Open Access **Baghdad Science Journal** **P-ISSN: 2078-8665**
2021, Vol. 18 No.2 (Suppl. June) **E-ISSN: 2411-7986**

DOI: http://dx.doi.org/10.21123/bsj.2021.18.2(Suppl.).1056

Multifactor Algorithm for Test Case Selection and Ordering

Atulya Gupta [*] ***Rajendra Prasad Mahapatra***

Department of Computer Science and Engineering, SRMIST -201204, Delhi-NCR Campus, Ghaziabad, U.P., India
[*]Corresponding Author: atulya.gupta.301@gmail.com [*] , mahapatra.rp@gmail.com
[*]ORCID ID: https://orcid.org/0000-0002-1801-7674 [*], https://orcid.org/0000-0002-9292-6331

Received 25/8/2020, Accepted 17/1/2021, Published 20/6/2021

Abstract:

Regression testing being expensive, requires optimization notion. Typically, the optimization of test cases results in selecting a reduced set or subset of test cases or prioritizing the test cases to detect potential faults at an earlier phase. Many former studies revealed the heuristic-dependent mechanism to attain optimality while reducing or prioritizing test cases. Nevertheless, those studies were deprived of systematic procedures to manage tied test cases issue. Moreover, evolutionary algorithms such as the genetic process often help in depleting test cases, together with a concurrent decrease in computational runtime. However, when examining the fault detection capacity along with other parameters, is required, the method falls short. The current research is motivated by this concept and proposes a multifactor algorithm incorporated with genetic operators and powerful features. A factor-based prioritizer is introduced for proper handling of tied test cases that emerged while implementing re-ordering. Besides this, a Cost-based Fine Tuner (CFT) is embedded in the study to reveal the stable test cases for processing. The effectiveness of the outcome procured through the proposed minimization approach is anatomized and compared with a specific heuristic method (rule-based) and standard genetic methodology. Intra-validation for the result achieved from the reduction procedure is performed graphically. This study contrasts randomly generated sequences with procured re-ordered test sequence for over '10' benchmark codes for the proposed prioritization scheme. Experimental analysis divulged that the proposed system significantly managed to achieve a reduction of 35-40% in testing effort by identifying and executing stable and coverage efficacious test cases at an earlier phase.

Key words: GA, Regression testing, Test cases, Test case minimization, Test case prioritization.

Introduction:

Whensoever evolution has occurred in the field of software, testing is required. Software testing is predominantly the operation conducted by testers to identify the defects or gaps and verify whether or not the system under consideration correctly complies with the client's specifications. During the software modification phase, the development teams, testers, and the stakeholders are more concerned about the authenticity and reliability of new features being worked on, not about the existing features that have been extensively tested and stable. As the latest piece of code is supposed to be incorporated with the existing features, during this time, it is exceedingly possible that any functionality may have been broken in the existing code. To ensure that the final product performs well even after the latest improvements have been pushed, regression testing must be executed. This notion also results in the formation and execution of a sizeable number of test cases and makes regression testing economically expensive in terms of maintenance, exhausting the testing budget to an approximation of up to 80% (1).

The unarguable reality that there are always thousands of variations and potential explanations why anything might go wrong is synonymous with testing. Sometimes, testers with a vivid imagination are merely unable to spot each one of them, particularly if the launch's delivery date is getting closer. Also, there is never sufficient time and resources for all alternative test conditions to be found and tested. So it is necessitated to reduce or prioritize the test conditions to retain the testing

process (2). Thence, this study's focal point is to emphasize the prime issue in software testing research, i.e., optimization of test cases ('Minimization' + 'Prioritization' strategies). A large body of research exists for Test Case Minimization, which generally executes fewer test cases depending upon some criteria. Fundamental delineation of the problem of selecting a reduced set of test cases could be (3,4):

Definition 1: A test suite T, a series of testing requirements r_1, r_2……r_n that must be tested in order to have an appropriate testing coverage in accordance with the program and a list of subsets of T, one associated with each of the requirements (iterating from 1 to n) such that any one of the test cases T_i belonging to the subsets of T can be used to test the requirement r_j.

Problem: Find a representative set of test cases T_i that will satisfy all of the r_j's.

Various approaches have been hypothesized to minimize the test suites. For instance, Harris and Raju (5) expounded an idea for diminishing the test suite size by plying an uncomplicated approach that concentrates on test metrics (i.e., desideratum and size coverage) and accordingly proposed a CBTSR (Coverage Based Test Suite Reduction) algorithm. The principal contribution of their modus operandi embraced the construction of test cases and desiderata through data flow testing that aimed to inspect the physical framework of the program and to discover the sub-paths which were being traversed by variables. Lin et al. in (6) emphasized and empirically estimated Greedy-based strategies (i.e., cost-aware Greedy tactics and the auxiliary Greedy) using gzip space, siemens, and ant applications. The result of their estimation indicated the accomplishment of higher proficiency of fault detection and lesser cost for regression testing with cost-aware procedures.

With progression in production code, test suites can congregate redundancies overtime. Vahabzadeh et al. in (7) focused on fine-graining the test minimization procedure and thus proposed a model for the statement-level analysis of test cases. A technique accompanying a tool (named Testler) was presented to lessen substantial redundancies in test statements of test cases. Many empirical studies also articulated the degradation of FDE (Fault Detection Effectiveness) due to the reduction mechanism. Jeffrey and Gupta (8) focused on improving this fault detection capability by affixing a concept of selectively retaining those test cases that are fault revealing but reduced because of being redundant. The work discussed in (9) offered a solution for complications of regression test suite optimization, named FCBTSO (Fault Coverage- based Test Suite Optimization), which was formulated on HGS (Harrolds-Gupta-Soffa) test suite minimization strategy. Singh et al. in (10) propounded an algorithm to equilibrate the tradeoff between the time needed for test suite execution and their FDE.

A. Lawanna (11,12) described the design based technique for test cases, resulting in refinement of test case selection procedure and devised efficacious algorithms with the embedded concept of filtration, classification, and selection of germane test cases. Research regarding test case selection also deployed linear programming model to extract the subset of test cases, to rerun (13). The apprehension of these selection procedures was improvised in the study (14), where the weighted average sum of quantifiable aspects served as a base for the selection framework. Testing cost, code coverage, FDE of test suites, and code alter data were mentioned aspects of the study. Over the years, researchers also exercised NSGA- II (Non-dominated Sorting Genetic Algorithm II), a customary multi-objective approach, for reduction scheme. A variant of NSGA-II is presented in study (15) titled MORE+ (Multi-Objective test suite REduction).

Apart from reduction approaches, TCP (Test Case Prioritization) also proved to be efficacious, which optimally arrange the set of test cases for attaining certain criteria such as fault detection capability as expeditiously as possible. In one or the other way, this technique acquires two objectives, that is, re-ordering of test cases according to some criteria and detecting faults at the earlier stage, resultantly reducing testing time with much smaller overhead (16).

Along with the minimization of test cases, prioritization also encompassed a large body of research. Beena and Sarala (17) formulated a selection and prioritization approach, mainly concerned with the coverage aspect. Categorization of coverage particulars collected during the ordering of test cases reveals the static or dynamic nature. Zhou and Hao (18) conducted an extensive empirical study to evaluate and contrast different methods of prioritization based on these natures, together with various test granularities and coverage criteria. Mirarab et al. and other researchers utilized techniques such as BN (Bayesian network), clustering approach, or a hybrid technique incorporating both to prioritize the test cases. This hybrid technique

illustrated the idea of employing clustering methods for grouping the test cases according to the similarity based on the code coverage and then prioritizing those clustered test cases according to their probabilistic inference (i.e., the failure probability of test cases) being employed by BN model (19-21).

Many other researchers precisely worked on improving the prioritization of test cases by concentrating on real-world aspects, i.e., focused on practical priority features (22). The present study observed that the historical execution data is also significant as that data conveniently reveals how the test cases failed previously and to what extent the test cases are likely to fail later. Khalilian et al. in (23) deployed historical execution data for the computation of prioritization equations and modified them to possess dynamic coefficients. These enhanced mathematical equations were composed of execution history, test case priority, and historical FDE. Research presented in (24) ameliorated the history-based approach by applying it on each altered line of code, i.e., prioritizing the modified lines first and afterwards followed up with concerned test cases. Moreover, the data also depicted that some test cases have execution relations among them, i.e., the execution history of one test case predicts the other, therefore mining such execution relations among the test cases based on historical execution data would improve the optimization approach further (1,25). According to Tanzeem Bin Noor and Hadi Hemmati (26), practically, it is not necessitated that a failing test case would always be exactly identical to the test case being failed previously, viz. the failing test case could be a slightly altered version of the former failing test case to reveal a fault that is being undetected.

In the view of studies surveyed and the need for enhancement in optimization tactics, this research addresses some significant issues in the above-mentioned existing conventional systems for optimizing the test cases as:

- The existing traditional methods or the genetic process deployed for test case minimization exploits a single parameter that is unjustified, resulting in non-fulfilment of either the objective or the requirements to be procured during software testing.
- Many of the optimizations approach results in static execution of test cases; that is, the order of optimized test cases would not be updated based on the runtime execution of test cases. Although the authors (1) proposed the dynamic way of optimizing the test case execution result, which overpowers the previous concept, optimization is still needed to get the most desirable outcome.
- No specific methodology is described for handling the test case tied issues, i.e., the stated problem either solved by preferring the test cases through FCFS (First Come First Serve) approach or random selection between the tied test cases, resulting in lower efficacy of the system.

Consequently, this paper presents a methodology that:

- Considers multiple factors/ features/ parameters, regarding test case optimization approach and thence satisfying all the needed requirements as possible, for testing the system, i.e., it would:
  - Analyze historical execution data for mining fail/pass rules of test cases.
  - Compute the Fault Questing Potentiality (FQP) and Test case Dependency Score (TDS) from the historical execution data and pruned rules.
  - Minimize the test cases by exercising one of the computational-intelligence-based processes (Genetic algorithm) to keep the coverage measure similar to the test suite.
- Prioritizes the reduced test cases, accompanied by a specific strategic approach for resolving issues cognate with tied test cases, for the optimal outcome. Moreover, the main concentration of the proposed method lies in coverage only.

A Genetic algorithm (GA) is the widely used population-based approach inspired by the natural phenomenon of survival to the fittest. An extended version for applicability of Genetic algorithm in test case optimization could be understood from (27-32). The current study aims to manage the issues that occur during test case analysis by incorporating GA as a base structure and prioritizing test cases by suggesting key variables and strategies for the notion of the tied test cases.

The rest of the paper's organization is as follows: Subsequent section deals with the technique proposed for test case optimization (Section 2). For a concise overview of how the methodology progresses, this section includes three subsections. Section 3 addresses the confirmation of the findings and the possible future enhancements for the proposed method. The final section (Section 4) of this paper presents the conclusion.

Proposed Method:

The regressive test procedure makes it almost impossible to perform all probable and preferred tests.

That is why the significant challenge is to choose an adequate test for code. If this critical step is not accomplished, the code's important characteristic may not be covered by the testing process. Additionally, it is essential to prioritize the tests that are likely to emphasize issues and are paramount to code functioning. In order to deal with the complexities of testing procedures, this research suggested and implemented the techniques for optimizing test cases. The broad visual perspective is represented in Fig. 1. The production of the proposed model requires multiple stages, which are described below.

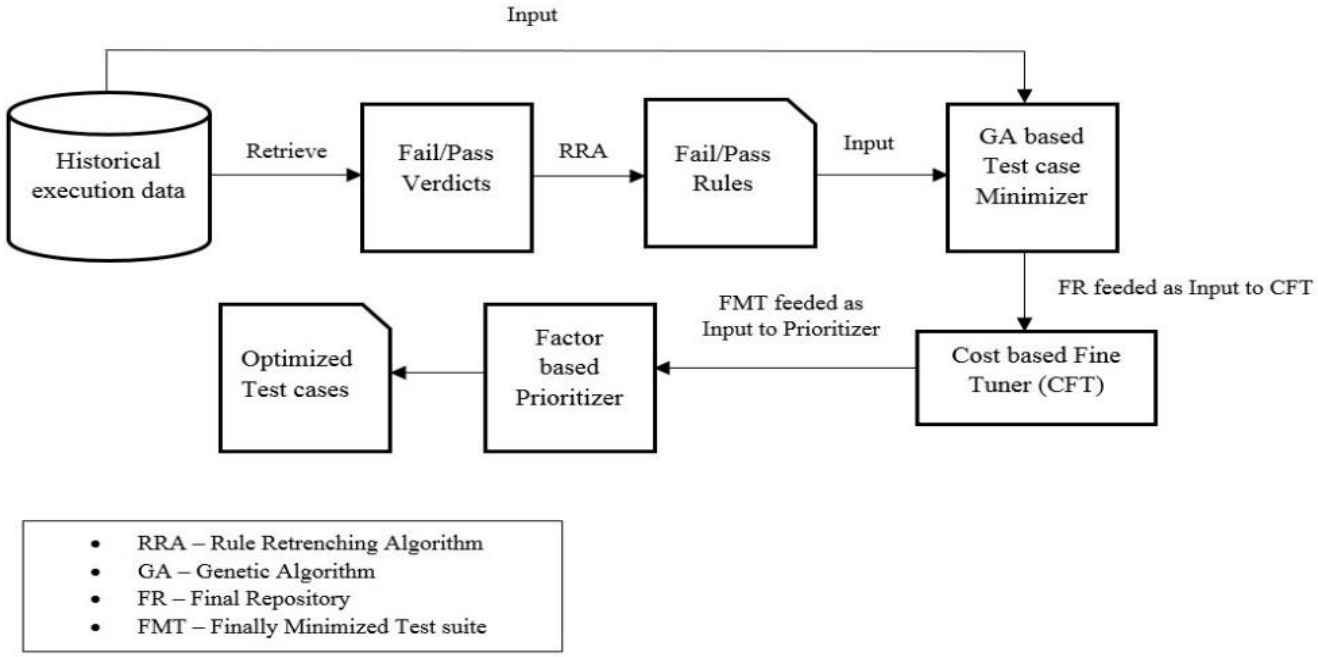


Figure 1. Diagrammatic representation of overall Proposed Approach (Block-representation)

Investigation and mining of test case linkage:

The conceptual framework of the proposed solution starts from here. In general, during regression testing, a background data depository is maintained, which is a storage for historical details of every test case. It includes the number of times a particular test case is being deployed, the faults disclosed by the test cases, and the severity of detected defects. In the course of progression and planning of the present work, it is observed that test cases' past performance provides insight for fail/pass verdicts of test cases. Any association was not considered before executing the test cases, but results manifested the link among the test cases after execution. The variables guided by the historical execution relationship of test cases were the current study's core concept.

Table 1. Sample data for the execution history of 10 test cases for five cycles

Execution cycles	Test cases									
	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
1	Pass	Pass	Fail	Fail	Pass	Pass	Fail	Pass	Pass	Pass
2	Fail	Pass	Pass	Fail	Fail	Pass	Pass	Pass	Fail	Fail
3	Fail	Pass	Fail	Pass	Fail	Pass	Pass	Fail	Fail	Fail
4	Fail	Fail	Pass	Fail	Pass	Fail	Fail	Pass	Fail	Pass
5	Pass	Pass	Fail	Pass	Fail	Pass	Pass	Fail	Pass	Fail

The relations of test cases are illustrated as which test case to be executed first to get the predicted test case executed precisely after that if it is found to be fault revealing. From Table 1, this concept could be delineated as if a test case T_1 fails or passes for a particular cycle; test case T_9 also generates the same verdict for that cycle. This notion also includes that the two test cases would have identical verdicts for all the execution cycles taken into consideration at that time. For test cases

revealing such a type of association, rules can be therefore pruned via RRA.

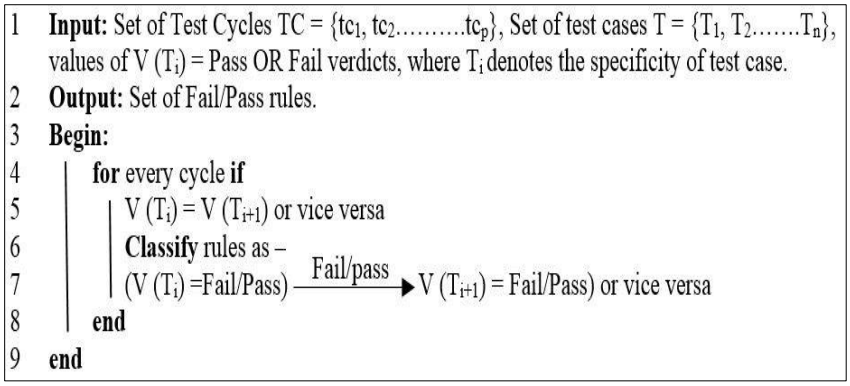


Figure 2. Algorithm 1: Rule Retrenching Algorithm (RRA)

Table 2. Fail/Pass rules derived from RRA

Fail Rules	Pass Rules
1. $(V(T_1) = Fail) \overset{fail}{\longrightarrow} (V(T_9) = Fail)$	2. $(V(T_1) = Pass) \overset{pass}{\longrightarrow} (V(T_9) = Pass)$
3. $(V(T_2) = Fail) \overset{fail}{\longrightarrow} (V(T_6) = Fail)$	4. $(V(T_2) = Pass) \overset{pass}{\longrightarrow} (V(T_6) = Pass)$
5. $(V(T_5) = Fail) \overset{fail}{\longrightarrow} (V(T_{10}) = Fail)$	6. $(V(T_5) = Pass) \overset{pass}{\longrightarrow} (V(T_{10}) = Pass)$
7. $(V(T_6) = Fail) \overset{fail}{\longrightarrow} (V(T_2) = Fail)$	8. $(V(T_6) = Pass) \overset{pass}{\longrightarrow} (V(T_2) = Pass)$
9. $(V(T_9) = Fail) \overset{fail}{\longrightarrow} (V(T_1) = Fail)$	10. $(V(T_9) = Pass) \overset{pass}{\longrightarrow} (V(T_1) = Pass)$
11. $(V(T_{10}) = Fail) \overset{fail}{\longrightarrow} (V(T_5) = Fail)$	12. $(V(T_{10}) = Pass) \overset{pass}{\longrightarrow} (V(T_5) = Pass)$

The pruned rules (Table 2) according to Algorithm 1 (Fig. 2) and historical execution data (Table 1) are adequate to calculate the factors like Fault Questing Potentiality (FQP) and Test case Dependency Score (TDS). These two components illustrated the empirical values depending on the number of times a test case failed or predicted others, respectively.

Fault Questing Potentiality (FQP) (1) would be the ratio of the number of times a test case failed to the number of times it is executed.

i.e. $FQP = \frac{Number\ of\ times\ a\ test\ case\ have\ a\ fail\ verdict}{Number\ of\ times\ the\ test\ case\ was\ executed}$

For example, from Table 1, the FQP value of T_1 will be 3/5, i.e., 0.6. Comparably, the FQP factor for every test case would be evaluated.

Test case Dependency Score (TDS) (1) clarifies the number of specific test cases whose outcome would be determined by implementing T_i using the stated failure and pass rules that are extracted from the historical data (Table 2). For example, it is only possible to use the execution of test case T_1 to determine the execution result of test case T_9 formulated on rules 1 and 2 of Table 2, and thus, the TDS value of T_1 will be '1'. Conversely, no test case can be determined based on the execution outcome of T_5, so the TDS value of T_5 will be '0'.

Reduction mechanism based on GA notion:

The concept proceeds with scheming the initial population for every execution cycle (i.e., for every source code). The gene formation for every execution cycle will describe the statement coverage by the particular test case, i.e., the gene value '0' will depict no coverage by a processing test case for the statement. In contrast, '1' will illustrate that the test case covers a particular execution cycle's statement.

```
Input: Set of test cases T = {T1, T2……..Tn}, Statement set S = {S1, S2……Sm}
Output: Initial population driven by test cases.
Begin:
    for every statement Sj in S if
        Test case Ti in T covers statement Sj in S
            then Return '1'
        Otherwise '0'
    end
end
```

Figure 3. Algorithm 2: Initial population generation

Algorithm 2 (Fig. 3) describes how to structure the initial population in context to test coverage. The current work explained the detailed methodological account with a case study comprising of ten test cases and five execution cycles as:

Initial Population: The 0-1 matrix will be formed depicting the statement coverage by the test cases respectively, where '0' and '1' are supposed to be a gene, while the complete coverage information by a test case for a particular cycle will be a chromosome (Fig. 4). Each information residing in the execution cycle is assigned some random weightage ranging in between [0-1]. The weightage factor discloses the criticality of the data and conditions that these statements are holding (Table 3).

Table 3. Showing the initial population (Statement coverage by test cases) with respective weightage values for execution cycle1

Weight for every statement (W_j)	Statements (S_i) for execution cycle '1'	Test cases									
		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
0.4	1	1	1	1	1	1	1	1	1	1	1
0.6	2	1	1	1	1	1	1	1	1	1	1
0.5	3	1	0	1	0	1	1	1	1	1	1
0.2	4	1	0	1	0	0	1	1	0	1	1
0.2	5	0	0	0	0	1	0	0	1	0	0
0.1	6	0	0	0	0	1	0	0	1	0	0
0.4	7	0	1	0	1	0	0	0	0	0	0
0.7	8	0	1	0	1	0	0	0	0	0	0
0.1	9	0	1	0	0	0	0	0	0	0	0
0.5	10	0	0	0	1	0	0	0	0	0	0
0.3	11	0	0	0	1	0	0	0	0	0	0
0.6	12	1	1	1	1	1	1	1	1	1	1
0.5	13	1	1	1	1	1	1	1	1	1	1
0.4	14	1	1	1	1	1	1	1	1	1	1
0.1	15	1	1	1	1	1	1	1	1	1	1

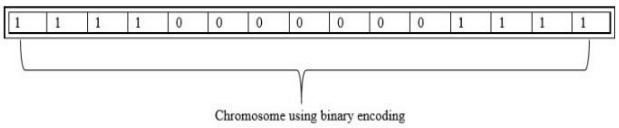


Figure 4. Portrayal of coverage information by test case '1' for execution cycle '1'

Fitness Function: The fitness of every test case for all of the 'p' execution cycle would be calculated as:

$$F(T_i) = \left[\left\{\sum_1^p\left\{\sum_{j=1}^m\left(CV\ (S_j) \times W_j\right)\right\}\right\} + FQP(T_i) + TDS(T_i)\right] \quad \ldots (1)$$

where 'p' denotes the number of execution cycles; for the considered case study, the value of 'p' is 5, CV (S_j) is the statement coverage value by the test case whose fitness is being computed, i.e., either '1' or '0' by the test case, W_j is the weight assigned to every specific statement, 'm' defines the total number of statement.

Input: Set of test cases T = {T_1, T_2……..T_n}, Set of Test Cycles TC = {tc_1, tc_2…….tc_p}, F (T_i) = Fitness value of each test cases in T.
Output: FR = Final Repository of test cases containing test case sequences for every execution cycle in TC, $fr_{p'} \in$ FR where $fr_{p'}$ evince every individual sequence of test cases present within the FR.
Begin:
  FR = { }
  $k \leftarrow 1$
  do
    do
      $fr_{p'} \leftarrow GA()$
      $FR = FR \cup fr_{p'}$
    while (50 % test cases are not in each $fr_{p'}$)
    $k \leftarrow k + 1$
  while ($k \leq p$)
end

Figure 5. Algorithm 3: Test Case Minimization

Input: Initial population of every test case in T
Output: Test sequences with 50% test cases for every execution cycle (EC) in TC.
Begin:
  For every test case T_i in T
    $P_1 \leftarrow Choose\ Parent(T_i[random(\,)])$ //Selection operator is applied i.e. two test case are randomly selected from the original test suite (set of test cases 'T').
    $P_2 \leftarrow Choose\ Parent(T_i[random(\,)])$
    $C_1 \leftarrow P_1\ ExOR\ P_2$ // Ex-OR operation is applied.
    if ($Cov(C_1) \geq 75\%$) **then** select P_1, P_2 in $fr_{p'}$ // P_1, P_2 get selected in FR for that EC.
    else if
      $C_2, C_3 \leftarrow Crossover(P_1, P_2)$ // Swap gene position in between P_1 and P_2.
      $C \leftarrow C_2\ ExOR\ C_3$ // Ex-OR operation is applied.
      If ($Cov(C) \geq 75\%$) then select P_1, P_2 in $fr_{p'}$ // P_1, P_2 get selected in FR for that EC.
    else if
      $C_4 \leftarrow Mutation(C)$ // weight based.
      If ($Cov(C_4) \geq 75\%$) then select P_1, P_2 in $fr_{p'}$ // P_1, P_2 get selected in FR for that EC.
    else
      $WQ\{\} \leftarrow LF\{P_1, P_2\}$ //place the low fitness valued test case from P_1, P_2 in the waiting queue as no P_1, P_2 satisfies the coverage criteria.
      $T\{\} \leftarrow T\{\} - WQ\{\}$ // remove that test case from the suite of test.
      check if
        Test cases in $T\{\} = 1\ or\ empty$ **then** // waiting queue is full.
        Reduce "coverage percentage" and **continue** the process again with all test cases dequeued from WQ{ } along with processing and left test case in T.
      end
      continue selection with the left test case i.e. either P_1 or P_2 with one randomly chosen parent from test suite i.e. T.
    $T\{\} \leftarrow T\{\} - \{P_1, P_2\}$ // remove selected test cases from original T.
    if
      Test cases in $fr_{p'}$ is 40% **then**
      $fr_{p'} \leftarrow fr_{p'} \cup HF\{P_1, P_2\}$ //from P_1 and P_2, select the test case in $fr_{p'}$ which has high fitness.
      Return $fr_{p'}$ and **terminate** for that EC and continue process for other cycles in TC.
    end
    else if
      WQ{ } is not empty **then**
      $fr_{p'} \leftarrow \{P_1, P_2\}$ and dequeue 1st test case from waiting queue.
      // **continue** selection operation with the dequeued test case and one randomly selected test case from the test suite i.e. T.
    else
      continue same course of action from **Begin** until 50% test cases are selected in test sequence of every EC.
    return $fr_{p'}$
end

Figure 6. Algorithm 4: GA( )

The above-stated algorithms, i.e., algorithm 3 and 4 (Fig. 5 and 6) elucidate the procedure as to how the reduction methodology works and how the genetic operators would aid in extracting the minimal amount of test cases for every execution cycle with at least 75% of initial termination criterion. The final termination requirements are set to 50% to extract the best test sequence from the original test suite for every execution cycle. Following is the elaborative working of genetic operators.

Selection: This study deploys a random selection scheme to initially select two test cases from the test suite and perform an Ex-OR operation between the randomly selected test cases. The consequent chromosome would be analyzed on coverage factor, i.e., the coverage percentile of the

resultant chromosome would either be equal to or exceeds 75% (initial termination), to be in the reduced test sequence of the processing execution cycle. Within the modules, a knock-out based selection scheme is followed. Coverage for every test case would be enumerated as:

$$Cov(T_i) = S_{covered}/S_{total} \quad \ldots (2)$$

where $S_{covered}$ = number of statements executed by the test case and S_{total} = total number of statements in the processing execution cycle.

Crossover: If the selected pair of test cases would not satisfy the initial termination criterion, then crossover is carried out. Those two test cases' genes are swapped from the position where both test cases are covering the same statement to the point where neither of the two test cases covers a statement (Fig. 7). This swapping is performed only once in between the genes of two test cases.

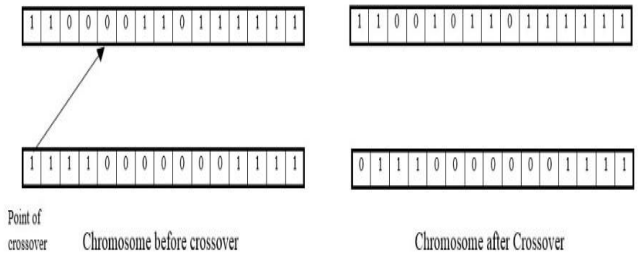


Figure 7. Crossover Operation

Ex-OR is performed betwixt the newly formed chromosomes, and coverage will be evaluated. If the coverage attained through the crossover mechanism gives unsatisfactory results, i.e., below 75%, then the outcome acquired through the crossover step is mutated.

Mutation: This refers to a slight change in the chromosome. The '0' bit is flipped to '1' according to the weightage assigned to the statements (single bit mutation).

Suppose the initial termination criterion is not satisfied by any of the three operators for the two test cases. In that case, from one of the two test cases, the test case with a low fitness value will be placed in a waiting queue. So, a waiting queue is maintained every time the pair of test cases would not satisfy the coverage criterion (Fig. 8). If the combination of randomly selected test cases fulfils the genetic loop coverage criteria, then those two test cases would be included in the processing cycle test sequence. The test sequences are further maintained in the final repository, and the selected test cases are excluded from the original test suite (T). Therefore for every execution cycle, a separate reduced test sequence is formed.

Whenever the inclusion of fitted pair of test cases in the reduced test sequence for a particular cycle occurred, the waiting queue is being inspected for the presence of any test case. If any test case exists, it will be extracted from there, and the procedure continues with the same extracted test case by pairing it with some other randomly selected test case from the original test suite. The last test case in the reduced test sequence will be fitness based only. If no test case satisfies the coverage criterion (initial termination), then the initial termination criterion should be dropped (relaxation). This genetic loop is repeated until the final termination criterion is met, i.e., 50%.

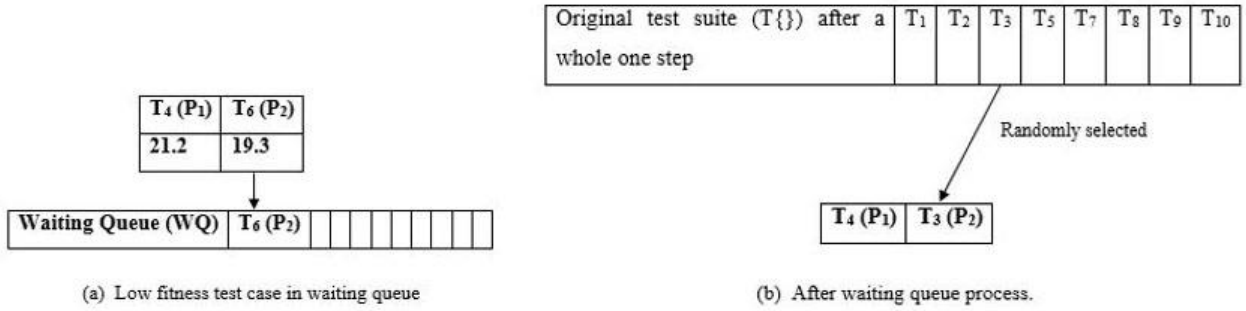


Figure 8. Exemplar view of how a test case with lower fitness value shifts to waiting queue and the random selection of parent from the test suite after shifting procedure

The reduction methodology denouement will result in a catalogue embedded with individual test sequences for every specific EC, i.e., FR. Further, CFT scrutinizes FR to have overall one best test sequence among all the test sequences existing in FR, which would be applicable to every execution cycle, i.e., FMT. Every test case in their respective execution cycles are covering a number of statements and also has some random weightage value for those statements. CFT utilizes these particulars for the assessment of the cost factor of every test case residing test sequences of FR as:

$$C(T_i) = \sum_{j=1}^{m} \{CV(S_j) \times W_j\} \quad \ldots (3)$$

For example, {T_5, T_2, T_6, T_4, T_1} is the sequence for execution cycle '1' of FR. The cost of each test case residing in this sequence would be computed as:

$C(T_5) = (1 * 0.4 + 1 * 0.6 + 1 * 0.5 + 1 * 0.2 + 1 * 0.1 + 1 * 0.6 + 1 * 0.5 + 1 * 0.4 + 1 * 0.1) = 3.4$
$C(T_2) = (1 * 0.4 + 1 * 0.6 + 1 * 0.4 + 1 * 0.7 + 1 * 0.1 + 1 * 0.6 + 1 * 0.5 + 1 * 0.4 + 1 * 0.1) = 3.8$
$C(T_6) = (1 * 0.4 + 1 * 0.6 + 1 * 0.5 + 1 * 0.2 + 1 * 0.6 + 1 * 0.5 + 1 * 0.4 + 1 * 0.1) = 3.3$
$C(T_4) = (1 * 0.4 + 1 * 0.6 + 1 * 0.4 + 1 * 0.7 + 1 * 0.5 + 1 * 0.3 + 1 * 0.6 + 1 * 0.5 + 1 * 0.4 + 1 * 0.1) = 4.5$
$C(T_1) = (1 * 0.4 + 1 * 0.6 + 1 * 0.5 + 1 * 0.2 + 1 * 0.6 + 1 * 0.5 + 1 * 0.4 + 1 * 0.1) = 3.3$

Table 4. Displaying the cost estimation by CFT for each test case of individual sequences which are located in FR

	Test cases/cost				
Cycle 1	T_5	T_2	T_6	T_4	T_1
	3.4	3.8	3.3	4.5	3.3
Cycle 2	T_8	T_1	T_5	T_3	T_4
	3.8	4.3	3.8	4.3	3.8
Cycle 3	T_5	T_6	T_2	T_8	T_4
	4.2	3.1	3.6	4.2	3.6
Cycle 4	T_3	T_8	T_2	T_5	T_4
	3.6	3.8	3.8	4.2	3.8
Cycle 5	T_5	T_1	T_{10}	T_2	T_4
	4.2	4.9	4.2	4.9	4.9

This stage concludes by demystifying every test case's final cost, associated with FR, through the aggregation of individual cost of test cases in every particular execution cycle. This cost evaluation indicates each test case's importance as it is directly associated with the weight factor values allocated at the time of structuring the initial population. The higher the cost, the most consequential the test case is. For example, the final cost of T_5 would be:

$T_5 = 3.4 + 3.8 + 4.2 + 4.2 + 4.2 = 19.8$ (Data from Table 4)

Final termination criterion and highly valued test cases from final cost are being considered together for procuring FMT (Fig.9).

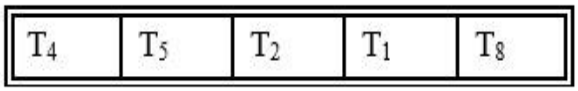


Figure 9. Outcome of the case study considered for reduction mechanism (FMT)

If the final cost of any test case collides with some other test case while selecting it in a finally minimized test suite, then the occurrence factor is considered for those two test cases. The occurrence of a specific test case would be computed by summation of the position of that test case in every execution cycle. If two test cases had the same final cost while selecting, then the test case with the highest occurrence value would be preferable.

The test cases in FMT would result in high coverage compared to the test cases in the test sequences residing in FR. Through the reduction methodology of this proposed approach, the test cases got lessened. Still, the lessened test cases' prioritization would re-order them into a sequence that would be more optimal in detecting more faults at an earlier phase.

Prioritization mechanism for lessened test cases:

The prioritization structure would incorporate some indispensable features and the test cases that got minimized due to the reduction mechanism. These aspects would be the coverage, cost, DU-pair, requirement covered by the test cases, including the historical data that would exploit the same features during the prioritization mechanism, i.e., eight features as a whole. Test cases coverage precisely relies on $S_{covered}$ being elucidated in equation (2) while

cost computation of test cases for prioritization is devised as:

$$Cost(T_i) = \{C(T_i)\} \times 10 \quad \ldots (4)$$

where $C(T_i)$ is effectively defined in equation (3), to be utilized in equation (4), for simplification in calculating decimals, equation (4) is multiplied by 10.

DU-pair is the abbreviation of Definition-Use pair, a dataflow-dependent adequacy criterion, utilizing either predicate-use or computational-use of variable, in a manner that there would have had at least one definition clear path in between the definition and the use of the variable. For instance, let say variable 'a' has a DU-pair [2,6], which would elucidate that '2' in the pair is the statement number defining the variable 'a' and '6' in the pair is the statement number using the variable 'a' (the use could be either p-use or c-use).

Another aspect is the requirement data, being prepared by the desideratum coverage of every test case. For instance, the features could be depicted for a certain code in consideration as:

Coverage Matrix (Statement coverage) =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Cost Matrix =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 11 \\ 23 \\ 25 \\ 21 \\ 29 \end{bmatrix}$$

History Coverage Matrix =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

History Cost Matrix =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 17 \\ 13 \\ 17 \\ 16 \\ 33 \end{bmatrix}$$

History du-pair coverage Matrix =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 2/8 \\ 2/8 \\ 3/8 \\ 4/8 \\ 6/8 \end{bmatrix}$$

History Requirement Matrix =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 11 \\ 21 \\ 24 \\ 29 \\ 48 \end{bmatrix}$$

DU-pair Matrix =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 4/16 \\ 8/16 \\ 2/16 \\ 6/16 \\ 10/16 \end{bmatrix}$$

Requirement Matrix =

$$\begin{matrix} T_1 \\ T_2 \\ T_4 \\ T_5 \\ T_8 \end{matrix} \begin{bmatrix} 30 \\ 44 \\ 46 \\ 52 \\ 49 \end{bmatrix}$$

The FMT test cases acquired five different positions; therefore, every test case in the FMT would have five positions to be re-ordered. Every position possesses some threshold value, which is being fixed at the time of re-ordering the test cases. These threshold values will help during the evaluation, as of which test case would be best suited for which particular possie. These test cases could be arranged in 'n' number of sequences. That is, there could be 'n' combinations regarding the test cases in FMT, but this methodology targets at achieving an optimal outcome that incorporates a highly preferable sequence of test cases with a high possibility of earlier fault detection.

Table 5. Threshold percentile fixed for every possie for the five test cases.

Feature threshold in percentile	Positions of test cases				
	1st position	2nd position	3rd position	4th position	5th position
Coverage	65	85	90	91	92
Cost	25	33	34	35	37
His. coverage	60	75	81	82	90
His. cost	25	34	35	35	35
His. Du-pair	60	81	82	83	90
His. requirement	26	49	50	51	59
DU-pair	51	75	75	80	85
Requirement	45	70	71	72	73

Table 5 delineates the percentile depiction fixed for the five positions. Variation within the threshold values of diverse parameters (present in Table 5) is notable for every time the test case evaluation is done for every specific possie. Different positions would have a distinct or slight equal threshold for every aspect being considered for assessment. This variability in threshold percentile indicates that the significant information from previously ordered test cases is weighed to find out more optimal test cases at subsequent positions. Then, while reckoning a test case for succeeding positions, percentile varies from previous ones.

The du-pair and requirement feature could be estimated through:

$$D(T_i) = d_{covered}/d_{total} \quad \ldots (5)$$

$$R(T_i) = \sum_{j=1}^{m}\{CV(S_j) \times r_j\} \quad \ldots (6)$$

$D(T_i)$ in equation (5) reveals DU-pair Matrix particulars, where $d_{covered}$ is the number of du-pair covered by that test case, and d_{total} is the total number of du-pair. For practicable demonstration, $d_{covered}$ details are taken into account. In equation (6), $R(T_i)$ denotes the test case requirement calculation, and r_j is the requirement value attained by every statement of the cycle in execution.

Input: Test case sequence.
Output: Prioritized test case at '1' position.
Begin:
  Initiate the process with the considered sequence and check the features regarding the threshold for every test case.
  if
    T_i's fulfill the set threshold criteria for every aspect evaluated **then**
    Select those T_i's procuring the threshold in their respective features.
  end
  Choose the T_i common among all the selected T_i's.
  Prioritize T_i to first position.
  Remove that T_i from all the 'n' sequences, proceed with the rest of test cases in all 'n' sequences.
end

Figure 10. Algorithm 5: Test Case Prioritization (Prioritizing the test case at position 1)

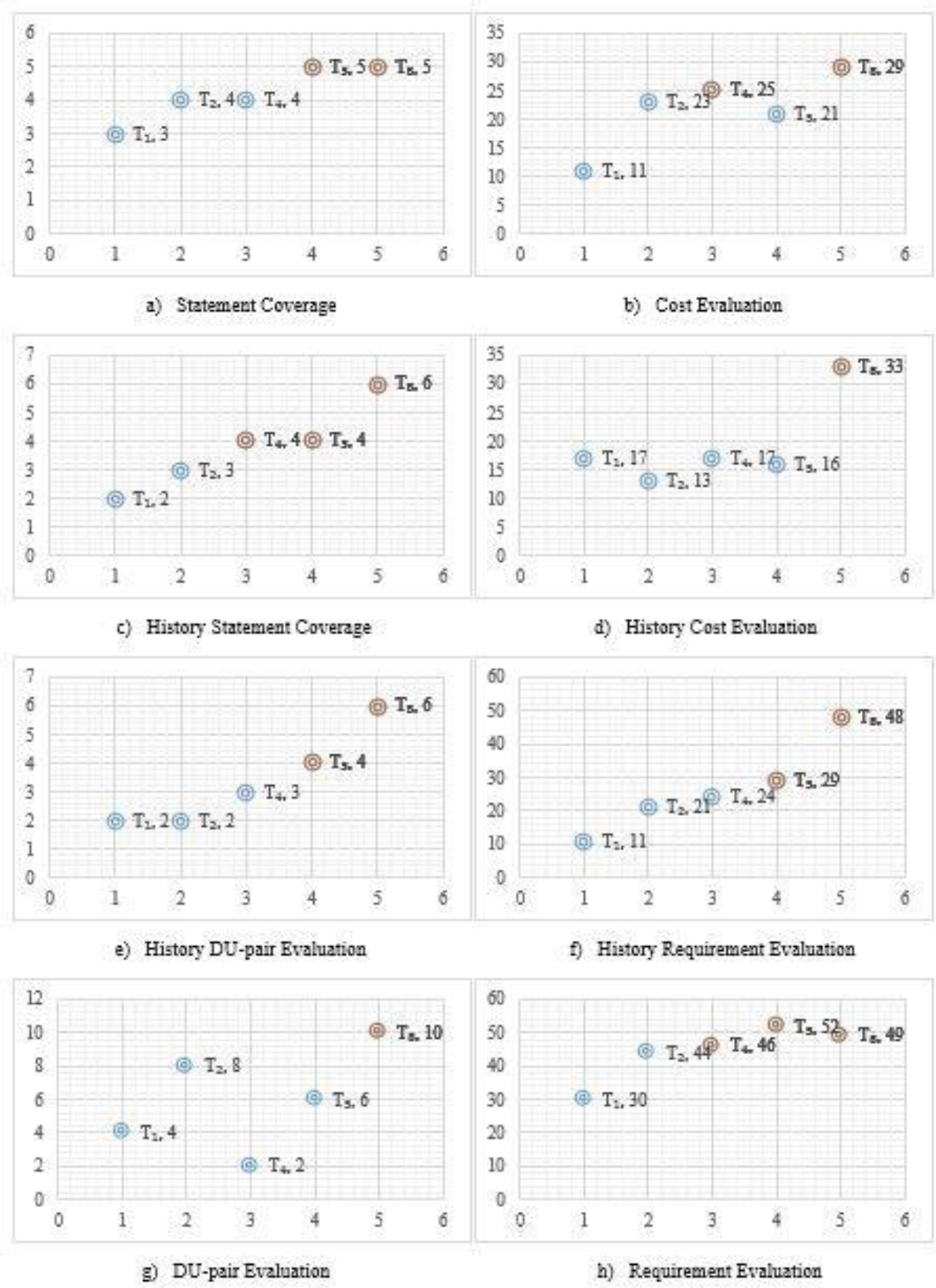


Figure 11. Grid representation of features plotted for prioritizing appropriate test case at position '1'.

Algorithm 5 (Fig. 10) reveals the best-suited test case for position '1' while Fig. 11 depicts the eight features being plotted for position '1', with considered test sequence {T_1, T_2, T_4, T_5, T_8}. The red-colour-highlighted plots elucidate that those test cases satisfy the threshold criteria, which is being set for the first position. Test case T_8 satiated all the aspects and extracted as common among all the test cases meeting the threshold of the features. After being prioritized at the very first, T_8 would then be used as the previous test case for all other T_i's left in 'n' sequences where all ['n' sequences – {T_8}] would be served as input for the second iteration of the prioritization procedure.
For this mechanism, this study took '5' sequences into consideration; therefore, every data of test case T_8 would be utilized for every feature in prioritizing the test cases being left in the sequences, at the specific position. The computational procedure for every aspect present in Table 5 would be updated for the test case being prioritized at the next position to T_8. The equations of the features to be utilized at succeeding positions would be:

$$Cov(T_i) = S_{covered}(T_{i-1}) + T_i n(st) \quad \dots (7)$$
$$Cost(T_i) = Cost(T_{i-1}) + Cost\big(T_i n(st)\big) \quad \dots (8)$$
$$D(T_i) = d_{covered}(T_{i-1}) + T_i n(d) \quad \dots (9)$$
$$R(T_i) = R(T_{i-1}) + R\big(T_i n(st)\big) \quad \dots (10)$$

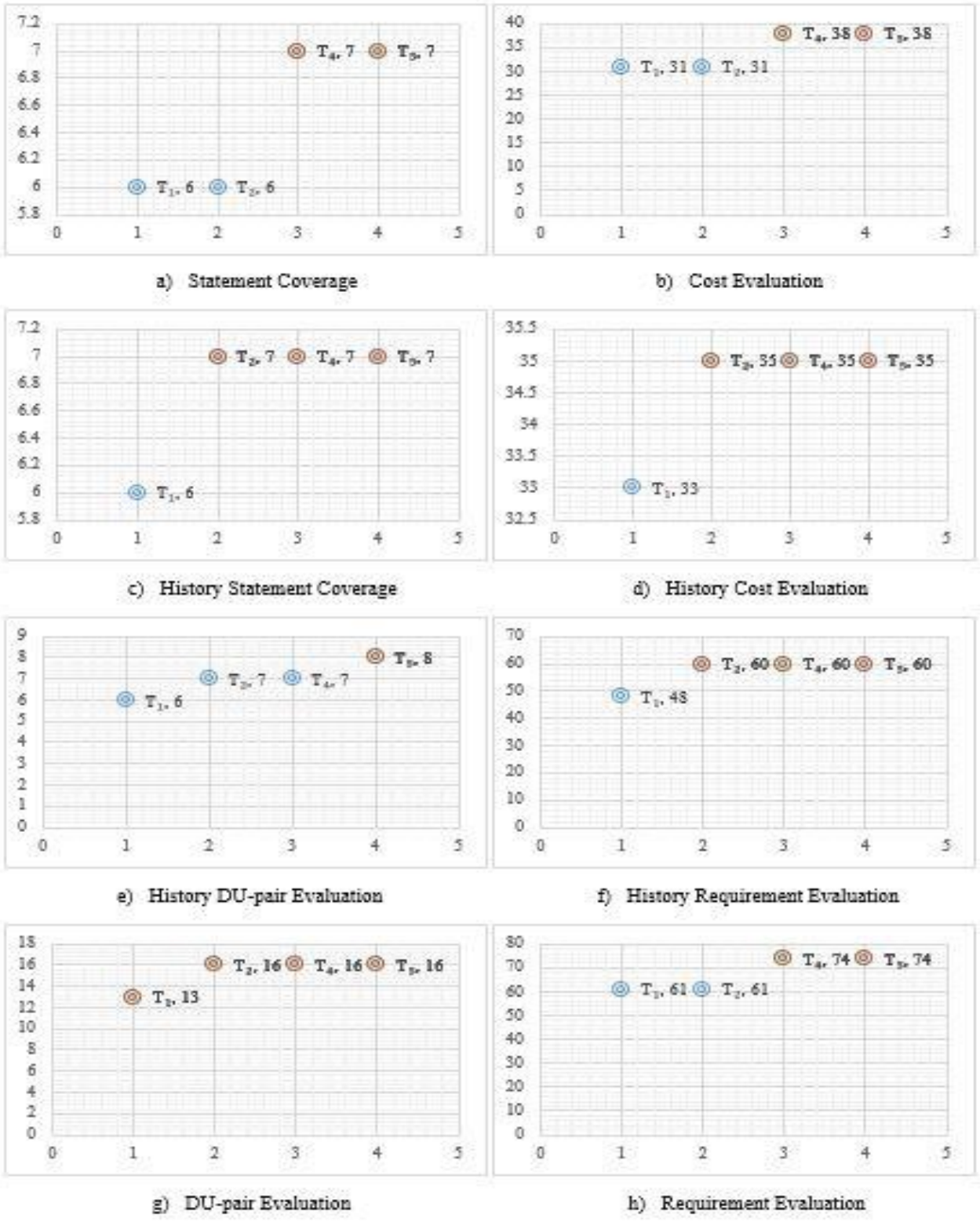
where T_i is the test case in processing, (T_{i-1}) is the previously selected test case; T_in(st) is the number of additional new statement covered by the processing test case concerning the coverage of the previously selected test case, and T_in(d) is the number of additional du-pair covered by the processing test case pertaining to the previously selected test case.

History features will utilize the same equations, i.e., 7-10, for evaluating the integrated features in it. The five sequences would assess all the features respectively, and the test case that is found to be frequent in all the sequences would acquire the succeeding position with reference to the former test case. Some amount of relaxation in the threshold will be expected if the commonality notion is not satiated. This whole procedure for prioritizing the test cases continues until a final optimal sequence of '5' test cases is procured.

The graphs for the sequence {T_1, T_2, T_4, T_5} are portrayed precisely in Fig. 12 that reveals T_5 as the test case to be prioritized at position '2'. Similarly, the three different sequences considered for the study, i.e. {T_2, T_4, T_5, T_1}, {T_4, T_5, T_1, T_2}, {T_5, T_1, T_2, T_4} would be plotted, the fourth sequence comes out to be identical to the first sequence and the common test cases from these sequences extracted. From those extracted test cases, from every sequence, the commonality notion is preferable to have a test case for that processing possie. After affixing test case T_5 at possie '2', test cases T_8 and T_5 would be considered as previous test cases for evaluating the aspects of left test cases (i.e., {T_1, T_2, T_4}) in order to prioritize them at succeeding positions.

Evaluation of left test cases depicts a tie among them for the succeeding positions. Hence, the tied test cases are prioritized further by allocating priority or preference to each considered feature. According to the priority-levels, the individual value of every feature corresponding to the tied test cases is examined. The test case, which would have higher figures for the features, is extracted among the tied test cases and is re-ordered at the appropriate position.

Baghdad Science Journal
P-ISSN: 2078-8665
E-ISSN: 2411-7986



a) Statement Coverage
b) Cost Evaluation
c) History Statement Coverage
d) History Cost Evaluation
e) History DU-pair Evaluation
f) History Requirement Evaluation
g) DU-pair Evaluation
h) Requirement Evaluation

Figure 12. Grid representation of features plotted for prioritizing appropriate test case at position '2'.

Table 6. Preference and feature evaluation for tied test cases.

Preference allotted (Priority)	Features in accordance with the priority assigned	Test cases to be prioritized		
		T_1	T_2	T_4
I	Cost	11	23	**25**
II	History Requirement	11	21	**24**
III	History Coverage	2	3	**4**
IV	History Cost	17	13	17
V	History du-pair	2	2	**3**
VI	Coverage	3	4	4
VII	Requirement	30	44	**46**
VIII	DU-pair	4	8	2

Table 6 explains that T_4 is the test case consisting of at most five features to be high valued while two features with equal value compared with test cases T_1 and T_2, consequently prioritizing T_4 at the third position. A similar strategy would be applicable for the two test cases, i.e., for T_1 and T_2. The final optimal outcome, i.e., the prioritized sequence of test cases, would be:

Figure 13. Final outcome of the proposed methodology

The test cases acquiring at first in Fig. 13 are coverage effective, i.e., more faults would be revealed earlier, and the least coverage effectual test cases are at last in Fig. 13, deducing a notion that more fault detection would be procured at earlier phase only.

Results and Discussion:

Experimental evaluation of the proposed Test Case Minimization methodology:

To validate the significance of the strategy proposed in this study for lessening of test cases, the proposed system was compared with some previously stated and existing algorithms and processes from the studies, i.e., rule-based methodology and conventional genetic algorithm. For experimentation analysis, few benchmark exemplar codes were taken and inspected for the parameters, such as the number of test cases reduced and statements covered by the respective algorithm. The altered version of GA (proposed algorithm) attained up to 80% (maximum percentile) statement coverage, with only 50 % of test cases (Maximum reduction achieved) during analysis (Table 7).

Table 7. Results of experimental analysis for proposed reduction scheme in this study.

Exemplar code for examination	Techniques/Algorithms used	Test cases left after reduction (Out of 10 test cases)	Number of statements covered (in percentage)
Finding maximum among three integers	Rule-based optimization strategy	7	70%
	GA	6	75%
	Proposed Algorithm	**5**	**75%**
Identification of Prime number	Rule-based optimization strategy	7	70%
	GA	8	80%
	Proposed Algorithm	**5**	**80%**
Finding a minimum among three integers	Rule-based optimization strategy	7	80%
	GA	6	70%
	Proposed Algorithm	**5**	**75%**
Factorial code using recursion	Rule-based optimization strategy	7	75%
	GA	7	78%
	Proposed Algorithm	**5**	**80%**
Program to implement Binary Search Algorithm	Rule-based optimization strategy	7	67%
	GA	6	65%
	Proposed Algorithm	**5**	**80%**

Comparative corroboration within the proposed Test Case Minimization methodology:

The test case sequence attained after the reduction procedure (i.e., FMT) (Fig. 9) was compared with the test sequences for the five execution cycles (i.e., with (FR)). According to the case study considered in this research, the embedded test sequences in FR and their graphical comparison are presented in Fig. 14. Three out of five cases have shown promising results regarding the proposed reduction methodology. The outcomes of the comparison between FMT and test sequences of FR comes out to be either higher or equal in three cases for FMT (in terms of cost) (Fig. 14).

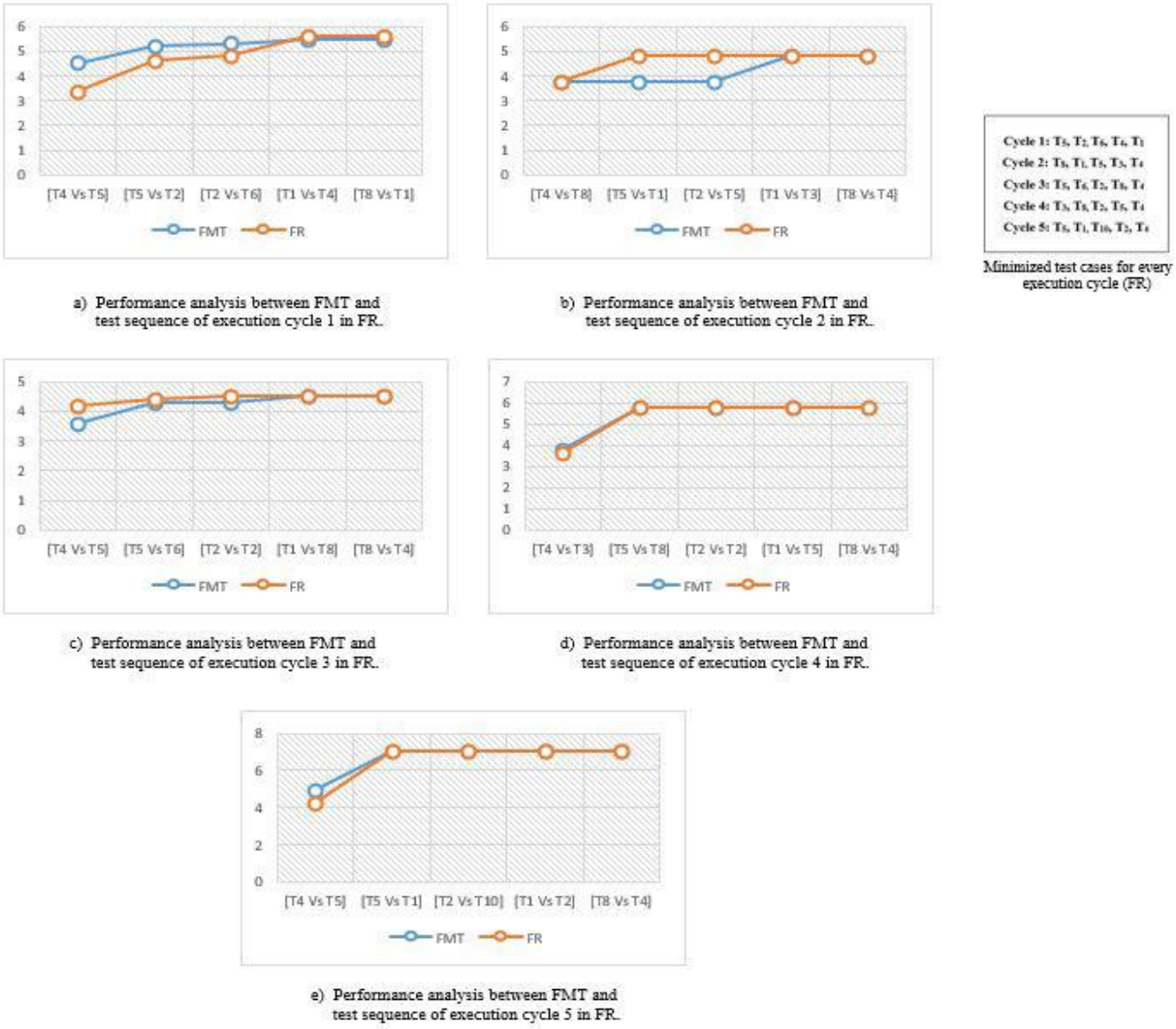


Figure 14. Validation of FMT over FR through graphical representation.

Performance analysis of the proposed Test Case Prioritization scheme:

The prioritized test cases (Fig. 13) were collated with a randomly generated sequence of these '5' test cases, i.e. {T_2, T_1, T_5, T_4, T_8}. This was done to scrutinize the accuracy of prioritized test cases at their specific position, following the threshold set in Table 5. The prioritized test sequence proved to be more scrupulous exact, i.e., highly accurate when collated with a random sequence. The graph below delineates the performance analysis of the two sequences (Fig. 15):

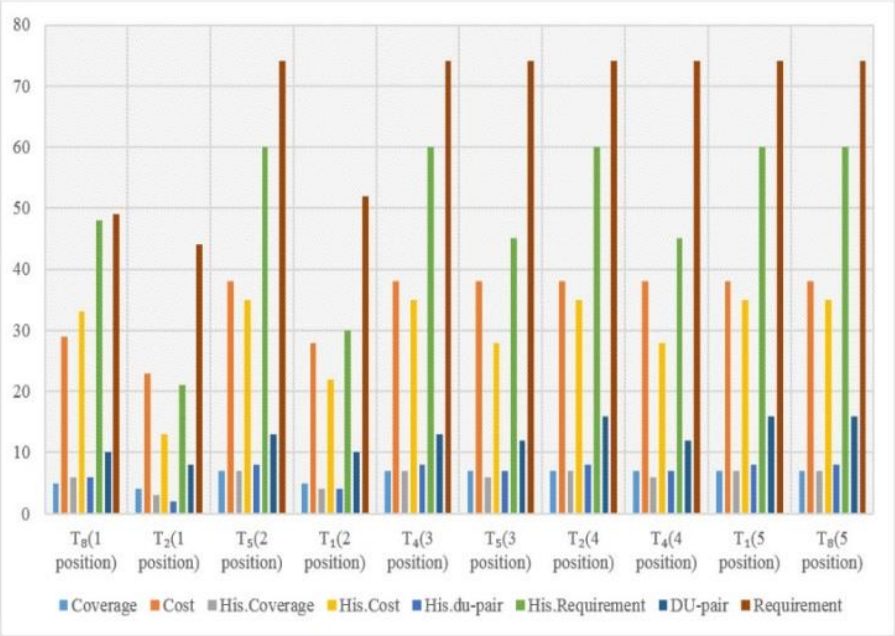


Figure 15. {T_8, T_5, T_4, T_2, T_1} v/s {T_2, T_1, T_5, T_4, T_8}

Experimental evaluation of the proposed Test Case Prioritization scheme:

Effort calculated during this phase of the analysis showed that the test sequence that was achieved as an outcome (Fig. 13) required only 60% of exertion. In comparison, the test sequence that was generated by randomly shuffling the tied test cases with the remaining test cases requires 100% effort (Fig. 15). To understand this nature and verify the system's robustness, an analysis was performed with ten benchmark exemplar codes (Table 8).

Table 8. Results of experimental analysis for proposed prioritization scheme in this study.

Exemplar code for examination	Effort required through the proposed system	Effort needed for random approach	Coverage attained through the proposed system
Finding maximum among three numbers	60%	100%	85%
Factorial program using recursion	**80%**	100%	80%
Fibonacci series	**60%**	100%	90%
String reverse code	**60%**	100%	80%
Finding a minimum among three numbers	**60%**	100%	85%
Calculating Permutation and combination of 2 numbers	**80%**	100%	90%
Identification of Prime number	**80%**	100%	85%
Verifying whether a string is palindrome or not	**60%**	100%	75%
Program to implement Binary Search Algorithm	**80%**	100%	90%
Finding the area of a parallelogram	**60%**	100%	75%

The presented results and analysis suggest the significance of reduction and prioritization methodologies proposed in this study. Earlier studies demonstrated relevant findings regarding the test case optimization problem but lack an appropriate strategic solution for handling the tied test case problem together with optimizing them. Although this current study proposes an optimal solution for test case

reduction and re-ordering problem, there are certain threats to validity.

Earlier detection of fault revealing test cases reduces the cost and effort required for the testing process. For this, the current study suggests an algorithmic framework; however, with increased complexity, supervised machine learning models such as Support Vector Machine (SVM) should be practiced for full automation. These models with kindred algorithms will provide a strong mathematical base (separating hyperplane) to classify faulty and non-faulty test cases. Further, unsupervised machine learning tasks such as clustering will aid in forming the clusters of those test cases that will be going to collide on the same position while prioritizing and hence will provide a futuristic view of test case tie.

An increment in the line of codes will directly affect the length of the population generated; that is, it will increase drastically. Therefore more intelligent optimization methods such as Particle Swarm Optimization (PSO), Grey-Wolf Optimization must be exercised to reduce population or to handle independent paths. Moreover, this study was evaluated with a limited number of programs and can be examined with much more complicated codes to get better insights.

Conclusion:

This paper exhibits a novel strategic approach towards regression test case optimization with certain predetermined objectives. This study proposes an algorithmic rule (basic structure) for processing data obtained from the test case execution history at the earliest stage of the technique. Refinement of this data gives a visual illustration of the existence of relationships among different test cases. These test cases proceed at the reduction stage, where genetic operators are merged with factors that uncover fault and dependency ratios, further, on obtaining the reduced repository, the filtration stage aids in procuring the most favourable test cases.

Additionally, to effectuate this study's objective, the prioritization stage is initiated that discloses the optimal order for the filtered test cases. Analysis records clarified the effectiveness of the proposed system as compared to the randomly generated test sequence.

For future work, the multifactor algorithm propounded in this research can be collated with the concept of artificial intelligence for robustness and can be evaluated with many programs.

Authors' declaration:

- Conflicts of Interest: None.
- We hereby confirm that all the Figures and Tables in the manuscript are mine ours. Besides, the Figures and images, which are not mine ours, have been given the permission for re-publication attached with the manuscript.
- Ethical Clearance: The project was approved by the local ethical committee in SRMIST -201204, Delhi-NCR Campus.

References:

1. Pradhan D, Wang S, Ali S, Yue T, Liaaen M. REMAP: Using Rule Mining and Multi-Objective Search for Dynamic Test Case Prioritization. In 2018 IEEE 11th ICST; 2018; Vasteras. p. 46-57.
2. Gupta N, Sharma A, Pachariya MK. An Insight Into Test Case Optimization: Ideas and Trends With Future Perspectives. IEEE Access. 2019; 7: 22310-22327.
3. Alian M, Suleiman D, Shaout A. Test Case Reduction Techniques - Survey. Int J Adv Comput Sci Appl. 2016 Jun; 7(5): 264-275.
4. Mohapatra SK, Pradhan M. Finding Representative Test Suit for Test Case Reduction in Regression Technique. In 2015 IEEE IC4; 2015; Indore. p. 1-6.
5. Harris P, Raju N. A Greedy Approach for Coverage-Based Test Suite Reduction. Int Arab J Inf Technol. 2015 Jan; 12(1): 17-23.
6. Lin CT, Tang KW, Wang JS, Kapfhammer GM. Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity. Sci Comput Program. 2017 Dec; 150: 1-25.
7. Vahabzadeh A, Stocco A, Mesbah A. Fine-Grained Test Minimization. In 2018 ACM/IEEE 40th Int Conf Softw Eng; 2018; Gothenburg. p. 210- 221.
8. Jeffrey D, Gupta N. Improving Fault Detection Capability by Selectively Retaining Test Cases during Test Suite Reduction. IEEE Trans Softw Eng. 2007 Feb; 33(2): 108-123.
9. Agrawal AP, Choudhary A, Kaur A, Pandey HM. Fault coverage-based test suite optimization method for regression testing: learning from mistakes-based approach. Neural Comput Appl. 2020 Jun; 32: 7769–7784.
10. Singh L, Singh SN, Dawra S, Tuli R. A New Technique for Test Suite Minimization in Regression Testing. SSRN Electron J. 2019 Jan.
11. Lawanna A. Test case design based technique for the improvement of test case selection in software maintenance. In 2016 55th Annu Conf SICE Jpn; 2016; Tsukuba. p. 345-350.
12. Lawanna A. Filtering test case selection for increasing the performance of regression testing. Int J Appl Sci

Technol. 2016; 9(1): 19-25.

13. Panda S, Mohapatra DP. Regression test suite minimization using integer linear programming model. Softw Pract Exp. 2017 Nov; 47(11): 1539-1560.
14. Kazmi R, Jawawi DNA, Mohamad R, Ghani I, Younas M. A Test Case Selection Framework and Technique: Weighted Average Scoring Method. J Telecommun Electron Comput Eng. 2017; 9: 15-22.
15. Marchetto A, Scanniello G, Susi A. Combining Code and Requirements Coverage with Execution Cost for Test Suite Reduction. IEEE Trans Softw Eng. 2019 Apr; 45(4): 363-390.
16. Mukherjee R, Patnaik KS. A Survey on Different Approaches for Software Test Case Prioritization. J King Saud Univ - Comput Inf Sci. 2018 Oct.
17. Beena R, Sarala S. Code coverage based test case selection and prioritization. Int J Softw Eng Appl. 2013; 4(6): 39-49.
18. Zhou J, Hao D. Impact of Static and Dynamic Coverage on Test-Case Prioritization: An Empirical Study. In 2017 IEEE ICST Workshops; 2017; Tokyo. p. 392-394.
19. Mirarab S, Tahvildari L. A Prioritization Approach for Software Test Cases Based on Bayesian Networks. Fundam Approaches Softw Eng. 2007; 4422: 276-290.
20. Carlson R, Do H, Denton A. A clustering approach to improving test case prioritization: An industrial case study. In 2011 27th IEEE ICSM; 2011; Williamsburg. p. 382-391.
21. Zhao X, Wang Z, Fan X, Wang Z. A Clustering-Bayesian Network Based Approach for Test Case Prioritization. In 2015 IEEE 39th Annu Int Comput Softw Appl Conf; 2015; Taichung. p. 542-547.
22. Mahmood MH, Hosain MS. Improving Test Case Prioritization Based on Practical Priority Factors. In 2017 8th IEEE ICSESS; 2017; Beijing. p. 899-902.
23. Khalilian A, Azgomi MA, Fazlalizadeh Y. An improved method for test case prioritization by incorporating historical test case data. Sci Comput Program. 2012; 78(1): 93-116.
24. Gupta A, Mishra N, Tripathi A, Vardhan M, Kushwaha DS. An Improved History-Based Test Prioritization Technique Technique Using Code Coverage. Adv Comput Commun Eng Technol. 2015 Nov; 315: 437-448.
25. Anderson J, Salem S, Do H. Improving the effectiveness of test suite through mining historical data. In MSR 2014: Proc 11th Work Conf Min Softw Repos; 2014. p. 142-151.
26. Noor TB, Hemmati H. A similarity- based approach for test case prioritization using historical failure data. In 2015 IEEE 26th ISSRE; 2015; Gaithersbury, MD. p. 58-68.
27. Goyal S, Mishra P, Lamichhane A, Gandhi P. Software Test Case Optimization Using Genetic Algorithm. Int J Sci Eng Sci. 2018; 1(12): 69-73.
28. Bhawna, Kumar G, Bhatia PK. Software Test Case Reduction using Genetic Algorithm: A Modified Approach. Int J Innov Sci Eng Technol. 2016 May; 3(5): 349-354.
29. Priyanka, Kumar R, Nipur. Generation of optimized and effective test case : A proposed model. Int J Eng Sci Math. 2017 Jul; 6(3): 115-123.
30. Mateen A, Nazir M, Awan SA. Optimization of Test Case Generation using Genetic Algorithm (GA). Int J Comput Appl. 2016 Oct; 151(7): 6-14.
31. Akour M, Abuwardih L, Alhindawi N, Alshboul A. Test Case Minimization using Genetic Algorithm: Pilot Study. In 2018 8th Int Conf CSIT; 2018. p. 66-70.
32. Serdyukov KE, Avdeenko TV. Using genetic algorithm for generating optimal data sets to automatic testing the program code. Inf Technol Nanotechnol. 2019 Jan;: 173-182.

Open Access **Baghdad Science Journal** **P-ISSN: 2078-8665**
2021, Vol. 18 No.2 (Suppl. June) **E-ISSN: 2411-7986**

خوارزمية متعددة العوامل لاختيار الحالة وترتيبها

أتوليا جوبتا **راجندرا براساد ماهاباترا**

قسم علوم وهندسة الكمبيوتر ، SRMIST -201204 ، حرم دلهي-إن سي آر ، غازي أباد ، يو بي ، الهند

الخلاصة :

اختبار الانحدار باهظ التكلفة ، وهو مفهوم التحسين المطلوب. عادةً ما ينتج عن تحسين حالات الاختبار تحديد مجموعة مصغرة أو مجموعة فرعية من حالات الاختبار أو إعطاء الأولوية لحالات الاختبار لاكتشاف الأخطاء المحتملة في مرحلة سابقة. كشفت العديد من الدراسات السابقة عن آلية تعتمد على الكشف عن مجريات الأمور للوصول إلى الأمثل مع تقليل حالات الاختبار أو تحديد أولوياتها. ومع ذلك ، فقد حُرمت تلك الدراسات من إجراءات منهجية لإدارة قضية حالات الاختبار المقيدة. علاوة على ذلك ، غالبًا ما تساعد الخوارزميات التطورية مثل العملية الجينية في استنفاد حالات الاختبار ، جنبًا إلى جنب مع انخفاض متزامن في وقت التشغيل الحسابي. ومع ذلك ، عند فحص قدرة الكشف عن الخطأ مع المعلمات الأخرى ، فإن الطريقة تقصر. يقترح البحث الحالي ، بدافع من هذا المفهوم ، خوارزمية متعددة العوامل مدمجة مع عوامل وراثية وميزات قوية. يتم تقديم مُحدِّد الأولويات المستند إلى العوامل من أجل المعالجة السليمة لحالات الاختبار المقيدة التي ظهرت أثناء تنفيذ إعادة الطلب. إلى جانب ذلك ، تم تضمين جهاز ضبط دقيق يعتمد على التكلفة (CFT) في الدراسة للكشف عن حالات الاختبار المستقرة للمعالجة. يتم تشريح فعالية النتيجة التي يتم الحصول عليها من خلال نهج التقليل المقترح ومقارنتها بطريقة إرشادية محددة (قائمة على القواعد) ومنهجية وراثية قياسية. يتم إجراء التحقق الداخلي للنتيجة المحققة من إجراء التخفيض بيانياً. قارنت هذه الدراسة التسلسلات التي تم إنشاؤها عشوائيًا مع تسلسل الاختبار الذي تم شراؤه المعاد ترتيبه لأكثر من 10 أكواد معيارية لخطة تحديد الأولويات المقترحة. كشف التحليل التجريبي أن النظام المقترح نجح بشكل كبير في تحقيق انخفاض بنسبة 35-40 ٪ في جهد الاختبار من خلال تحديد وتنفيذ حالات اختبار فعالة ومستقرة للتغطية في مرحلة مبكرة.

الكلمات المفتاحية: GA :، اختبار الانحدار ، حالات الاختبار ، تصغير حالة الاختبار ، تحديد أولويات حالة الاختبار.