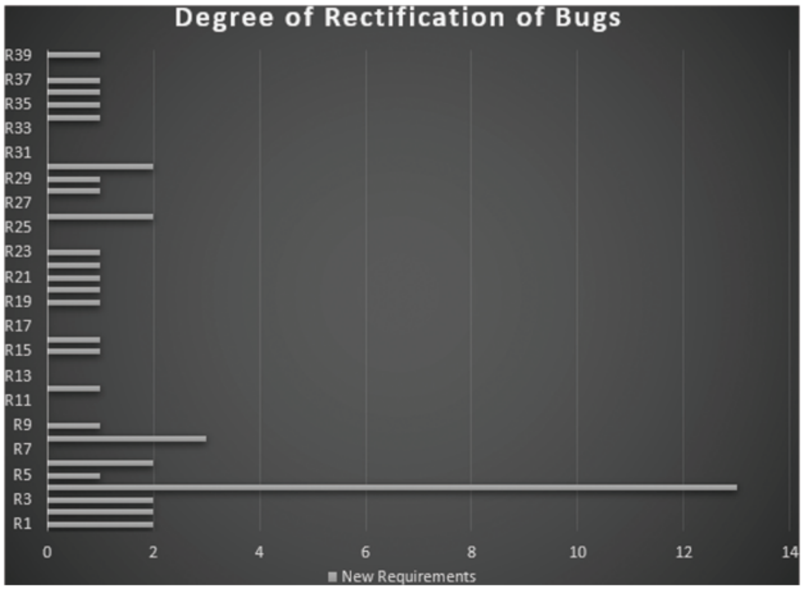

Figure 6: Degree of rectification of bugs

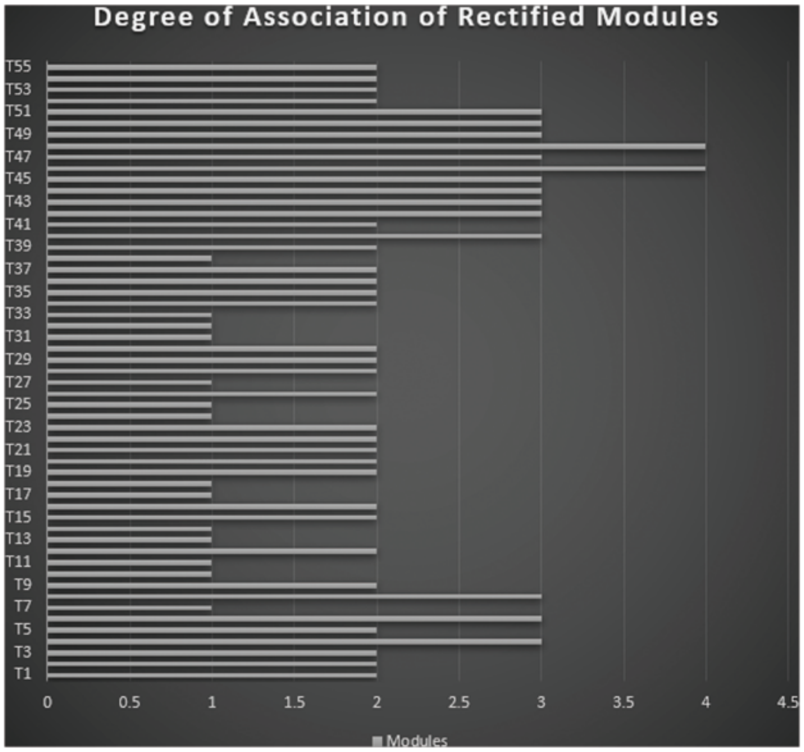

Figure 7: Degree of association of rectified modules

In the fifth step, the objective function is derived in which all above parameters (p, da, di, dr, dm) are used in the equation. The Eq. (1) is given below:

$$f\left(p,\ d_{a,}\ d_{i,},\ d_r, d_m\right) = (p + d_a +\ d_i +\ d_r + d_m) - 5 \quad (1)$$

In the above Eq. (1), 5 is the hypothetical constant value best suited for the said problem of software test case minimization.

p: The prioritized value of a requirement

da: The degree of association

di: The inner degree of association among newly added requirements

dr: The degree of rectific ation of bugs

dm: The degree of association of rectified module

The test case significance values are obtained by using derived objective function and are given in Table 1.

Table 1: Test case significance values using derived objective function

T. no.	(p	+	da	+	di	+	Dr	+	dm)	–	5	=	Ans
1	100	+	2	+	0	+	2	+	2	–	5	=	101
2	80	+	2	+	0	+	2	+	2	–	5	=	81
3	80	+	2	+	0	+	2	+	2	–	5	=	81
4	90	+	13	+	0	+	13	+	3	–	5	=	114
5	96.6	+	1	+	0	+	1	+	2	–	5	=	95.6
6	90	+	2	+	0	+	2	+	3	–	5	=	92
7	56.6	+	0	+	0	+	0	+	1	–	5	=	52.6
8	70	+	3	+	0	+	3	+	3	–	5	=	74
9	80	+	1	+	0	+	1	+	2	–	5	=	79
10	100	+	0	+	0	+	0	+	1	–	5	=	96
11	86.6	+	0	+	0	+	0	+	1	–	5	=	82.6
12	86.6	+	1	+	0	+	1	+	2	–	5	=	85.6
13	76.6	+	0	+	0	+	0	+	1	–	5	=	72.6
14	100	+	0	+	0	+	0	+	1	–	5	=	96
15	100	+	1	+	0	+	1	+	2	–	5	=	99
16	86.6	+	1	+	0	+	1	+	2	–	5	=	85.6
17	91.6	+	0	+	0	+	0	+	1	–	5	=	87.6
18	90	+	0	+	0	+	0	+	1	–	5	=	86
19	95	+	1	+	0	+	1	+	2	–	5	=	94
20	91.6	+	1	+	0	+	1	+	2	–	5	=	90.6
21	85	+	1	+	0	+	1	+	2	–	5	=	84
22	83.3	+	1	+	0	+	1	+	2	–	5	=	82.3
23	91.6	+	1	+	0	+	1	+	2	–	5	=	90.6
24	85	+	0	+	0	+	0	+	1	–	5	=	81
25	88.3	+	0	+	0	+	0	+	1	–	5	=	84.3
26	91.6	+	2	+	0	+	2	+	2	–	5	=	92.6

(Continued)

Table 1: Continued

T. no.	(p	+	da	+	di	+	Dr	+	dm)	−	5	=	Ans
27	90	+	0	+	0	+	0	+	1	−	5	=	86
28	95	+	1	+	0	+	1	+	2	−	5	=	94
29	98.3	+	1	+	0	+	1	+	2	−	5	=	97.3
30	88.3	+	2	+	0	+	2	+	2	−	5	=	89.3
31	71.6	+	0	+	0	+	0	+	1	−	5	=	67.6
32	85	+	0	+	0	+	0	+	1	−	5	=	81
33	91.6	+	0	+	0	+	0	+	1	−	5	=	87.6
34	100	+	1	+	0	+	1	+	2	−	5	=	99
35	100	+	1	+	0	+	1	+	2	−	5	=	99
36	95	+	1	+	0	+	1	+	2	−	5	=	94
37	93.3	+	1	+	0	+	1	+	2	−	5	=	92.3
38	83.3	+	0	+	0	+	0	+	1	−	5	=	79.3
39	95	+	1	+	0	+	1	+	2	−	5	=	94
40	100	+	4	+	13	+	4	+	3	−	5	=	119
41	96.6	+	3	+	11	+	3	+	2	−	5	=	110.6
42	93.3	+	2	+	11	+	2	+	3	−	5	=	106.3
43	95	+	2	+	11	+	2	+	3	−	5	=	108
44	95	+	3	+	11	+	3	+	3	−	5	=	110
45	93.3	+	2	+	11	+	2	+	3	−	5	=	106.3
46	100	+	7	+	12	+	7	+	4	−	5	=	125
47	93.3	+	4	+	11	+	4	+	3	−	5	=	110.3
48	95	+	4	+	12	+	4	+	4	−	5	=	114
49	100	+	2	+	11	+	2	+	3	−	5	=	113
50	93.3	+	3	+	11	+	3	+	3	−	5	=	108.3
51	100	+	2	+	5	+	2	+	3	−	5	=	107
52	96.6	+	3	+	5	+	3	+	2	−	5	=	104.6
53	95	+	2	+	5	+	2	+	2	−	5	=	101
54	100	+	1	+	6	+	1	+	2	−	5	=	105
55	96.6	+	1	+	6	+	1	+	2	−	5	=	101.6

A test case is selected for regression analysis:

if da or di or dr or dm > 2.

Hence, the minimized set of test cases is shown in the Table 2.

Table 2: Selected test cases by testreduce

T. no.	(p	+	da	+	di	+	Dr	+	dm)	−	5	=	Ans
4	90	+	13	+	0	+	13	+	3	−	5	=	114
6	90	+	2	+	0	+	2	+	3	−	5	=	92
8	70	+	3	+	0	+	3	+	3	−	5	=	74
40	100	+	4	+	13	+	4	+	3	−	5	=	119

(Continued)

Table 2: Continued

T. no.	(p	+	da	+	di	+	Dr	+	dm)	–	5	=	Ans
41	96.6	+	3	+	11	+	3	+	2	–	5	=	110.6
42	93.3	+	2	+	11	+	2	+	3	–	5	=	106.3
43	95	+	2	+	11	+	2	+	3	–	5	=	108
44	95	+	3	+	11	+	3	+	3	–	5	=	110
45	93.3	+	2	+	11	+	2	+	3	–	5	=	106.3
46	100	+	7	+	12	+	7	+	4	–	5	=	125
47	93.3	+	4	+	11	+	4	+	3	–	5	=	110.3
48	95	+	4	+	12	+	4	+	4	–	5	=	114
49	100	+	2	+	11	+	2	+	3	–	5	=	113
50	93.3	+	3	+	11	+	3	+	3	–	5	=	108.3
51	100	+	2	+	5	+	2	+	3	–	5	=	107
52	96.6	+	3	+	5	+	3	+	2	–	5	=	104.6
53	95	+	2	+	5	+	2	+	2	–	5	=	101
54	100	+	1	+	6	+	1	+	2	–	5	=	105
55	96.6	+	1	+	6	+	1	+	2	–	5	=	101.6

Let's suppose

$$(\mathrm{dp} + \mathrm{da} + \mathrm{di} + \mathrm{dr} + \mathrm{dm}) = \sum\nolimits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{d}$$

if

$$\mathrm{D} = \sum\nolimits_{i=1}^{n} \boldsymbol{d}$$

Then

$$\mathrm{f}(\mathrm{dp}, \mathrm{da}, \mathrm{di}, \mathrm{dr}, \mathrm{dm}) = \mathbf{D} - \mathbf{5}$$

Lastly, GA is applied to get the optimized results in order to minimize the test cases hence in application of GA

1. "d" is defining the input variables
2. "y=D − 5" is the fitness function and
3. "y" is defining the output variable

The best fitness draws the graph between the fitness value and the generation and shows the best function value in one-by-one generation.

The best individual draws the graph between the current best individual and the number of variable and shows the best function value of best individual in one-by-one generation.

The distance draws the graph between the average distance and the generation and shows the best function value of the average distance between individuals in one-by-one generation.

The expectation draws the graph between the expected values and the raw scores and shows the best function value of fitness scaling in one-by-one generation.

The genealogy draws the graph between the individuals and the generation and shows the red and blue lines that represented the mutation and crossover children in one-by-one generation.

The range draws the graph between the function value and the generation and shows the best, worst and mean scores in one-by-one generation.

The score diversity draws the graph between the number of individuals and the range that shows the score histogram at all generation.

The scores draw the graph between the number of individuals and the generation that shows the fitness value of each individual in one-by-one generation.

The selection graph shows the number of children and the individual. It shows the histogram of the parents that share properties in every generation as mentioned.

The stopping graph draws the stopping criteria points (among generation, time, stall (G) and stall (T)) and the % of criteria met that shows the best levels of stopping criteria.

The values of current iteration, objective function and final point of different parameters discussed in this section are highlighted in Table. 3.

Table 3: GA data

	Best fitness	Best individual	Distance	Expectation	Genealogy	Range	Score diversity	Fitness score of individual	Selection	Stopping
Current iteration	41	36	33	30	42	37	31	40	39	40
Objective function value	34.6	34.6	34.6	34.6	34.6	34.6	34.6	34.6	34.6	34.6
Final point	39.6	39.6	39.6	39.6	39.6	39.6	39.6	39.6	39.6	39.6

In this research, proposed methodology comprises 6 steps namely requirement prioritization, requirement associations, rectification of the bugs in modules, their degree of association, derived objective function and application of GA. Results showed proposed technique is effective to select an optimal set of test cases. Numerous solutions have been proposed to handle TSM problems. A novel African Buffalo-Convolution Neural Slicing (AB-CNS) based approach was proposed in [26] where RT is performed on the identification of faults. The proposed model slices out the retested classes and the fitness function of AB is defined for the classification layer. Association of requirements and test cases were not presented in this model. However, the proposed methodology uses defined mechanism for test case selection association of requirements and degree of associations helps to identify a good quality test cases and an objective function was derived to minimize the test suit. If the value of da, di, dm, dr > 2 then it will be selected. Moreover, the degree of bugs rectification adds value to it.

In AB-CNS approach once test cases are minimized, their prioritization depends on the calculation of the test case size. The performance of the model is calculated on the number of faults detected using this approach. The model detects the faults based on the fitness function. Whereas in proposed model GA was used to generate an optimal set of test cases with high priority. Results showed GA has successfully identified best individuals for regression test analysis and proved an effective solution for TSM problems in web-based environment. However, some limitations exist, it might be difficult to follow this approach when there's time constraint.

Table 4 shows the results of GA and the set of selected test cases that may provide maximum path coverage and exploration of the bugs. The selected approach has helped to find the test cases and by executing them maximum number of bugs were found. The bugs captured by each selected test case are mentioned in Table 4. The defects were seeded in the code. The minimized set of the test suit was executed and found the induced defects inside the web-based system. The calculated efficiency of the technique was acceptable in terms of capturing the defects. However, the key defects were detected by executing the minimized set of the test cases. The bug detection proved that the proposed technique is time efficient, cost effective and it reduced the effort being induced. Moreover, GA produced an optimum set of test cases with a higher accuracy to find out the bugs and to ensure the quality of the system under test.

Table 4: Selected test cases by GA

T. no.	P	da	di	dr	dm	F()	Bugs
4	90	13	0	13	3	114	4
8	70	3	0	3	3	74	1
40	100	4	13	4	3	119	3
41	96.6	3	11	3	2	110.6	2
42	93.3	2	11	2	3	106.3	2
43	95	2	11	2	3	108	3
44	95	3	11	3	3	110	1
45	93.3	2	11	2	3	106.3	1
46	100	7	12	7	4	125	4
47	93.3	4	11	4	3	110.3	3
48	95	4	12	4	4	114	5
49	100	2	11	2	3	113	3
50	93.3	3	11	3	3	108.3	2
51	100	2	5	2	3	107	3
52	96.6	3	5	3	2	104.6	2
53	95	2	5	2	2	101	1

6 Conclusion

Web-based applications play a vital role in gaining market leverage. When new requirements are added to the existing system the effect of new requirements or functions ripple across other modules or components of the web application. Hence, regression analysis and testing are significant in software testing. Most of the RT techniques that have been presented are costly, time-consuming, miss valuable test cases and error prone. Genetic Algorithm (GA) was used for optimization purposes to find out the most optimized set of TCs for a given system. The chromosomes in this research are based on values of 'p', 'd_a', 'd_i', 'd_r' and 'd_m'. The requirement prioritization is important to define the priority of the regression TCs. Hence, for this purpose 100 Dollar prioritization approach is used to define the priority of the newly emerged requirements. However, if requirement change is high, then ultimately the number of requirements will be higher in number. The prioritized value of a requirement is represented by variable 'p'. In the case of regression analysis there is a need to redefine the priorities of the new requirements. Moreover, the associations of with existing and new requirements, is highly vital to define the priority of TCs in a correct manner. A new regression testing technique 'TestReduce' with the use of GA is proposed to optimize a solution in the form of test cases that may best fit to find out an optimal set of TCs to initiate regression testing. The selected set will help to test the quality of code and to analyze the ripple effect.

Acknowledgement: Special thanks to anonymous reviewers for reviewing this research. The authors fully acknowledged the Deanship of Scientific Research at King Khalid University for funding this work through Large Groups, Project under grant number RGP.2/49/43.

Funding Statement: The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through Large Groups, Project under grant number RGP. 2/49/43.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] D. R. Graham, "Software testing tools: A new classification scheme," *Softw. Testing, Verif. Reliab.*, vol. 1, no. 3, pp. 17–34, 1991. https://doi.org/10.1002/stvr.4370010304.

[2] R. H. Rosero, O. S. Gómez and G. Rodríguez, "15 years of software regression testing techniques-A survey," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 5, pp. 675–689, 2016. https://doi.org/10.1142/S0218194016300013.

[3] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Testing, Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, 2012. https://doi.org/10.1002/stvr.430.

[4] P. McMinn, "Search-based software test data generation: A survey," *Softw. Test. Verif. Reliab.*, vol. 14, no. 2, pp. 105–156, 2004. https://doi.org/10.1002/stvr.294.

[5] A. M. Khamis, M. R. Girgis and A. S. Ghiduk, "Automatic software test data generation for spanning sets coverage using genetic algorithms," *Comput. Informatics*, vol. 26, no. 4, pp. 383–401, 2007.

[6] G. Rothermel, R. H. Untcn, C. Chu and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, 2001. https://doi.org/10.1109/32.962562.

[7] R. Lefticaru and F. Ipate, "Automatic state-based test generation using genetic algorithms," *Proc. -9th Int. Symp. Symb. Numer. Algorithms Sci. Comput. SYNASC 2007*, pp. 188–195, 2007. https://doi.org/10.1109/SYNASC.2007.47.

[8] D. Berndt, J. Fisher, L. Johnson, J. Pinglikar and A. Watkins, "Breeding software test cases with genetic algorithms," *Proc. 36th Annu. Hawaii Int. Conf. Syst. Sci. HICSS 2003*, 2003. https://doi.org/10.1109/HICSS.2003.1174917.

[9] B. Suri, I. Mangal and V. Srivastava, "Regression test suite reduction using an hybrid technique based on BCO and genetic algorithm," *Int. J. Comput. Sci. Informatics*, vol. 4, pp. 26–33, 2013. https://doi.org/10.47893/ijcsi.2013.1113.

[10] L. C. Briand, Y. Labiche and M. Shousha, "Stress testing real-time systems with genetic algorithms," *GECCO 2005–Genet. Evol. Comput. Conf.*, pp. 1021–1028, 2005. https://doi.org/10.1145/1068009.1068183.

[11] L. C. Briand, J. Feng and Y. Labiche, "Using genetic algorithms and coupling measures to devise optimal integration test orders," *SEKE '02: Proc. of the 14th Int. Conf. on Software Engineering and Knowledge Engineering*, pp. 43, 2002. https://doi.org/10.1145/568766.568769.

[12] P. Tonella, "Evolutionary testing of classes," *ISSTA 2004-Proc. ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, pp. 119–128, 2004. https://doi.org/10.1145/1013886.1007528.

[13] D. J. Berndt and A. Watkins, "Investigating the performance of genetic algorithm-based software test case generation," *Proc. IEEE Int. Symp. High Assur. Syst. Eng.*, vol. 8, pp. 261–262, 2004. https://doi.org/10.1109/hase.2004.1281750.

[14] A. A. Andrews, J. Offutt and R. T. Alexander, "CHECK IF IT IS USED–testing Web applications by modeling with FSMs," *Softw. Syst. Model*, vol. 4, no. 3, pp. 326–345, 2005.

[15] J. R. Barbosa, A. M. R. Vincenzi, M. E. Delamaro and J. C. Maldonado, "Software testing in critical embedded systems: A systematic review of adherence to the DO-178B standard," *VALID 2011-3rd Int. Conf. Adv. Syst. Test. Valid. Lifecycle*, no. January, pp. 126–130, 2011.
[16] A. Singhal, A. Bansal and A. Kumar, "A critical review of various testing techniques in aspect-oriented software systems," *ACM SIGSOFT Softw. Eng. Notes*, vol. 38, no. 4, pp. 1–9, 2013. https://doi.org/10.1145/2492248.2492275.
[17] B. Nielsen, "Towards a method for combined model-based testing and analysis," *Model. 2014-Proc. 2nd Int. Conf. Model. Eng. Softw. Dev.*, pp. 609–618, 2014. https://doi.org/10.5220/0004873106090618.
[18] A. Orso and G. Rothermel, "Software testing: A research travelogue (2000–2014)," *Futur. Softw. Eng. FOSE 2014-Proc.*, pp. 117–132, 2014. https://doi.org/10.1145/2593882.2593885.
[19] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, 2002. https://doi.org/10.1109/32.988497.
[20] A. G. Malishevsky, G. Rothermel and S. Elbaum, "Modeling the cost-benefits tradeoffs for regression testing techniques," *Conf. Softw. Maint.*, pp. 204–213, 2002. https://doi.org/10.1109/icsm.2002.1167767.
[21] J. Ali Khan, I. Ur Rehman, Y. Hayat Khan, I. Javed Khan and S. Rashid, "Comparison of requirement prioritization techniques to find best prioritization technique," *Int. J. Mod. Educ. Comput. Sci.*, vol. 7, no. 11, pp. 53–59, 2015. https://doi.org/10.5815/ijmecs.2015.11.06.
[22] A. Bajaj and O. P. Sangwan, "Nature-inspired approaches to test suite minimization for regression testing," *Comput. Intell. Tech. Their Appl. to Softw. Eng. Probl.*, no. August, pp. 99–110, 2020. https://doi.org/10.1201/9781003079996-7.
[23] A. Ansari, A. Khan, A. Khan and K. Mukadam, "Optimized regression test using test case prioritization," *Procedia Comput. Sci.*, vol. 79, pp. 152–160, 2016. https://doi.org/10.1016/j.procs.2016.03.020.
[24] B. S. Ahmed, "Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing," *Eng. Sci. Technol. an Int. J.*, vol. 19, no. 2, pp. 737–753, 2016. https://doi.org/10.1016/j.jestch.2015.11.006
[25] O. Ö. Özener and H. Sözer, "An effective formulation of the multi-criteria test suite minimization problem," *J. Syst. Softw.*, vol. 168, pp. 110632, 2020. https://doi.org/10.1016/j.jss.2020.110632.
[26] U. Sivaji and P. Srinivasa Rao, "Test case minimization for regression testing by analyzing software performance using the novel method," *Mater. Today Proc.*, pp. 1–9, 2021. https://doi.org/10.1016/j.matpr.2021.01.882.
[27] J. W. Lin, R. Jabbarvand, J. Garcia and S. Malek, "Nemo: Multi-criteria test-suite minimization with integer nonlinear programming," *Proc.-Int. Conf. Softw. Eng.*, no. 1, pp. 1039–1049, 2018. https://doi.org/10.1145/3180155.3180174.
[28] B. Miranda and A. Bertolino, "Scope-aided test prioritization, selection and minimization for software reuse," *J. Syst. Softw.*, vol. 131, pp. 528–549, 2017. https://doi.org/10.1016/j.jss.2016.06.058.
[29] P. K. Mishra and B. K. S. Pattanaik, "Analysis of test case prioritization in regression testing using genetic algorithm," *Int. J. Comput. Appl.*, vol. 75, no. 8, pp. 1–10, 2013.
[30] A. Kaur and S. Goyal, "A genetic algorithm for fault based regression test case prioritization," *Int. J. Comput. Appl.*, vol. 32, no. 8, pp. 975–8887, 2011.
[31] S. Raju and G. V. Uma, "Factors oriented test case prioritization technique in regression testing using genetic algorithm," *Eur. J. Sci. Res.*, vol. 74, no. 3, pp. 389–402, 2012.
[32] R. P. Pargas, M. J. Harrold and R. R. Peck, "Test-data generation using genetic algorithms," *Softw. Test. Verif. Reliab.*, vol. 9, no. 4, pp. 263–282, 1999. https://doi.org/10.1002/(SICI)1099-1689(199912)9:4&lt;263::AID-STVR190>3.0.CO;2-Y.
[33] E. Díaz, J. Tuya, R. Blanco and J. Javier Dolado, "A tabu search algorithm for structural software testing," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3052–3072, 2008. https://doi.org/10.1016/j.cor.2007.01.009.
[34] S. Wang, S. Ali and A. Gotlieb, "Minimizing test suites in software product lines using weight-based genetic algorithms," *GECCO 2013-Proc. 2013 Genet. Evol. Comput. Conf.*, pp. 1493–1500, 2013. https://doi.org/10.1145/2463372.2463545.

[35] Z. Li, M. Harman and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Trans. Softw. Eng.*, vol. 33, no. 4, pp. 225–237, 2007. https://doi.org/10.1109/TSE.2007.38.

[36] K. Singh and R. Kumar, "Optimization of functional testing using genetic algorithms," *Int. Jpurnal Innov. Manag. Technol.*, vol. 1, no. 1, pp. 43–46, 2010.

[37] L. You and Y. Lu, "A genetic algorithm for the time-aware regression testing reduction problem," *Proc.-Int. Conf. Nat. Comput.*, pp. 596–599, 2012. https://doi.org/10.1109/ICNC.2012.6234754.

[38] S. R. Sugave, S. H. Patil and B. E. Reddy, "DDF: Diversity dragonfly algorithm for cost-aware test suite minimization approach for software testing," *Proc. 2017 Int. Conf. Intell. Comput. Control Syst. ICICCS 2017*, vol. 2018-January, pp. 701–707, 2017. https://doi.org/10.1109/ICCONS.2017.8250554.

[39] D. B. Mishra, R. Mishra, A. A. Acharya and K. N. Das, "Test case optimization and prioritization based on multi-objective genetic algorithm," *Adv. Intell. Syst. Comput.*, vol. 741, pp. 371–381, 2019. https://doi.org/10.1007/978-981-13-0761-4_36.

[40] C. Coviello, S. Romano, G. Scanniello and G. Antoniol, "GASSER: Genetic algorithm for teSt suite reduction," *Int. Symp. Empir. Softw. Eng. Meas.*, 2020. https://doi.org/10.1145/3382494.3422157.

[41] M. C. Saputra and T. Katayama, "Code coverage similarity measurement using machine learning for test cases minimization," *2020 IEEE 9th Glob. Conf. Consum. Electron. GCCE 2020*, pp. 287–291, 2020. https://doi.org/10.1109/GCCE50665.2020.9291990.

[42] A. Stocco, A. Mesbah and A. Vahabzadeh, "Fine-grained test minimization," pp. 11, 2018. https://doi.org/10.1145/3180155.3180203.