IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 July 2024, accepted 25 July 2024, date of publication 30 July 2024, date of current version 28 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3435678

METHODS

A Detection-Based Multi-Objective Test Case Selection Algorithm to Improve Time and Efficiency in Regression Testing

ISRAR GHANI[1], WAN MOHD NASIR WAN KADIR[1], (Member, IEEE), ADILA FIRDAUS ARBAIN[1], AND IMRAN GHANI[2]
[1]Faculty of Computing, Universiti Teknologi Malaysia (UTM), Johor Bahru 81310, Malaysia
[2]Department of Computer and Information Science, Virginia Military Institute, Lexington, VA 24450, USA

Corresponding author: Wan Mohd Nasir Wan Kadir (wnasir@utm.my)

This work was supported in part by the Research Management Center (RMC), Universiti Teknologi Malaysia (UTM), and the Ministry of Higher Education Malaysia (MOHE) through the Junior Visiting Researcher (JVR) Grant Scheme under Grant Q.J130000.7128.07E42; and in part by the Fundamental Research Grant Scheme (FRGS) under Grant R.J130000.7828.5F677.

ABSTRACT Regression testing is carried out to ensure that changes or enhancements are not impacting previous working software. Deciding how much retesting is required after modifications, bug fixes or before product deployments are difficult. Therefore, Test Case Selection (TCS) select the satisfactory subset of modified test cases from already executed test suites. The testing primary concerns in TCS for regression testing are efficiency (i.e., coverage, fault detection ability, redundancy) and time. The first challenge in TCS concerns the efficiency of multi-objective test case selection. The second challenge is to improve the execution time to detect the changes in a test suite, which makes it impractical to use these efficiency measures as a single goal for TCS. To overcome these challenges, there is a need to introduce an efficient detection-based multi-objective framework to improve the Time and efficiency of TCS. A multi-objective advanced and efficient regression test case selection (ARTeCS) framework is devised to improve the time performance and efficiency of a given TCS objective relative to the other TCS approaches. An algorithm to detect the changes in test cases using multiple TCS objectives. This comparison found that the enhanced ARTeCS algorithm improves redundancy efficiency by 44.02%. The selection technique showed ARTeCS improved the modified change detection by 43.00%, whereas the Hybrid Whale Optimization Algorithm (HWOA) stated 23% and ACO showed 33% only for selected test cases. Regarding average for fault detection, ACO scores 21%, HWOA scores 11%, and ARTeCS scores 31.08% with total execution times of 12, 21 and 09 seconds, respectively. In conclusion, the multiple-objective ARTeCS framework with four test suite selection parameters is more efficient than the existing multi-objective selection framework.

INDEX TERMS Software testing, regression testing, test case selection, TCS algorithm, TCS framework, multi-objective approach in TCS.

I. INTRODUCTION

Regression testing confirms that previously functioning software continues to function after changing, modifying, or adding functionality [1], [2], [3]. The changes are unavoidable when a software system is evolved, modified or under maintenance [4]. The retesting of such software is time-consuming, complex and difficult to manage. Regression testing ensures the user that newly introduced changes do not produce unintentional behaviour. Regression testing is a very complex and repetitive part of software testing. Minhas et al. Stated that three thousand test cases were subjected to 100 machine hours of regression testing [5]. Regression testing guarantees that no new flaws will be added to the database of expanded code or specification changes [6], [7]. Therefore, regression testing is an important and costly

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

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

activity every time a program is modified [8]. It ensures that modifications do not introduce new bugs into previously validated code. In regression testing, models, frameworks and automated tools are designed and proposed to establish multi-objective test case selection, prioritization and reduction techniques [5]. Models, algorithms, and frameworks are predicated on the relationship between time and efficiency (fault detection [9], redundancy [10], coverage [2]) measures for Test Case Selection (TCS). Therefore, test case selection techniques have the ability for coverage-based, fault detection and redundancy as compared to TCM and TCP [11]. Musa, Sultan introduced a dynamic framework to detect modified changes in the System Under Test (SUT) for TCS [12]. However, regression testing has been widely used to ensure that software evolution does not break existing modules in the system [13]. Efficient test case selection and change detection is a challenge in practice, especially in a world of ever-increasing complexity, distribution, and size of test suite solutions. Thus, the efficient test suite selection targets acquiring an insignificant subset of a test suite that preserves an unambiguous competence standard (e.g., coverage, fault detection, redundancy). For example, some modification steps and change detection have been selected after changes in requirements in the application based on the input value. Change detection is helpful when selecting modified test cases [13], [14]. Regression testing is a rapidly growing concept for software testers, especially in test case selection and change detection [15]. These challenges have led to much work on regression testing techniques. At whatever point an application program is modified for completing any maintenance process, the test suite is planned and selected to verify that the modified parts or detection of modified value of the code work appropriately according to the respective test cases [16], [17]. However, running the full set of regression test suites is essential and expensive because of the large number of test cases and modifications in the existing program [18], [19]. In this paper, we propose a multi-objective test case selection and change detection algorithm (ARTeCS), which detects only modified change based on the input value of a heterogeneous application to improve change detection efficiency and reduce selection time. This study recommended a test case selection method, excluding fault and redundant test cases within a given timeframe. This research also contributes a test case selection and change detection method, which detected only modified change based on input value. (2) We experimented with two open-source ASP.NET applications for this article. Good experimental findings are obtained by evaluating Test Case Selection (TCS) and Change detection (CHD) using acknowledged assessment metrics (failure detection and redundancy) in comparison to two other approaches Hybrid Whale Optimization Algorithm (HWOA) [3], [20] and Ant Colony Optimization (ACO) algorithm [3]. This is how the rest of the paper is organized. The TCS-related work is covered in the second section. The approach suggested in this paper is presented in the third section. Section IV is where the experimental evaluation is carried out. The summary is given in Section V.

II. RELATED WORK PERSPECTIVE

Regression testing is an important but expensive activity every time a program is modified [21]. It ensures that modifications do not introduce new bugs into previously validated code [15]. Complete software testing is not possible, especially in regression testing. The possible way out is adequate testing with certain objectives to be fulfilled. Test case selection and change detection in regression testing are adopted to reduce time and improve efficiency. Associating traditional and composite applications in extensive organizations and associated with critical business-to-business collaborations, where the usually diverse situation demands a severe explanation of the quality product and service interfaces and collaboration patterns. Recent years have seen the introduction of research projects aimed at speeding up the selection of test cases and decreasing testing time [22]. Therefore, certain testing procedures are limited to black-box or white-box techniques and need experiments on medium and large-scale TCS systems.

Additionally, manual testing in small-scale situations is restricted by current TCS techniques. The absence of TCS experiments in newly presented techniques results from ineffective test case selection and change detection of updated test cases in regression testing on a medium (10–20 test cases) or large (30+ test cases) only scale in the literature. An efficient se-lection and change detection in regression testing is emerged to evaluate QoS parameters such as time and efficiency to make a test suite according to the allocated time. The classic TCS techniques [23], [24] for test case selection as a one-time process instead of continuous activity. TCS is repetitive because regression testing is a repetitive activity [1], [25]. The single objective TCS does not fulfil the purpose of test case selection and detection of modified test cases due to the dependency on these efficient measures (Time & efficiency). The test case selection parameters (coverage, redundancy fault detection ability) are used without assessing their impact on the selection process; multi-objective. The redundancy [1], [2], [3], coverage [10], fault detection ability [9] and coverage information are used individually as TCS parameters but not considered simultaneously, as the testing team experience is ignored for test case selection process improvement in existing approaches [26]. The relationship between redundancy, coverage and fault detection ability with time assessment is also ignored in evaluating the test case selection parameters of modified test suite size by current studies [27], [28]. To effectively improve the test suite selection of such heterogeneous and homogeneous scenarios, we need an efficient test case selection and change detection technique using an enhanced change detection approach in the regression. Furthermore, these challenges prohibit testing the functionality and other components of the software [29]. Therefore, testing issues in traditional

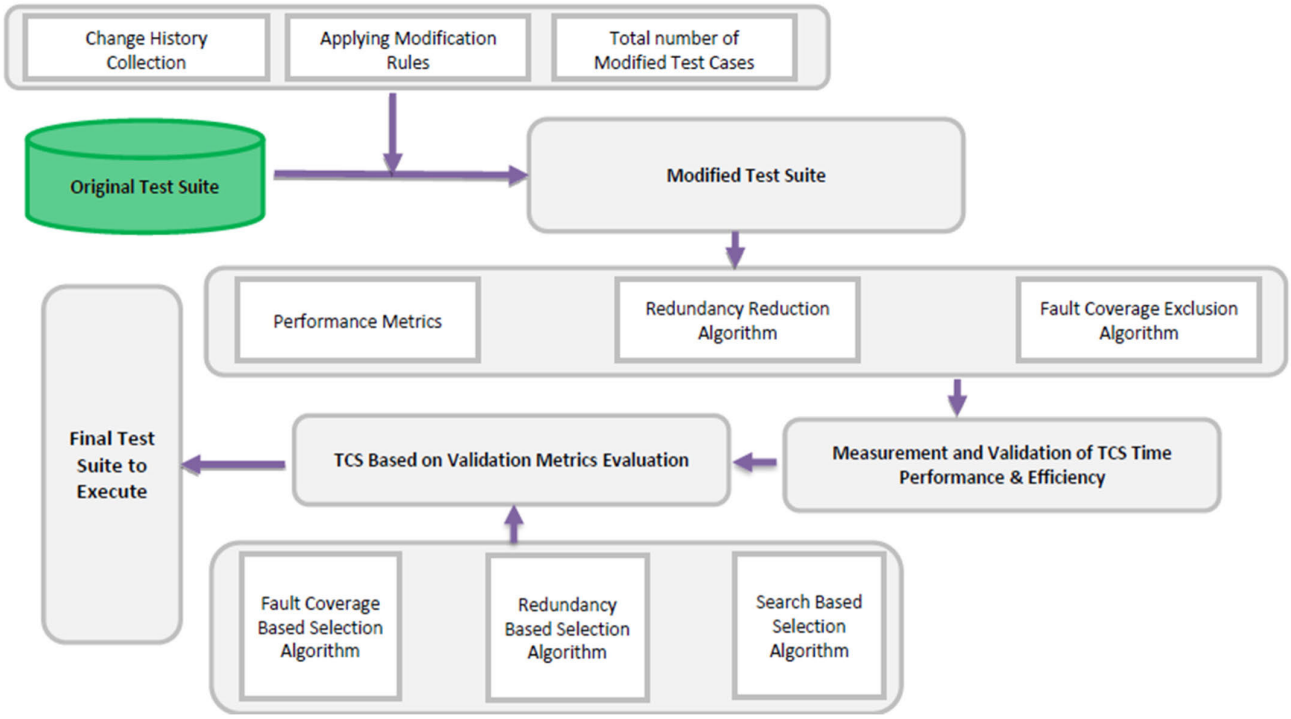


FIGURE 1. Proposed ARTeCS workflow.

and service-oriented applications, like test case selection for interface-to-interface and end-to-end testing, still need to be addressed [30], [31], [32], [33].

III. PROPOSED ARTECS METHODOLOGY

This study minimizes testing time to optimize an efficient regression test case selection and change detection approach for modified test cases. The proposed research also recommends a test case selection technique on the accumulative score of time, coverage, redundancy, fault detection ability and code change information for affected test cases. We present the experimental process setup in Figure 1 for the test case selection framework. Figure 1 depicts a structured process for refining a test suite, starting from an original set of tests and applying a series of modifications and evaluations to arrive at a final, optimized test suite ready for execution. The process begins with the Original Test Suite, from which a Change History Collection (CHC) is performed to understand what has changed. Then, Applying Modification Rules (AMR) leads to the ''Total number of Modified TEST Cases.

A. TEST CASE SELECTION

This section explains the workflow of the proposed Test Case Selection (TCS) algorithm, where the modified cases are compiled into a modified Test Suite (MTS). Once the modified test suite is established, it undergoes further refinement through two different algorithms: the Redundancy Reduction Algorithm (RRA) [34] and the Fault Test Cases Exclusion Algorithm [35]. The redundancy reduction algorithm aims to remove unnecessary test cases that do not add value to the test suite. In contrast, the fault coverage exclusion algorithm eliminates test cases that do not contribute to uncovering faults [36]. Figure 1 represents a flowchart for a test suite optimization process in a software development context to a software testing process. Another component of this workflow is about Change History Collection (CHC) process starts with gathering the change history of test cases, which means collecting data on what has been altered in the software since the last testing cycle where this process includes the code changes, feature additions, or bug fixes. By applying Modification Rules (MR) to detect a modified change in the test cases, modification rules are applied in order to detect modified test cases. These Modification Rules (MR) are predefined criteria where algorithms determine how the existing test cases should be altered to accommodate the recent changes. Original Test Suite before any modifications are made. Another component, ''Total number of Modified Cases'', represents a count or list of the test cases modified after the rules and algorithm were applied. Modified Test Suite (MTS), applying modification rules to the original test suite, resulting in an updated set of tests that reflect the recent changes to the software. From the modified test suite, two paths diverge. Firstly, the Redundancy Reduction Algorithm (RRA) explains the one path leads to a process that eliminates redundant test cases, which helps reduce the time and resources required for testing without compromising the test coverage. Secondly, Fault Coverage Exclusion Algorithm (FCEA) is another path involving an algorithm that excludes

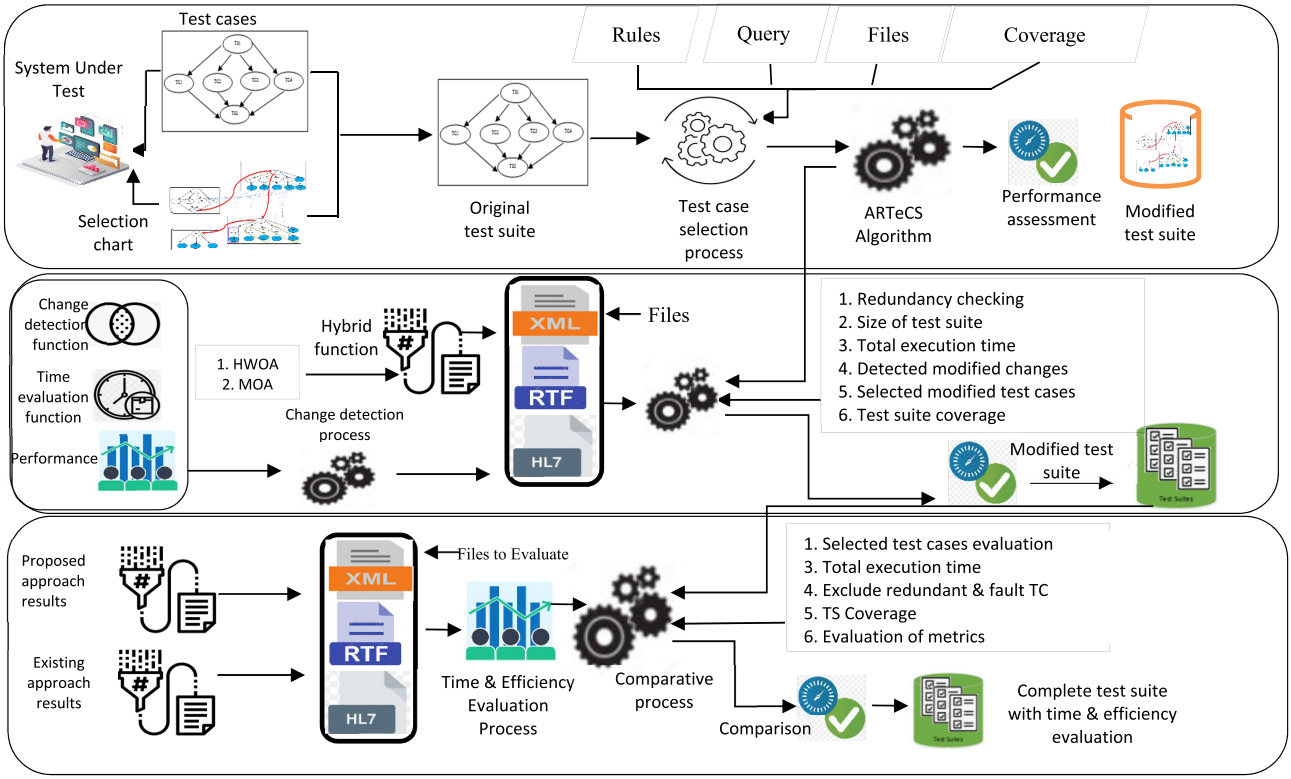


FIGURE 2. Proposed ARTeCS framework.

test cases that do not contribute to fault coverage, which means identifying and removing ineffective tests in discovering new defects.

Both of these paths converge into Performance Metrics that involve evaluating the performance of the test suite post-optimization, ensuring that it is efficient and effective. Final Test Suite to Execute, based on performance metrics, a final set of test cases is selected for execution. The modified test suite is optimized for both efficiency and time. TCS (Test Case Selection) Based on Validation Metrics Evaluation means the final suite is further refined based on additional validation metrics. Within this feedback loop, three algorithms are at play: Fault Coverage Based Selection Algorithm that covers more faults. Redundancy Based Selection Algorithm, focuses on removing duplicate test cases. Search Based Selection Algorithm, uses a heuristic or metaheuristic search technique to select the most modified suitable test cases. The loop completes with Measurement and Validation of TCS Time Performance & Efficiency. Overall, this flowchart outlines a systematic approach to refining a test suite, aiming to make it more focused, less redundant, and more efficient, thereby reducing the cost and time of the testing phase in software development.

The proposed experimental process is described briefly in Figure 1. Based on the proposed process, design and implementation flow are shown for the proposed approach. Generally, this approach can be described as a proposed solution to represent the selection of modified test cases and requirements of products in TCS. The increasing fault detection and redundancy can contribute to complexity in core asset development. In TCS, it can be interpreted as the correct & modified test cases are selected based on the modification.

Figure 1 also depicts a structured process for refining a test suite, starting from an original set of tests and applying a series of modifications and evaluations to arrive at a final, optimized test suite ready for execution. The process begins with the Original Test Suite, from which a Change History Collection (CHC) is performed to understand what has changed. Then, Applying Modification Rules (AMR) leads to the "Total number of Modified Cases. The modified cases are compiled into a Modified Test Suite (MTS). Once the modified test suite is established, it undergoes further refinement through the Redundancy Reduction Algorithm (ACO) and Fault Coverage Exclusion Algorithm (HWOA) [3]. The redundancy reduction algorithm aims to remove unnecessary test cases that do not add value to the test suite, while the fault coverage exclusion algorithm eliminates test cases that do not contribute to uncovering faults.

Figure 2 explains the proposed research framework, which includes code changes, feature additions, and bug fixes. These are probably predefined criteria and algorithms determining how the existing test cases should be altered to accommodate the recent changes. Original Test Suite before any modifications are made. Total number of Modified Cases, represents a count or list of the test cases that have been modified after the

rules were applied. Modified Test Suite, applying modification rules to the original test suite, resulting in an updated set of tests that reflect the recent changes to the software. From the modified test suite, two paths diverge. Redundancy Reduction Algorithm, one path leads to a process that eliminates redundant test cases, which helps in reducing the time and resources required for testing without compromising the test coverage. The fault Coverage Exclusion Algorithm is another path involving an algorithm that excludes test cases that do not contribute to fault coverage, which means identifying and removing tests that are ineffective in discovering new defects. Both of these paths converge into Performance Metrics that involve evaluating the performance of the test suite post-optimization, ensuring that it is efficient and effective. Final Test Suite to Execute, based on performance metrics, a final set of test cases is selected for execution. The modified test suite is optimized for both efficiency and time.

TCS (Test Case Selection) Based on Validation Metrics Evaluation means the final suite is further refined based on additional validation metrics. Within this feedback loop, three algorithms are at play: Fault Coverage Based Selection Algorithm that covers more faults. Redundancy Based Selection Algorithm, focuses on removing duplicate test cases. Search Based Selection Algorithm, perhaps uses a heuristic or metaheuristic search technique to select the most modified suitable test cases. The loop completes with Measurement and Validation of TCS Time Performance & Efficiency. Overall, this flowchart outlines a systematic approach to refining a test suite, aiming to make it more focused, less redundant, and more efficient, thereby reducing the cost and time of the testing phase in software development. The proposed framework to algorithm mapping approach aims to validate features with coverage to ensure that TCS and change detection (CHD) test cases achieved higher coverage in selecting modified test cases according to the Advanced Regression Efficient Test Case Selection ARTeCS framework. To accommodate this challenge, there is a need to establish the co-relation between coverage, time, redundancy and fault detection ability with a framework for original and modified test suites, which provides the basic mechanism and an algorithm to find the accumulative impact of highlighted measures. In this part, selection is performed based on change detection to obtain only modified test cases.

The dependencies and mapping are used to process relevant information (history, rules, mapping) from the input and to structure them into frames using the framework. The system has been implemented using the Eclipse Integrated Development Environment (IDE). XML, RTF and HL7 files have been used to extract the modified test cases and evaluate the performance between existing & proposed approaches. Detection of modified changes based on the redundancy or similarity in the modified test suite and similarity function in ARTeCS excludes the redundant test cases from the modified test suite.

Figure 2 illustrates a comprehensive process for evaluating and optimizing test cases within a system under Test. The process begins with selecting the system under Test, followed by creating an original test suite. This suite undergoes a test case selection process facilitated by the ARTeCS algorithm, which results in a performance assessment and the generation of a modified test suite. The next phase involves a change detection function, utilizing hybrid functions such as HWOA and ACO to identify changes in the system. This is followed by a time evaluation function to assess performance. The change detection process outputs files in various formats (XML, RTF, HL7), which are then used to check for redundancy, size, total execution time, detected changes, selected modified test cases, and test suite coverage. The result is a further refined modified test suite. In the final phase, the proposed approach results are compared with existing approach results through a time and efficiency evaluation process. This involves evaluating selected test cases, total execution time, excluding redundant and faulty test cases, test suite coverage, and other metrics. The comparative process culminates in a complete test suite with time and efficiency evaluations, ensuring a thorough and optimized testing process.

B. DEFINITIONS OF TCS FRAMEWORK AND ALGORITHM

This research also defines a formal framework to ensure that the developed and implemented ARTeCS algorithm is better understood for TCS. After the framework has been developed, the formal definitions are used to represent the abstraction of the ARTeCS algorithm. This part comprised the definitions of the redundancy and fault test case detection requirements and the developed algorithm. The ARTeCS has been developed based on the proposed product configurations extracted from the proposed framework. The formal method of the proposed framework is described as follows: Definition: Assume that a test suite T has n test cases, each of which is referred to as a change that reveals test cases for P and P′. Using a test case selection procedure M, m test cases are chosen from n test cases total. Total Inclusivity = (M/n) * 100 (3.1) Case 1: If n*0, the percentage shows the inclusiveness of the test suite. Case 2: if n = 0, the inclusiveness is considered 100 %, all modifications revealing. For instance, suppose that an examination set of thirty test cases makes up T for program P. Eight modification-revealing test cases for P and P′ are contained in the T, which is a test case selection method. Out of the eight test cases that show modifications, M chooses two. In this instance, M's inclusivity will be 25% for both P and T. M is considered safe if M chooses all modifications exposing test cases for P to P in T.

The selected subset of modification-revealing test cases is displayed when determining the inclusiveness of a selection strategy, indicating whether it is safe or unsafe. To overcome this problem, Table 1 shows the relationship between TCS and CHD mapping and co-relations of evaluation metrics. Thus, it requires the derivation of modified test cases based on features that link with the modified test suite used to represent the modification of the TCS. This is because the representation of the mapping framework cannot reflect the semantics of fault detection and redundancy. A mapping of the TCS

TABLE 1. Co-relationship of evaluation metrics.

Change detection relationship	Description
Dependency of feature	It indicates that the s1 is SUT since f1 is
Fault detection	It represents that the test suite3 s3 depends on the test suite2 s2. Since s3 is detected, s2 will be selected.
Similar of states	It signifies that the naming of test cases f1 and suite s1 is similar
Coverage feature	It means that all modifications from f1 and s1 differ or are similar to each other. However, the tester still requires this mapping for the selection.
Detection type	It signifies that only modified changes have been included in selected s3 is detected, s3

framework component with the requirements algorithm is essential to produce the efficient and modified test suite.

TABLE 1 describes the various types of change detection relationships and their corresponding descriptions. Each relationship highlights a specific aspect of how changes in features or test suites are detected and managed.

Dependency of feature: This indicates that the system under Test (SUT) is represented by s1, as feature f1 is involved.

Fault detection: This represents that test suite s3 depends on test suite s2. If s3 is detected, then s2 will also be selected.

Similarity of states: This signifies that the naming of test cases f1 and suite s1 is similar, indicating a close relationship or potential redundancy.

Coverage feature: This means that all modifications from feature f1 and suite s1 may differ or be similar to each other. However, this mapping is still necessary for the tester to make an informed selection.

Detection type: This indicates that only the modified changes included in the selected test suite s3 are detected, ensuring that s3 is up-to-date with the latest changes.

The selected subset of modification-revealing test cases is displayed when determining the inclusiveness of a selection strategy, indicating whether it is safe or unsafe. To link original and modified test suites, a redundancy process is needed since the research aims to validate modified test cases with the requirements using 100-index scaling values that have been used to understand this highlighted objective. Values less than 25 have been considered a very low score. The values ranging from 0 to 25 are considered low scores within factors in comparison. The values ranging from 25 to 50 are described as a medium score with comparing factors. The value ranges from 50 to 75 is considered a high score within comparing factors. The values range from 75 to 100 and are considered very high within comparing factors, as shown in Table 2.

Table 3 provides information about three different datasets, each identified by a unique name. It includes details on the version number, lines of code (LOC), and the number of test cases associated with each dataset.

TABLE 2. 100 index-values.

100-Index-Scale	
0 to 25	Very Low
25- to 50	Low
50 to 75	Medium
75 to 100	Very High

TABLE 3. Dataset for experiment.

NO	DATASET	Version	LOC	Test cases
1	Joda Time	3	280464	279
2	Tree Data Structure	3	1173	189
3	Triangles	3	678	93

Joda Time: This dataset is at version 3, containing 280,464 lines of code and 279 test cases. Tree Data Structure: Also at version 3, this dataset is considerably smaller, with 1,173 lines of code and 189 test cases. Triangles: This dataset, like the others, is at version 3 and consists of 678 lines of code and 93 test cases.

IV. FRAMEWORK TO ALGORITHM IMPLEMENTATION

This section provides the formal word count written in text for modified test cases process of mapping in terms of formal method, the process of implementation of the algorithm into.xml [37] and HL7 [38] (Health Level 7) format files and the algorithm repository. HL7 files are usually used for EDI (Electronic Data Interchange) in healthcare IT systems to trace modified changes in the existing file based on the original file. To conduct the mapping between the original and modified test suite, a framework-to-algorithm implementation is conducted and includes the transformation to produce the output in.xml and HL7 file formats, which are text-based objects. Any changes in the algorithm must be reflected in.xml and HL7 files. The tester needs to regenerate.xml or HL7 files for every change to ensure the mapping result is updated based on the latest framework. In this process, the mapping selection acts as a selector to run the mapping process based on the original file. Therefore, statement coverage is selected from HL& and.xml files for the TCS technique. The fault detection is used as an adequacy measure instead of a selection parameter in classic TCS techniques using HL7 and.xml files. The proposed framework and TCS technique use mutation score as fault detection ability. The modifications in test cases are the primary feature of all available TCS techniques. The proposed framework used statement changes as modification parameters because statement coverage was used as a coverage measure in HL7 &.xml files. The next scenario is a balanced scoring method to combine time, coverage, fault detection ability and redundancy with equal weights using HL7 and.xml. Therefore, the

bottleneck is that the problem needs to be solved multiple times for multiple solutions, which can be addressed using equations 1.

$$W1^{*}C + W2^{*}E + W3^{*}U \tag{1}$$

where: C represents code coverage (the percentage of code exercised by the test case). E represents the execution time of the test case. U represents the number of times the code under Test has been changed since the last execution of the test case. W1, W2, and W3 are weights assigned to each factor, indicating their relative importance. This equation combines code coverage, execution time, and code change (how frequently the code changes) to select test cases that provide good coverage while minimizing execution time and focusing on recently changed code. A Set is used to indicate a feature set covered by a test suite Tc, and Cov(t) to signify the set of features covered by the test case t ∈ Tc, then:

A test case is considered redundant in Tc if Cov(Tc/{t})= Set,

A test case ti is considered redundant with respect to ts if Cov(ts) ⊆ Cov(ts).

For the purpose of selection, single-feature coverage has been used as a target test coverage. A test Ts ∈ Tc can be considered:

Definition 1: Ts is totally redundant of Tc, if ∃Tj ∈ Tc, i = s, Cov(Tc)⊆ Cov(Tj).

Definition 2: Ts is partially redundant of Tc, if ∃Ts ∈ Ts, s = j, Cov(Ts) = Cov(Tj) and Cov(Ts) ∩Cov(Tj) = ∅.

Definition 3: Ts is unique, if ¬∃Tj ∈ Ts, s= j, Cov(Ts) = Cov(Tj).

The advantage of this multi-objective weighted sum is its linear and simple implementation. The main benefit is its usage as a single objective assignment into the objective function, and that objective needs no or minimum modification and constraint set. The constraints are also defined as part of the objective function with weighted average scoring. There are also minimum computations required to calculate the solution as compared to other GA-based optimization techniques [39]. The proposed framework and TCS technique used a slicing technique for modification identification, test suite data and multi-objective weighted average sum to design the solution. This technique avoids the drawbacks and includes the benefits of these contributing methods. This is worth mentioning that in this solution, GA is not used to find the optimal solution; only the weighted average scoring method was borrowed. The reason behind this selection is that GA finds all possible solutions from search mapping before and after modification in given test suites, while the proposed technique works only for a single solution each time it is invoked.

A. PROPOSED TCS METHOD

In this segment, the proposed TCS algorithm comes from a textual format representing the results returned by a TCS query. This view of the algorithm covers two types of information: the list of original and modified test suites that link each other. This view helps the tester to further refine the modified test cases based on their redundant and fault-based test cases with new query execution needed. This process will loop until the tester reaches the desired level of TCS. To implement TCS, queries, view and mapping rules, we have implemented TCS algorithm 1 and Figure 3 to exclude fault and redundant test cases.

Algorithm 1: Advanced Regression Test Case Selection Algorithm (ARTeCS)

```
1: Inputs:
2: Test[] The set of all test cases from an object of analysis.
3:  Query[] The set of program changes between two versions.
4:  Output:
5:  SelectionSet[] The set of selected test cases for execution.
6: Declare:
7: S The minimum similarity score of test cases to be selected for
query matching.
8: N The percentage of the tests to be selected for each query.
9: cap The number of tests to be selected.
10: src The source test is to be selected first.
11: nn The nearest Test in the RecomSet with min dist of src.
12: RecomSet[]  The set of tests that are similar to a specific query.
13: procedure TEST SELECTION(T est [], Query [])
14: for each query q ∈ Query[] do
15: for each test tc ∈ Test [] do
16: CalculateCosineSimilarity(t, q);
17:       if CosineSimilarity(tc, q) >= S & tc /∈ RecomSet[] then
18:          RecomSet.Add(tc);
19: Else if RemoveFaultCases(t, q) >= S & t /∈  RecomSet[] then
20:          RecomSet.Add(tc);
21: end else if
22:                                    end if
23: end for
24: end for
25: cap ← N * (RecomSet[].Size());
26: for each tc ∈ RecomSet[] do
27:       if t.hasFailed() & SelectionSet.Size() < cap then
28: SelectionSet.Add(tc);
29: RecomSet.Remove(tc);
30: end if
31: end for
32: src ← SelectionSet[0];
33: while SelectionSet.Size() < cap do
34: nn = Min Diff(src, RecomSet[]) ;
35: SelectionSet.Add(nn);
36: src = nn;
37: RecomSet.Remove(nn);
38:        end while
39:      return SelectionSet[];
40: end procedure []
```

FIGURE 3. Proposed ARTeCS algorithm.

Algorithm 1 in Figure 3 explains the overall objective is to reduce redundancy using the similarity function; the first goal of this study is to identify and extract the appropriate regression testing features for modality representation to enhance the regression testing approach to reduce redundancy. A similarity-based method is proposed to select the modified test cases and efficiently detect the newly added change to minimize redundancy. This function explains the detection of duplicate test cases based on string matches using the weightage average score technique.

Algorithm 2: Redundancy Removal

1. **Input:** test suite T Requirement coverage information to detect similarity for each test case in T, for testing criteria $C_1,C_2,\ldots,C_k$ (k >= 2)
2. **Output:** RS: a reduced set of test cases from T that satisfies all testing requirements for the k criteria algorithm ReduceWithSelectiveRedundancy
3. RS:= {};
4. for each criterion C_i,1<=i<=k label all associated testing requirements as unmarked;
5. **while** T is not empty do
6. R_S:= R_S ∪ {next(T)};
7. T:= T - {next(T)};
8. **for each** criterion C_i,1<=i<=k label as marked w.r.t.Ci for inclusion in R_S;
9. **redundant**= the set of test cases from T that have just become redundant w.r.t.C_i;
10. T:= T - redundant;
11. SelectRedundantTests(R_S,redundant,C_i);
12. **endwhile**
13. **return** RS;
14. **end** ReduceWithSelectiveRedundancy
15. **function** SelectRedundantTests(R_S,redundant,C_i)
16. **while** redundant is not empty do
17. toAdd:= the test case in redundant contributing maximum additional C_i coverage to R_S;
18. R_S:= R_S ∪ {toAdd};
19. **redundant**= redundant - {toAdd};
20. **for each** criterion C_j,1<=j<=k
21. **if** k then reLabel as marked the testing requirements satisfied by to Add;
22. redundantAgain:= the set of test cases from redundant that have just become redundant w.r.t.C_i;
23. **redundant**= redundant - redundantAgain,C_i+1);
24. **endif**
25. **endwhile**

FIGURE 4. Redundancy exclusion function.

Algorithm 2 in Figure 4 explains the overall objective is to detect and exclude fault test cases using the remove redundant function. The first goal of this study is to identify and extract the appropriate regression testing features for modality representation to enhance the regression testing approach to reduce redundancy. A fault removal-based function is proposed to select the modified test cases and efficiently detect the newly added fault to the modified test suite.

Algorithm 3 in Figure 5 The algorithm titled "Remove Fault Test Cases" aims to refine a given test suite SSS by identifying and eliminating faulty test cases, producing a subset S(F)′ S(F)'S(F)′. Initially, the subset S(F)′ S(F)'S(F)′ is empty, and the algorithm iterates through each test case d_{id} from a fault test suite S(F)S(F)S(F), which is built from the original or enhanced test suite SSS. During each iteration, the algorithm determines whether d_i is valid based on specific conditions, such as whether the input is valid and the output is as expected. Depending on the outcome, the algorithm either retains the test case in the fault test suite, discards it, or adds it to the subset S(F)′ S(F)'S(F)′. The algorithm continues this process until all test cases have been evaluated, ensuring that only non-faulty test cases are included in the final subset S(F)′ S(F)'S(F)′. The final output is a refined test suite that excludes invalid or faulty test cases, thereby enhancing the overall quality and reliability of the test suite.

Algorithm 3: Remove Fault Test Cases

1. **SelectTestCase** (S, S(F), ValidInput. ValidOutput) : S(F)'
2. **Input**:S. S' the original/enhanced test suite,
3. S(F) A fault test suite created to test S
4. **Output**: S(F)' - Subset of S(S') selected for executing S(F)
5. **begin**
6. S(F)'=0
7. i= 1
8. B = Build SF from S
9. d = Detect fault test cases from S(F)
10. **while** NOT end of log file(S(F))
11. **begin**
12. select d_i
13. **Case S_1** (ValidInput = FALSE) and (ValidOutput = TRUE):
14. **begin**
15. B_{si} = build SF from S
16. If InvokeProcedureCall(S, S_{si})a
17. **break;**
18. else if NonFaultTestCase (d_i, S(F)' b)
19. **break:**
20. else S(F)' = S(F)' + d_i
21. end
22. **Case S_2** (ValidInput = FALSE) and (ValidOutput = TRUE):
23. **begin**
24. if NonFault TestCase (d_i,S(F), S(F)'b)
25. **break;**
26. else S(FY = S(F) + d_i,
27. **end**
28. **Case S_3** (ValidInput = FALSE) and (ValidOutput = TRUE):
29. **begin**
30. d(S) = d(S)' + d_i
31. **End**
32. i= i+1
33. **end**
34. **Return S(S)'**
35. **end**

FIGURE 5. Proposed ARTeCS algorithm.

B. PROPOSED CHD METHOD

This section begins with trade-off issues related to test case selection (total execution time) with coverage of Change Detection (CHD). This is because the existing studies ignore the measurements for multi-objective test case selection, which caused unbalanced results based on time and efficiency. It covers the hybridization of two algorithms, The Hybrid Whale Optimization Algorithm (HWOA) and A Hybrid Algorithm for Multi-objective change detection has been enhanced to detect modified change efficiently by excluding fault and redundant test cases from the modified test suite. A multi-objective algorithm is enhanced to detect modified change with high coverage with a minimal time of execution. To evaluate the enhanced algorithm, two benchmark scenarios, which are the Search Based Function (SBF) and the slicing approach, were adopted to select test cases from the multi-objective ARTeCS algorithm. Here, the essential elements of the Advanced Regression Efficient Test Case Selection (ARTeCS) algorithm for CHD will be further discussed along with the slicing approach adopted for change detection to exclude fault and redundant test cases to improve Time for TCS.

Algorithm 4: Change Detection Algorithm (CHD)

```
1. Algorithm: SelectTestCase (S, S (F), ValidInput. ValidOutput) :
S(F)'
2. Input: S. S' the original/enhanced test suite,
3. Input: S. T' the original/enhanced test suite selection time,
4.          S(F) A false test suite created to test S
5.  Output: S(F)' - Subset of S(S') selected for executing S(F)
6.       begin
7.          S(F)'=0
8.            i= 1
9.            B = Build SF from S
10.            d = Select false test cases from S(F)
11.                  while NOT end of log file(S(F))
12.       begin
13.             select di
14. case S1 (ValidInput = FALSE) and (ValidOutput = TRUE):
15.        begin
16.           Bsi = build SF from S
17.             If InvokeProcedureCall(S, Ssi)a
18.            break;
19.             else if NonRedundantTestCase (di, S(F)' b)
20.             break:
21.            else S(F)' = S(F)' + di
22.end
23. case S2 (ValidInput = FALSE) and (ValidOutput = TRUE):
24.       begin
25.         if NonRedundantTestCase (di,S(F), S(F)'b)
26.               Searchbasedfuntion(S, SF);
27.    if Searchbasedfuntion (SF, q) >= S & t /∈ RecomSet[] then
28.        RecomSet.Add(t);
29.break;
30.              else S(FY = S(F) + di,
31.end
32.  case S3 (ValidInput = FALSE) and (ValidOutput = TRUE):
33.           DetectRendundntFaultTimebasedfuntion(S, SF);
34.if DetectRendundntFaultTime (SF, q) >= S & t /∈ RecomSet[]
then
35.        RecomSet.Add(t);
36.
37.         begin
38.       d(S) = d(S)' + di
39.       End
40.         i= i+1
41.   end
42.    Return S(S)'
43.    Return S(T),
44.end
```

FIGURE 6. CHD algorithm.

Algorithm 4 in Figure 6 aims the ''Change Detection Algorithm (CHD)'' is designed to refine a test suite SSS by identifying and removing false test cases, generating a refined subset S(F)′ S(F)'S(F)′ for executing the false test suite S(F)S(F)S(F). The algorithm begins by initializing S(F)′ S(F)'S(F)′ as empty and iterates through each test case did_idi from S(F)S(F)S(F). For each test case, the algorithm checks conditions related to valid input and output. In the first case (S1), if the input is invalid and the output is valid, it builds a new fault test suite S(F)S(F)S(F) from SSS and invokes a procedure call. If the call fails and the test case is non-redundant, the test case is added to S(F)′ S(F)'S(F)′. In the second case (S2), if the test case is non-redundant, a search-based function is invoked to check if the test case should be added to a recommendation set. If not, the test case is added to S(F)S(F)S(F). A time-based function detects redundant faults in the third case (S3). If a test case is found to be redundant and should be recommended, it is added to the recommendation set. Throughout the process, the algorithm continues to detect and handle redundant and non-redundant test cases, refining S(F)′ S(F)'S(F)′ by removing false test cases and enhancing the test suite's accuracy and efficiency. The final output includes both the refined test suite S(F)′ S(F)'S(F)′ and the recommendation set S(T)S(T)S(T).

Figure 7 presents a flowchart detailing a systematic approach to evaluating and optimizing test cases for a system under Test. The process begins with selecting the system under Test, followed by creating an original test suite. This suite is then subjected to a test case selection process, which is enhanced by the ARTeCS AI algorithm, leading to a performance assessment and the generation of a modified test suite. Subsequently, a change detection function is employed, utilizing hybrid functions such as HWOA (Hybrid Whale Optimization Algorithm) and MOA (Multi-Objective Algorithm), in identifying any changes in the system. A time evaluation function complements this to assess the performance impact. The change detection process produces output files in various formats (XML, RTF, HL7), which are then analyzed for redundancy, size, total execution time, detected changes, selected modified test cases, and test suite coverage. This analysis results in a further refined modified test suite. In the final stage, the results of the proposed approach are compared with those of existing approaches through a comprehensive time and efficiency evaluation process. This involves assessing selected test cases, total execution time, excluding redundant and faulty test cases, and evaluating test suite coverage and other relevant metrics. The comparative analysis culminates in a complete test suite that has been thoroughly evaluated for time and efficiency, ensuring an optimized and effective testing process.

Figure 7 explains the proposed flowchart and depicts the process of optimizing a test suite using the CHD (Change, Detection, and Exclusion) algorithm. It begins with defining the scope of test suite coverage, followed by the creation of a test suite execution plan. Changes in the detection process are implemented using the CHD algorithm. Execution of the CHD algorithm leads to a decision point: if faults and redundant test cases are found, the process branches into two parallel paths where a fault detection algorithm is applied to exclude faulty test cases and a redundancy detection algorithm is used to exclude redundant test cases. If no faults or redundancies are found, the execution time is calculated. Finally, the refined test suite, with excluded faulty and redundant test cases, is ready for execution.

In order to perform change detection, ARTeCS divides T into several groups using a clustering technique. The basic tenet of ARTeCS is that test cases with comparable behaviour characteristics can be clustered together. Figure 7 illustrates the proposed change detection algorithm by detecting fault and redundant test cases.

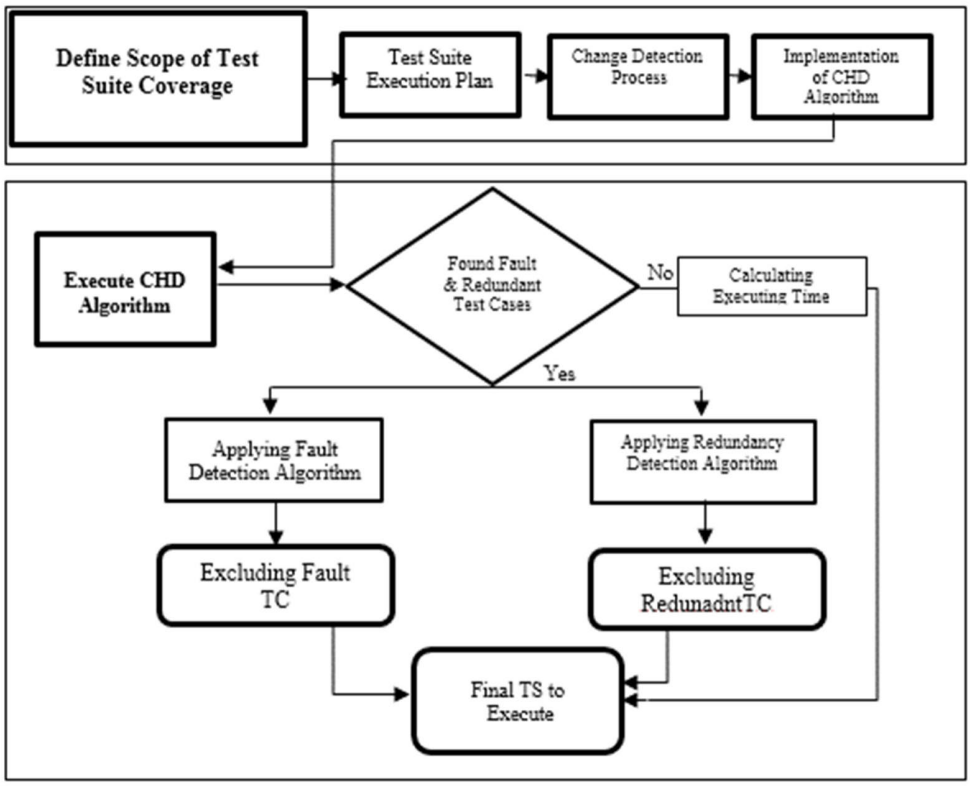


FIGURE 7. CHD algorithm workflow.

Figure 8 Shows the ARTeCS Algorithm for the Change Detection function using Variations steps consisting of selecting, detecting, and backtracking the algorithm between two (original and modified) test suites. It starts from the original test suite and extends until the modified version of the testing algorithm with time calculation. To conduct this phase, twelve test suite versions of test cases are selected randomly to handle permutations. The reduction process considers the coverage criteria defined as all-states, all-transition and transition-pair coverage by excluding fault test cases with modified change only. The proposed CHD algorithms' results in fault detection and time measurement. In the existing algorithm, there is also a lack of fault and redundancy detection in terms of time. It can cause the size of test suites to increase due to algorithms selecting similar test cases to be detected. A validation mechanism is also created to check the feasible path selection from implemented mapping to ensure that valid test cases are derived.

Figure 8 also explains the original and modified test case where the phone number and address have been updated for the HL7 file format; for every change, the tester needs to select.xml or HL7 files to ensure the mapping result is updated based on the modified changes in test cases. Fig 8 describes the difference between the two test case change modifications using the HL7 file format. Whereas, using the HL7 file format total of 100 test cases have been executed for 12 test suites, each test suite contains 100 test cases. ARTeCS approach shows the correct and unique test case selection time without any similarity. Here is the major contribution of TCS. It presents the main influence for file types HL7 and.xml.

Figure 8 also clarifies the difference between the two test case change modifications using the HL7 file format. Similarly, the original test case has been modified with gender and location information, so in this case, our enhanced algorithm (ARTeCS) detects these changes to select modified test cases only from the suite. Therefore, removing redundant and fault test cases to be selected in the modified test suite is useful. Figure 9 explains the change detection function using HL7 and.xml files where modified test cases based on these changes have been detected. Figure 9 also discusses the detection algorithm designed to optimize the selection and management of test cases in software testing by eliminating redundancy and faults. It introduces a technique based on four key selection scenarios: Time, coverage, redundancy, and fault detection ability. This technique allows testing teams to select modified test cases effectively. The aim is to enhance the efficiency and time of change detection processes. This is crucial for selecting the correct modified test cases for execution. Concludes by emphasizing the effectiveness of the enhanced CHD algorithm in improving the selection of modified test cases, providing better coverage and requiring less testing time than the original algorithm. However, it also notes that while the algorithm enhances certain aspects, it also

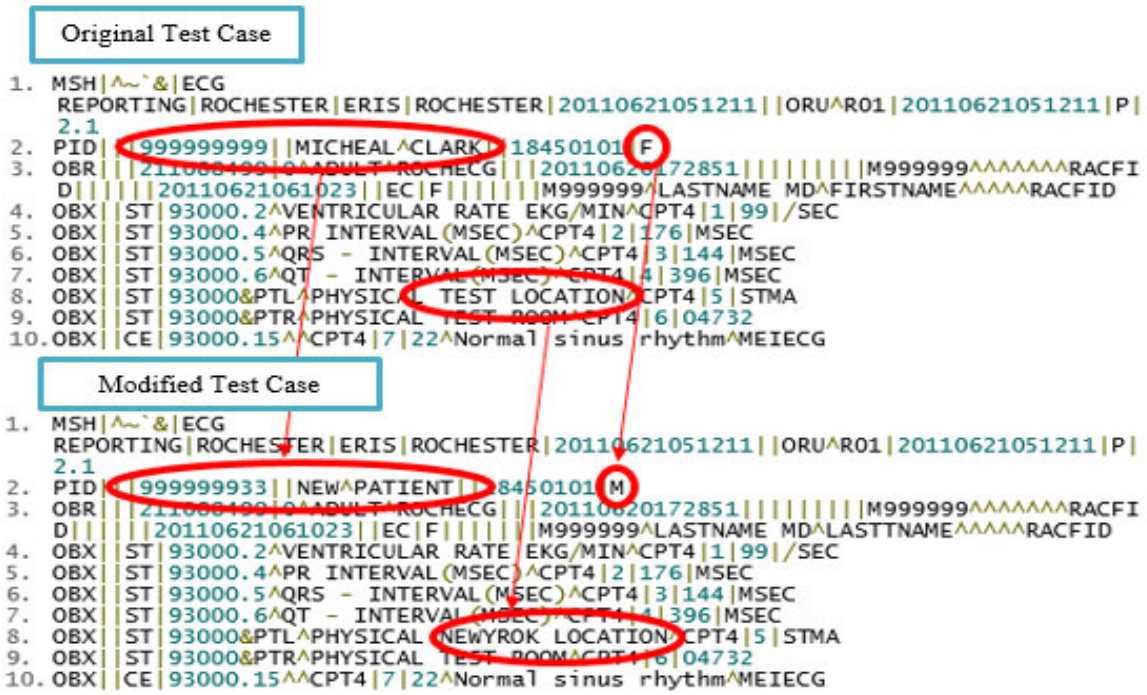


FIGURE 8. Change detection.

solves the optimization problem entirely. The implementation of optimization algorithms to improve the performance of the current technique, with the goal of achieving maximum coverage with optimal time and efficiency in the software testing industry.

V. RESULTS AND DISCUSSION

This section presents the comparative evaluation of proposed and existing approaches. This section is the further step of TCS in terms of change detection performance using the ARTeCS and CHD algorithms proposed based on three types of Search-Based Algorithms, Hybrid Whale Optimization (HWOA) and ACO Algorithm. A selection and detection mechanism, improvement detection of modified change with modified choice function, is used to guide the selection process to choose the best results based on three algorithms applied using JavaScript in ASP.NET.

A. COMPARATIVE ANALYSIS

The high coverage, redundancy, time performance and fault detection in the proposed algorithm compared with existing studies is due to the implementation of an efficient ARTeCS algorithm in the CHD. The ARTeCS concept will evaluate each iteration using the first iteration of the search and TCS algorithm to trace modified test cases from the proposed framework. The algorithm is then designed to detect the modified test cases process and select the overall and final modified version of a test suite to execute. Thus, this result encourages the implementation of the optimization algorithm to improve the currently proposed technique with the aim of improving the algorithm's performance measured by using time and efficiency measures with maximal coverage as a mapping result, as shown in Table 4, where Table 5 explains the total execution time for each test cases.

This table provides a comparison of different approaches to test case selection in regression testing, along with their evaluation metrics, Test Case Selection (TCS) ratio, and redundancy ratio. Three approaches are listed: ACO, HWOA, and ARTeCS. Each approach is evaluated based on several metrics. Average all-Suites coverage % indicates the percentage of coverage achieved by executing all test suites under the respective approach. It measures the comprehensiveness of test coverage across the software. Average all-Test cases % represents the overall percentage of coverage achieved by executing all individual test cases selected by the approach. It provides insight into the effectiveness of individual test cases in detecting faults. Average partial redundant test cases measure the percentage of redundant test cases identified by the approach. Partial redundancy refers to cases where some but not all aspects of a test case overlap with others. The average unique redundant Test cases metric quantifies the percentage of redundant test cases that are entirely redundant and do not offer unique coverage compared to others. The average time of execution (seconds) metric indicates the average time taken to execute the selected test cases under each approach. It assesses the efficiency of the approach in terms of execution time. Additionally, the table includes two ratios. TCS ratio compares the coverage achieved by executing all test cases (including redundant ones) to the coverage achieved by executing only selected test suites. A higher TCS ratio

TABLE 4. Mapping results to detect redundancy.

Approaches	Evaluation metrics	TCS ratio	Redundancy ratio
ACO	Average all-Suites coverage %	28.08%	13.08%
	Average all-Test cases (%)	32.02%	19.08%
	Average partial redundant test cases%	20.10%	11.00%
	Average unique redundant test cases%	11.90%	08.00%
	Average time of execution (seconds)	0.050	01.50
HWOA	Average all-Suites coverage %	33.00%	21.08%
	Average all-Test cases %	35.00%	29.08%
	Average partial redundant test cases%	26.80%	21.00%
	Average unique redundant test cases%	8.20%	08.00%
	Average time of execution (seconds)	0.075	02.50
Artemis	Average all-Suites coverage %	4.80%	44.02%
	Average all-Test cases %	46.00%	29.08%
	Average partial redundant test cases%	26.00%	15.02%
	Average unique redundant test cases%	20.00%	14.06%
	Average time of execution (seconds)	0.040	01.50

TABLE 5. Total execution time.

Test suites	*TC1*	*TC2*	*TC3*	*TC4*	*TC5*	*TC 6*	*TC7*	*TC8*	*TC9*	*TC10*	*TC11*	*TC12*	*TC13*	*TC14*	*TC15*	Redundant TC	Execution Time (in minutes/sec)
S1	√	√	×	√	√	√	√	√	√	√	×	×	√	√	√	3	1.00
S2	×	√	√	√	√	√	√	√	√	√	√	×	√	√	√	2	1.50
S3	×	√	×	×	√	√	√	√	√	√	√	√	√	√	×	4	2.00
S4	√	×	×	√	√	√	√	√	√	√	√	√	√	√	√	2	1.50
S5	√	√	√	√	√	×	×	√	√	√	√	√	√	√	√	2	1.50
S6	×	√	√	√	√	√	√	√	√	√	√	√	×	√	√	2	1.50
S7	√	√	√	√	√	√	√	×	√	√	√	√	√	√	√	1	1.00
S8	×	√	√	√	√	√	√	√	√	×	√	√	√	√	√	2	1.50
S9	√	√	√	×	×	√	√	√	√	√	√	√	√	√	√	2	1.50
S10	√	×	√	√	√	√	×	√	√	√	√	√	√	√	√	1	1.00
S11	√	√	√	×	√	√	√	√	√	×	×	√	√	√	√	3	1.50
S12	√	√	√	×	√	√	√	√	√	√	×	×	√	√	×	4	2.00

indicates a higher proportion of test cases contributing to the overall coverage. The redundancy ratio compares the percentage of redundant test cases to the total percentage of test cases. It quantifies the extent of redundancy within the selected test cases. Overall, the table comprehensively compares different regression testing approaches based on their coverage, redundancy, and execution efficiency metrics, offering insights into their strengths and weaknesses in ensuring effective test coverage while minimizing redundancy and execution time.

Table 5 presents a comprehensive overview of various test suites along with their respective test cases, indicating whether each test case is included (marked with a checkmark"√") or excluded (marked with an "×"). Additionally, it includes information on the number of redundant test cases within each test suite and the execution time required for running the test suite. Test Suites: The table lists several test suites (S1 to S12), each designed to test different aspects of the software. Test Cases: This allows for a clear understanding of which scenarios are covered by each suite.

This table also indicates the number of redundant test cases within each test suite. Redundant test cases are those that overlap in functionality with other test cases, potentially providing duplicated coverage. Execution Time: The last column displays the execution time required for running each test suite. This metric is crucial for assessing the efficiency of test execution, as it helps in understanding the time investment needed for testing each aspect of the software. Overall, this table serves as a valuable resource for test planning and execution, providing insights into the coverage offered by each test suite, identifying potential redundancies, and

TABLE 6. Change and fault coverage ration.

Approaches	Evaluation metrics	Change detection ratio	Fault detection ratio
ACO	Average all-Suites coverage (%)	33.00%	21.08%
	Average all-Test cases coverage (%)	35.00%	29.08%
	Average Size of the Test suite	26.80%	21.00%
	Average time of execution (seconds)	0.075	02.50
HWOA	Average all-Suites coverage (%)	23.00%	11.08%
	Average all-Test cases coverage (%)	25.00%	19.08%
	Average Size of the Test suite	16.80%	11.00%
	Average time of execution (seconds)	0.075	02.10
ARTeCS	Average all-Suites coverage (%)	43.00%	31.08%
	Average all-Test cases coverage (%)	35.00%	39.08%
	Average Size of the Test suite	26.80%	31.00%
	Average time of execution (seconds)	0.075	01.50

understanding the time investment required for running the test cases.

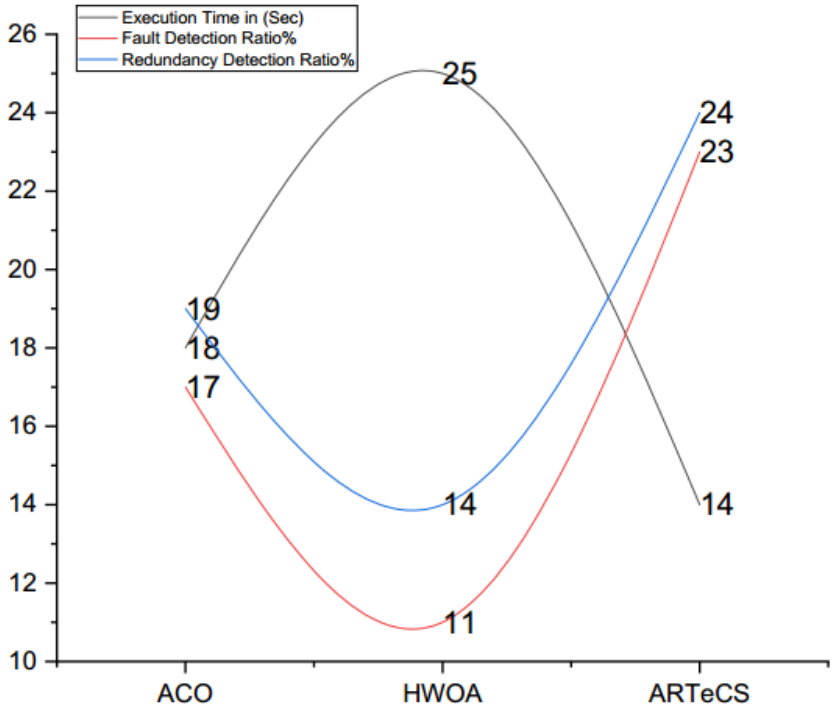
Table 6 compares three different approaches, ACO, HWOA, and ARTeCS used in some form of testing, likely related to change detection and fault detection. Each approach is evaluated across various metrics. The ACO approach achieves a relatively medium average all-suites coverage and all-test cases coverage, indicating comprehensive testing. However, the test suites' size is larger than other approaches, which may lead to longer execution times. HWOA achieves lower coverage percentages compared to ACO and ARTeCS. The size of the test suites is smaller, indicating a more focused selection of test cases. Despite similar execution times to ACO, the coverage achieved by HWOA is comparatively lower. ARTeCS demonstrates the highest average all-Suites coverage and all-Test cases coverage among the three. The size of the test suites is larger, suggesting a comprehensive selection of test cases. Despite this, ARTeCS achieves efficient execution times, indicating a balance between coverage and efficiency.

In summary, while ACO and ARTeCS achieve high coverage percentages, they may suffer from larger test suite sizes and potentially longer execution times. HWOA, on the other hand, maintains smaller test suite sizes but sacrifices coverage to some extent. ARTeCS emerges as a promising approach, striking a balance between comprehensive coverage, efficient execution, and manageable test suite sizes. Results show how the proposed ARTeCS approach is more efficient than ACO and HWOA. Overall comparison (Average ratio of TCS for fault selection, redundancy, Coverage and Time for TCS when test cases = (20, 40, 60, 80, 100) have been executed for each iteration where it shows ARTeCS approach comprehensively improved the execution time, redundancy, fault detection with maximum coverage of modified test cases in terms of test case selection (TCS).

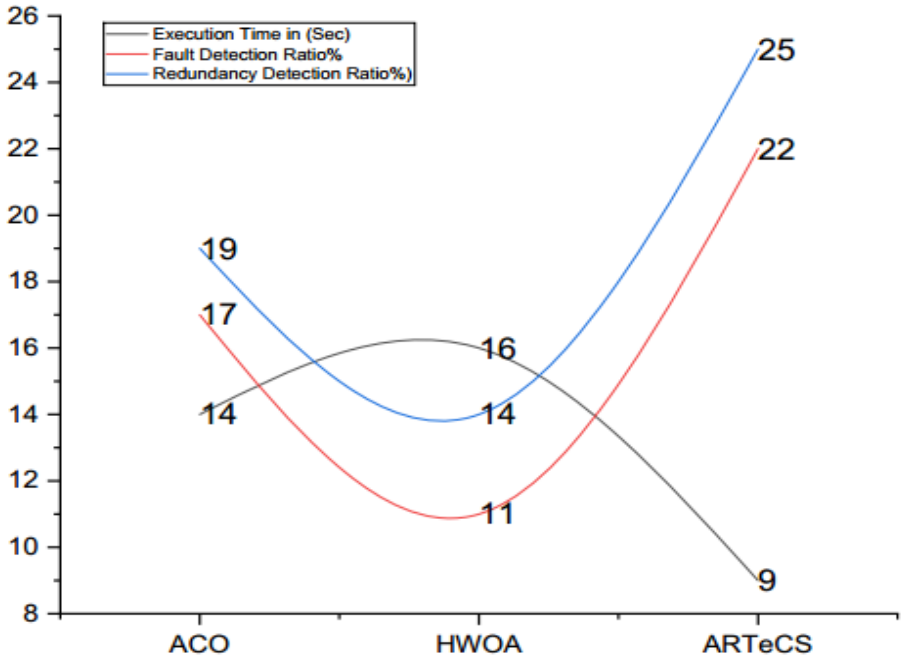
Figure 9 compares three different algorithms (ACO, HWOA, and ARTCS) across varying numbers of test cases (20, 40, 60, 80, and 100). Each graph has three lines representing Execution Time in Seconds (blue), Fault Detection Rate (red), and Test Execution Time (black). The y-axes of the graphs are labelled with numerical values representing performance metrics, while the x-axes are labelled with the three algorithms. In the graphs, the trends and performance of each algorithm can be observed: For 20 test cases, the HWOA algorithm shows the lowest values for all three metrics, while ACO and ARTCS have higher and varying values. In the 40 test cases graph, HWOA still shows lower execution times, but ACO has higher fault detection rates, and ARTCS presents a middle ground.

With 60 test cases, ACO again leads in fault detection, while ARTCS shows the highest test execution times, and HWOA maintains a low execution time. The graph for 80 test cases shows a significant rise in fault detection for ARTCS, with ACO and HWOA show lower fault detection but higher execution times. Finally, for 100 test cases, ARTCS has the highest fault detection rate, while ACO and HWOA have higher execution times and varying performance. The graphs collectively depict how each algorithm scales with an increasing number of test cases, highlighting their strengths and weaknesses in execution time and fault detection rate.

Figure 10 illustrates the overall comparison (Average ratio of for fault detection, redundancy, coverage and time when test cases = (20, 40, 60, 80, 100) have been executed for each iteration, where it shows the ARTeCS approach comprehensively improved the execution time, redundancy, fault detection with maximum coverage of modified test cases in terms of test case selection (CHD).



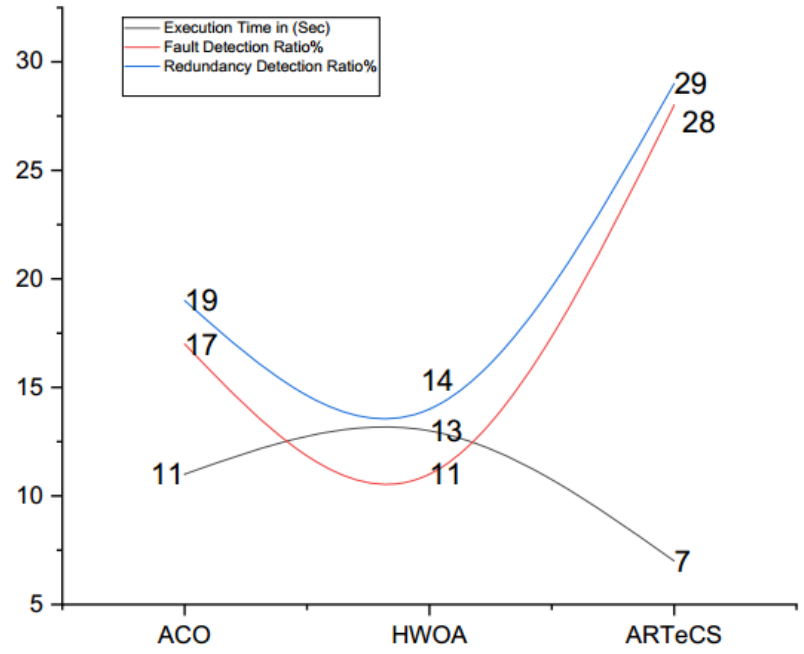
When Test Cases = 20



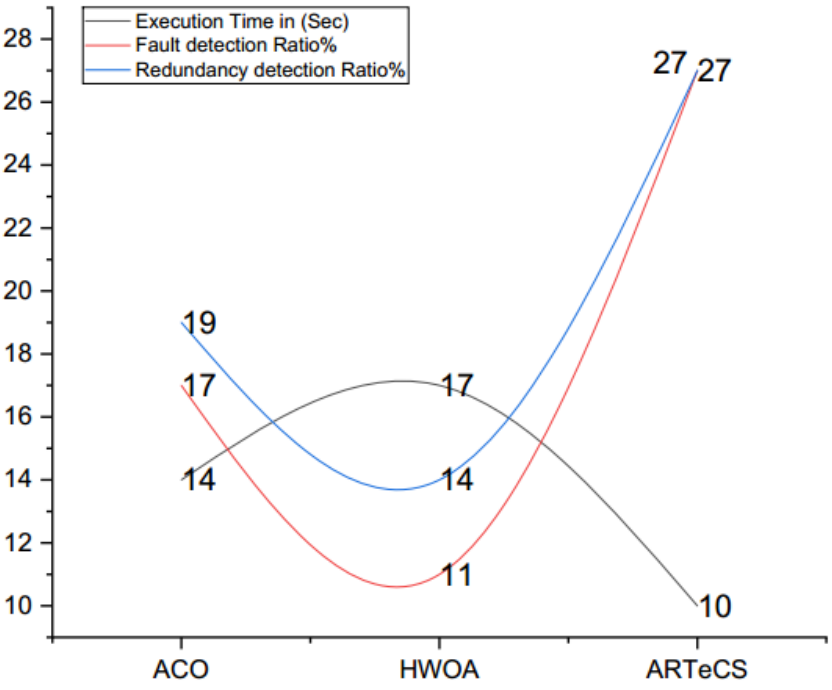
When Test Cases = 40

FIGURE 9. Overall comparison (Average ratio of TCS for fault detection, redundancy, coverage and Time for TCS when test cases = (20, 40, 60, 80, 100).

The image contains five line graphs illustrating the performance of three algorithms (ACO, HWOA, and ARTCS) over varying numbers of test cases: 20, 40, 50, 60, and 100. The performance metrics evaluated are Execution Time in Seconds (blue line), Fault Detection Rate (red line), and Test Execution Time (black line). The y-axis of each graph



When Test Cases = 60



When Test Cases = 80

FIGURE 9. *(Continued.)* **Overall comparison (Average ratio of TCS for fault detection, redundancy, coverage and Time for TCS when test cases = (20, 40, 60, 80, 100).**

represents the numerical values of these metrics, while the x-axis lists the algorithms.

In the 20 test cases graph, ARTCS has the highest fault detection rate, while HWOA has the lowest across all metrics. ACO and ARTCS have higher execution and test execution times. The 40 test cases graph shows HWOA maintaining the lowest execution time, ARTCS leading in fault detection, and ACO with intermediate values.

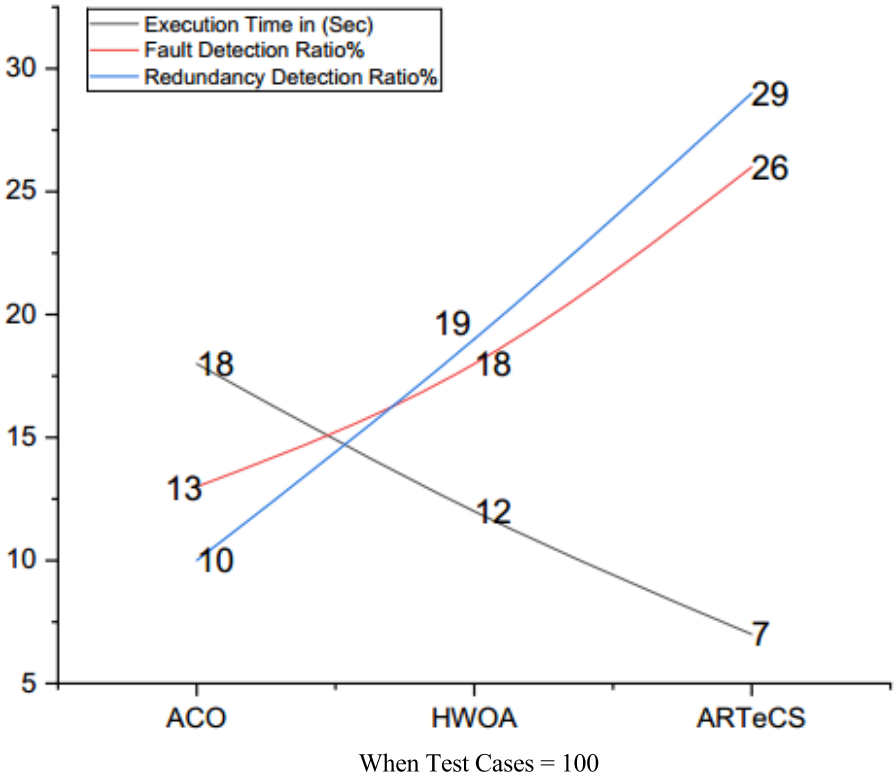


FIGURE 9. *(Continued.)* **Overall comparison (Average ratio of TCS for fault detection, redundancy, coverage and Time for TCS when test cases = (20, 40, 60, 80, 100).**

For 60 test cases, ARTCS again has the highest fault detection, with ACO and HWOA showing lower values. HWOA maintains a lower execution time compared to the others.

In the 80 test cases graph, ARTCS shows the highest fault detection and execution time, while ACO and HWOA have lower and more variable values.

The 100 test cases graph depicts ARTCS with the highest fault detection, ACO and HWOA with moderate fault detection rates, and HWOA maintaining the lowest execution time.

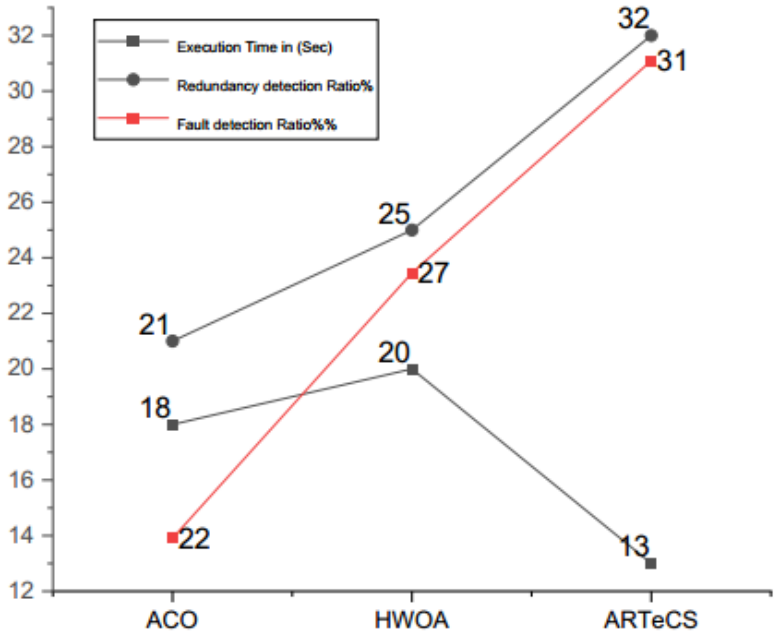
Overall, the graphs reveal that ARTCS tends to have the highest fault detection rates across different test cases and higher execution times. HWOA generally maintains lower execution times, while ACO exhibits varying performance. The trends highlight the trade-offs between fault detection capabilities and execution times for each algorithm as the number of test cases increases.

B. COMPARATIVE ANALYSIS OF THE FRAMEWORK

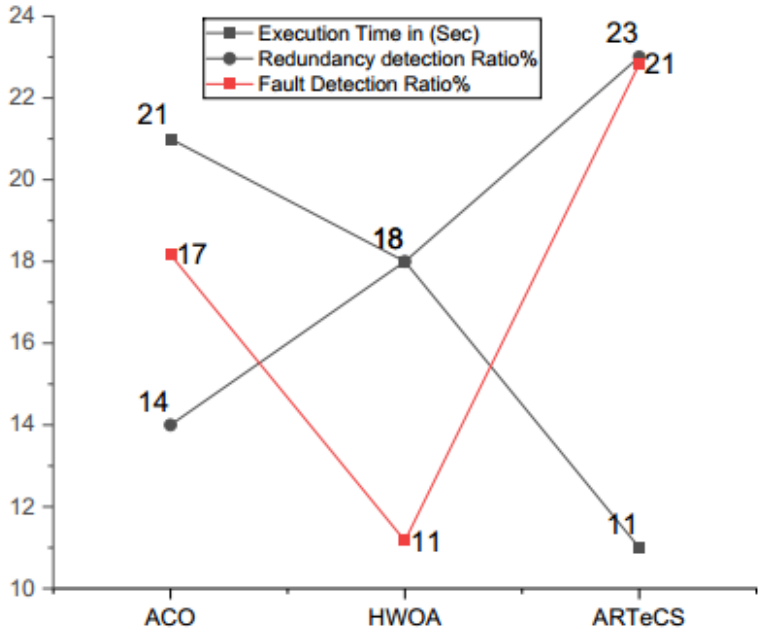
The framework enhancement process consists of scoping, planning, operation, analysis and evaluation of existing and enhanced algorithms for TCS and CHD. Each step has many sub-details to complete the process. The experiment scope set the goals for the experiment. The planning consists of the context of the experiment, hypothesis formulation, variable selection, subject programs or objects, experiment design type, instrumentation and validity evaluation for the experiment. Then, experiment operation and analysis are carried out as per the parameters chosen in the planning phase. The presentation of the experimental results is visualizations, graphs, descriptions, and discussion based on the data and the explanations. The scope of this experimental study is to analyses the test case selection criteria (Time, coverage, fault detection, redundancy), try to identify the relationship between time and efficiency measures (redundancy, coverage, fault detection ability) and propose a framework and test case selection technique based on these relative efficiency measures. The selection criterion is based on the accumulative effect of these efficiency measures on the selection procedure and tries to reduce the time and increase the overall efficiency of test case selection by using a change detection algorithm in TCS for regression testing. Whereas, enhanced Advanced Regression Efficient Test Case selection (ARTeCS) algorithm relies on the original and modified test suites to select the modified test cases to execute the System Under Test (SUT).

Very few studies focus on coverage and fault-based TCS techniques; however, redundancy in TCS techniques has been studied extensively on the software testing techniques within a given time frame with enough efficiency. The related research studies indicate the importance of TCS in past projects especially. This motivates the research community to further work on improving the TCS techniques for better testing the software testing effort by the ARTeCS framework.

Furthermore, it affects un-modified test cases to be selected. Thus, no valid selection of test cases is derived. Then, the tester needs to re-execute. Here, time, size, fault



When Test Cases = 20



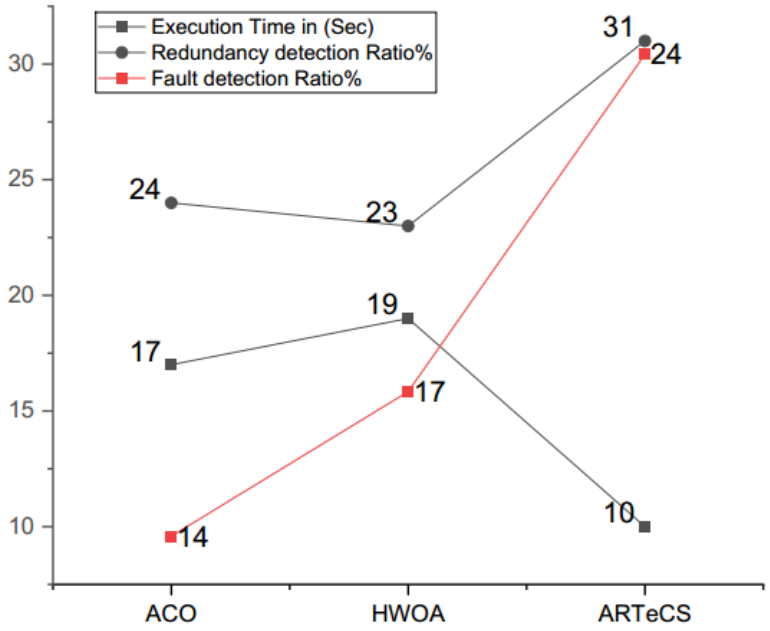
When Test Cases = 40

FIGURE 10. Overall comparison (Average ratio of CHD for fault detection, redundancy, coverage and Time for TCS when test cases = (20, 40, 60, 80, 100).

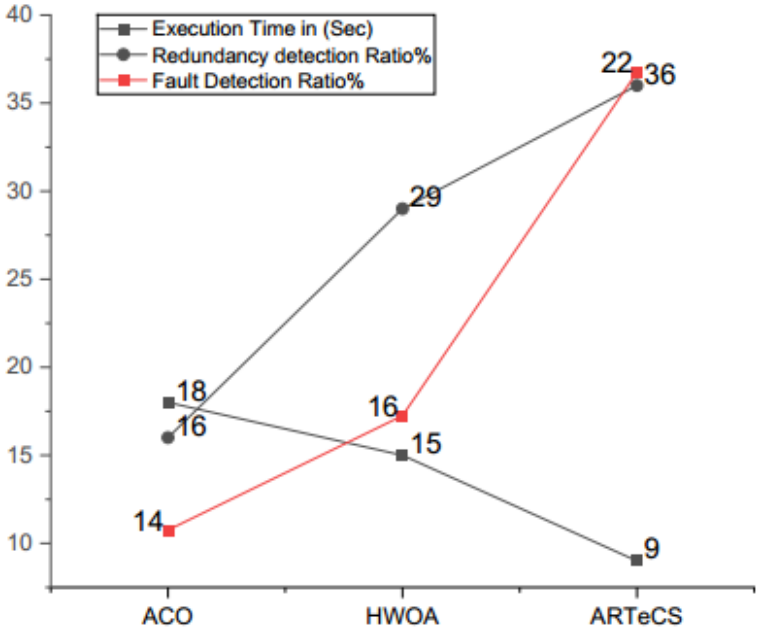
test case, and redundancy will still be increased even though the execution is faster. In terms of total execution time, the enhanced algorithm does not greatly improve the performance compared with the existing study. It supports effectiveness cannot represent good testing time if no good algorithm is enhanced efficiently. Thus, this caused the need to enhance the framework to ensure that the enhanced algorithm can significantly improve the efficiency of TCS. The enhanced technique uses a Change Detection Algorithm (CHD) to assess the modified change. The evaluation metrics (time, fault detection, redundancy, and coverage) are the same for Musa and ARTeCS framework and algorithm. The over results are mentioned in Figure 11 for existing and enhanced frameworks.

VI. ANALYSIS AND RESEARCH CONTRIBUTION

This study contributes to achieving the first objective of TCS techniques and enhanced efficient regression test case



When Test Cases = 60



When Test Cases = 80

FIGURE 10. *(Continued.)* **Overall comparison (Average ratio of CHD for fault detection, redundancy, coverage and Time for TCS when test cases = (20, 40, 60, 80, 100).**

selection algorithm to improve test suite efficiency in terms of fault detection ability, redundancy reduction and coverage by using the enhanced ARTeCS algorithm. This research advances the development of Test Case Selection (TCS) techniques by introducing an optimized regression test case selection algorithm that enhances the efficiency of test suites. The improved ARTeCS algorithm aims to elevate fault detection capabilities, diminish redundancy, and increase coverage. This study provides a framework that assists researchers in designing and conducting empirical studies on regression testing, utilizing enhanced rules and queries to refine the TCS process. The techniques implemented for TCS in regression testing (RT) are applicable to various types of coverage, fault metrics, and redundancy challenges in empirical research. The enhanced TCS algorithm promises to elevate both the time efficiency and overall effectiveness

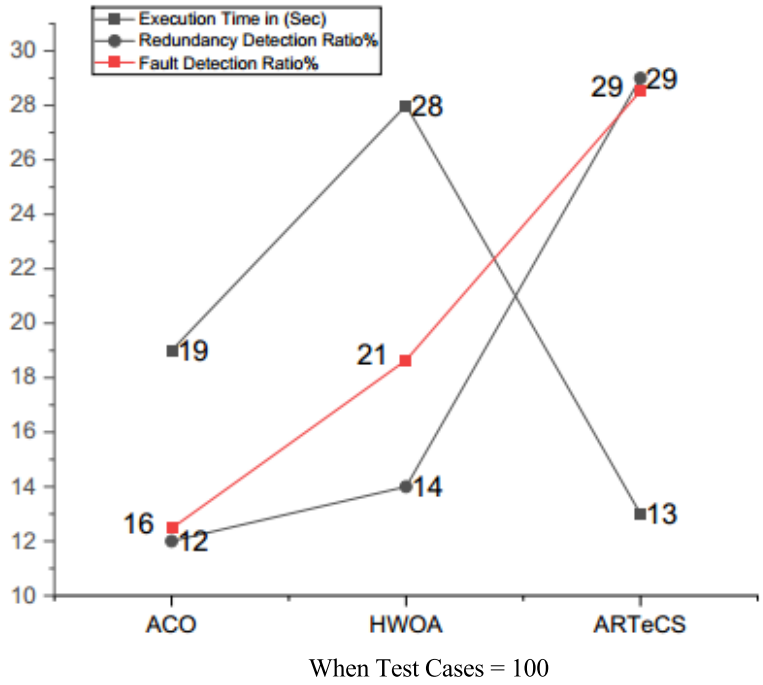


FIGURE 10. *(Continued.)* **Overall comparison (Average ratio of CHD for fault detection, redundancy, coverage and Time for TCS when test cases = (20, 40, 60, 80, 100).**

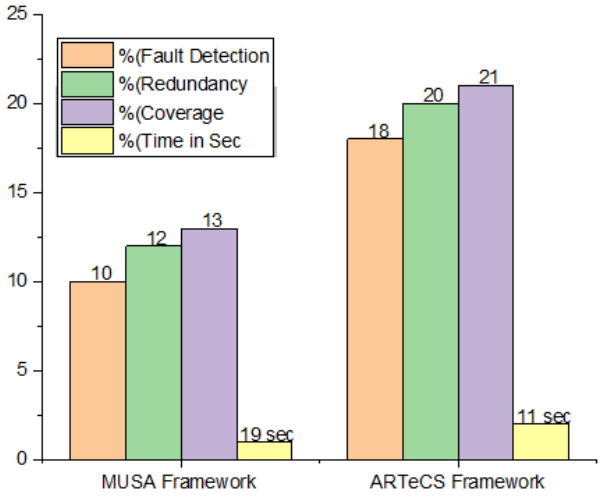


FIGURE 11. **Framework comparison results.**

of regression testing. This research is a valuable resource for test engineers, scholars, and industry professionals seeking to enhance change detection algorithms' efficiency, particularly in regression test case selection, by employing a multi-objective algorithm. Furthermore, the study introduces a sophisticated multi-objective change detection algorithm, contributing a nuanced understanding of the interplay between time, coverage, and fault detection in test case selection. Understanding a multi-objective change detection algorithm is vital in increasing the change detection process's efficiency.

The ARTeCS algorithm, which utilizes a detection-based, multi-objective approach for test case selection, has been refined in line with the research framework. It employs four key selection parameters—Time, coverage, fault detection, and redundancy integrating them into a weighted sum that aids in selecting appropriate test suites. This enables testing teams to apply a multi-criteria approach to evaluate the efficacy of multi-objective test case selection and change detection algorithms, particularly regarding time performance and test suite efficiency. The ARTeCS algorithm, bolstered by a multi-objective framework, has shown marked improvements in efficiency and time metrics over existing methods. It is particularly noteworthy when compared to existing HWOA and ACO algorithms, demonstrating statistically significant advancements in average selection time, coverage percentage, and test suite size. While some metrics showed less significant improvements, the overall findings indicate that the enhanced ARTeCS and multi-objective TCS approaches can reduce total execution time, thereby optimizing efficiency.

While comparing the existing ACO approach and the proposed multi-objective approach with the enhanced change detection, results show that average coverage test selection time(s) and average coverage reflect a significant improvement. Compared with the average selection time and size of test suites, results showed insignificantly improved results. Thus, it can be concluded that the enhanced ARTeCS and multi-objective TCS can improve efficiency to reduce the time (total execution time in seconds).

VII. THREATS TO VALIDITY

The results of this research are consistent; however, when the primary study is being selected, the analysis is being done, or the study is nearing its finish, there could be dangers to validity and bias in terms of different sizes of test suites and datasets. By eliminating bias, the primary study collecting step or exploration technique identifies the greatest number of studies that are available in the literature. Certain studies may be omitted due to murky research issues raised by the literature, such as the role played by solution strategies and the connections between primary research, categorization methods, and prediction levels. The study is connected to several software testing communities. The study is established on applying a method to promote the review for additional validation. The study's transparency will let the researchers make an informed assessment of the consistency of the results. As indicated in Figures 10 and 11, the results of this study reveal that validation of the study is necessary for all forms of support. Researchers are encouraged by the realistic case studies and expert reports that were needed for assessing TCS in execution time and overlooking critical efficiency measures such as coverage, fault detection, and redundancy. ARTeCS, however, is designed as a multi-objective technique to enhance time performance and efficiency relative to other TCS approaches. Through a comparison with existing methods, the study demonstrates significant improvements achieved by ARTeCS.

VIII. CONCLUSION

The study introduces a novel approach, the Advanced and Efficient Regression Test Case Selection (ARTeCS) framework, aimed at addressing the shortcomings of existing Test Case Selection (TCS) frameworks. The first objective, focused on designing and developing an algorithm that would address the issue regarding the importance of the test case selection objectives. The second objective, develop a CHD (change detection) algorithm that would use multiple objectives and combine it with the first object algorithm to improve the execution time in a test suite, which makes it more efficient as compared to existing algorithms. This technique was dubbed by adopting ACO and MOA algorithms, which is the acronym for Multi-Objective TCS Techniques. This objective was achieved by combining the knowledge achieved from the first objective, and detecting the modified change algorithm towards test case selection. This was done by developing a change detection algorithm and feeding it the appropriate data to become TCS through the change detection process. The third and final objective focused evaluation of the proposed framework through a comparative analysis of the said algorithms with another multi-objective framework. After rigorous TCS and CHD process, the results indicated that the proposed multi-objective technique is better in terms of fault detection redundancy, time and coverage. These two metrics were also compared in a time and coverage-cognizant manner to ensure the completeness of the results. However, the enhanced framework was able to overcome this particular challenge due to the fact that it is able to select and detect the TCS objectives based on modified test cases rather than the original test cases.

REFERENCES

[1] S. Romano, G. Scanniello, G. Antoniol, and A. Marchetto, "SPIRITuS: A simple information retrieval regression test selection approach," *Inf. Softw. Technol.*, vol. 99, pp. 62–80, Jul. 2018.

[2] F. Dobslaw, R. Wan, and Y. Hao, "Generic and industrial scale many-criteria regression test selection," *J. Syst. Softw.*, vol. 205, Nov. 2023, Art. no. 111802.

[3] A. P. Agrawal, A. Choudhary, and A. Kaur, "An effective regression test case selection using hybrid whale optimization algorithm," *Int. J. Distrib. Syst. Technol.*, vol. 11, no. 1, pp. 53–67, Jan. 2020.

[4] P. Jung, S. Kang, and J. Lee, "Automated code-based test selection for software product line regression testing," *J. Syst. Softw.*, vol. 158, Dec. 2019, Art. no. 110419.

[5] N. M. Minhas, K. Petersen, J. Börstler, and K. Wnuk, "Regression testing for large-scale embedded software development—Exploring the state of practice," *Inf. Softw. Technol.*, vol. 120, Apr. 2020, Art. no. 106254.

[6] M. H. Alkawaz, A. Silvarajoo, O. F. Mohammad, and L. Raya, "A system for optimizing software regression test cases using modified ant colony optimization algorithm," in *Proc. IEEE Symp. Ind. Electron. Appl. (ISIEA)*, Jul. 2022, pp. 1–5.

[7] M. Bharathi, "Hybrid particle swarm and ranked firefly metaheuristic optimization-based software test case minimization," *Int. J. Appl. Metaheuristic Comput.*, vol. 13, no. 1, pp. 1–20, Nov. 2021.

[8] H. Jia, C. Zhang, and S. Wu, "Multi-objective software test case selection based on density analysis," in *Proc. 5th Int. Conf. Comput. Sci. Softw. Eng.* Guilin, China: Association for Computing Machinery, 2022, pp. 140–147.

[9] S. Singhal, N. Jatana, A. F. Subahi, C. Gupta, O. I. Khalaf, and Y. Alotaibi, "Fault coverage-based test case prioritization and selection using African Buffalo Optimization," *Comput., Mater. Continua*, vol. 74, no. 3, pp. 6755–6774, 2023.

[10] S. M. Nagy, H. Amin, and N. Badr, "Enhanced regression testing execution process using test suite reduction techniques and parallel execution," *Int. J. Intell. Comput. Inf. Sci.*, vol. 23, no. 4, pp. 33–49, 2023.

[11] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "CodaMosa: Escaping coverage plateaus in test generation with pre-trained large language models," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, May 2023, pp. 919–931.

[12] S. Musa, A. B. M. Sultan, A. A. B. Abd-Ghani, and S. Baharom, "Regression test cases selection for object-oriented programs based on affected statements," *Int. J. Softw. Eng. Its Appl.*, vol. 9, no. 10, pp. 91–108, Oct. 2015.

[13] S. Sutar, R. Kumar, S. Pai, and B. R. Shwetha, "Regression test cases selection using natural language processing," in *Proc. Int. Conf. Intell. Eng. Manage. (ICIEM)*, Jun. 2020, pp. 301–305.

[14] W. Shi, M. Zhang, R. Zhang, S. Chen, and Z. Zhan, "Change detection based on artificial intelligence: State-of-the-art and challenges," *Remote Sens.*, vol. 12, no. 10, p. 1688, May 2020.

[15] Z. Mafi and S.-H. Mirian-Hosseinabadi, "Regression test selection in test-driven development," *Automated Softw. Eng.*, vol. 31, no. 1, p. 9, May 2024.

[16] S. B. Lity, "Model-based product-line regression testing of variants and versions of variants," Ph.D. dissertation, Carl-Friedrich-Gauss-Fac., Technische Universität Braunschweig, Braunschweig, Germany, 2020.

[17] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, and A. Panichella, "Single and multi-objective test cases prioritization for self-driving cars in virtual environments," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 2, pp. 1–30, Apr. 2023.

[18] T. A. Khan and Z. J. Akhtar, "Requirements business value and test case attributes based test case prioritization using deep reinforcement learning," *Res. Square*, pp. 1–12, Oct. 2023, doi: 10.21203/rs.3.rs-3497933/v1.

[19] G. Guizzo, J. Petke, F. Sarro, and M. Harman, "Enhancing genetic improvement of software with regression test selection," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 1323–1333.

[20] P. Palak and P. Gulia, "Hybrid swarm and GA based approach for software test case selection," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 9, no. 6, p. 4898, Dec. 2019.

[21] M. A. Siddique, W. M. N. Wan-Kadir, J. Ahmad, and N. Ibrahim, "Hybrid framework to exclude similar and faulty test cases in regression testing," *Baghdad Sci. J.*, vol. 21, no. 2(SI), p. 0802, Feb. 2024.
[22] K. Rana, H. Shah, and C. Kapadia, "Regression testing of service oriented software," *Int. J. Innov. Technol. Exploring Eng.*, vol. 8, no. 10, pp. 448–453, 2019.
[23] A. Arrieta, P. Valle, J. A. Agirre, and G. Sagardui, "Some seeds are strong: Seeding strategies for search-based test case selection," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 1, pp. 1–47, Jan. 2023.
[24] M. Samaila and Z. U. Armiya'u, "Program model-based regression tests selection approaches: A review," *Int. J. Sci. Global Sustainability*, vol. 7, no. 1, p. 6, 2021.
[25] M. Machalica, A. Samylkin, M. Porth, and S. Chandra, "Predictive test selection," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, May 2019, pp. 91–100.
[26] S. K. Harikarthik, V. Palanisamy, and P. Ramanathan, "Optimal test suite selection in regression testing with testcase prioritization using modified ann and whale optimization algorithm," *Cluster Comput.*, vol. 22, no. S5, pp. 11425–11434, Sep. 2019.
[27] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools," *Eng. Appl. Artif. Intell.*, vol. 111, May 2022, Art. no. 104773.
[28] I. Ghani, W. M. N. Wan-Kadir, A. Firdaus Arbain, and N. Ibrahim, "Improved test case selection algorithm to reduce time in regression testing," *Comput., Mater. Continua*, vol. 72, no. 1, pp. 635–650, 2022.
[29] P. Rodriguez-Mier, C. Pedrinaci, M. Lama, and M. Mucientes, "An integrated semantic web service discovery and composition framework," *IEEE Trans. Services Comput.*, vol. 9, no. 4, pp. 537–550, Jul. 2016.
[30] P. Jha, M. Sahu, and T. Isobe, "A UML activity flow graph-based regression testing approach," *Appl. Sci.*, vol. 13, no. 9, p. 5379, 2023, doi: 10.3390/app13095379.
[31] N. Iqbal and J. Sang, "Fuzzy logic testing approach for measuring software completeness," *Symmetry*, vol. 13, no. 4, p. 604, Apr. 2021, doi: 10.3390/sym13040604.
[32] A. Bajaj, A. Abraham, S. Ratnoo, and L. A. Gabralla, "Test case prioritization, selection, and reduction using improved quantum-behaved particle swarm optimization," *Sensors*, vol. 22, no. 12, p. 4374, Jun. 2022, doi: 10.3390/s22124374.
[33] D. Panwar, P. Tomar, and V. Singh, "Hybridization of cuckoo-ACO algorithm for test case prioritization," *J. Statist. Manage. Syst.*, vol. 21, no. 4, pp. 539–546, Jul. 2018.
[34] A. Panichella, F. M. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 122–158, Feb. 2018.
[35] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 609–620.
[36] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system," *Softw. Test., Verification Rel.*, vol. 25, no. 4, pp. 371–396, Jun. 2015.
[37] W. Wang, D. Li, W. Yang, Y. Cao, Z. Zhang, Y. Deng, and T. Xie, "An empirical study of Android test generation tools in industrial cases," in *Proc. 33rd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*. Montpellier, France: Association for Computing Machinery, Sep. 2018, pp. 738–748.
[38] H. Suryawanshi and P. R. Deshmukh, "Building a UI based tool for configuration management," in *Proc. Int. Conf. Ind. 4.0 Technol. (I4Tech)*, Sep. 2022, pp. 1–4.
[39] A. Arcuri, "RESTful API automated test case generation with EvoMaster," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 1, pp. 1–37, Jan. 2019.

ISRAR GHANI received the M.S. degree from the University of Arid Agriculture Rawalpindi, Pakistan, and the Ph.D. degree in software engineering from Universiti Teknologi Malaysia (UTM). He is currently a Visiting Researcher with UTM under Smart Digital Community Research Alliance (SDCRA).

WAN MOHD NASIR WAN KADIR (Member, IEEE) received the Ph.D. degree from The University of Manchester. He was the Head of the Software Engineering Department, from 2005 to 2009, and the Deputy Dean of the Faculty of Computing, from 2010 to 2015. Since 1997, he has been an Academic Staff with the Faculty of Computing, Universiti Teknologi Malaysia (UTM), where he is currently a Professor in software engineering and the Dean. His research interests include software engineering (SE), software testing, software quality assurance, and the application of AI/ML/DL in SE. He is the Chairperson of the ICT Dean Council and the Head of the ICT Cluster, National Council of Professors.

ADILA FIRDAUS ARBAIN received the Ph.D. degree in software engineering from Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia. She is currently a Lecturer and a Senior Lecturer with UTM. Her research interests include secured software development methods, software requirements analysis, and agile software development.

IMRAN GHANI received the master's degree in information technology from the University of Arid Agriculture Rawalpindi (UAAR), Pakistan, in 2002, the M.S. degree in computer science from Universiti Teknologi Malaysia (UTM), Malaysia, in 2007, and the Ph.D. degree from Kookmin University, South Korea, in 2010. He is currently an Associate Professor of software engineering with Virginia Military Institute (VMI). He has more than five years of industrial experience. He has a great deal of experience in Agile software development frameworks, models, and processes, including Scrum, Kanban, Lean, XP, and hybrid models and processes. He worked in several software houses, focusing on web development, databases, requirements engineering, and software testing. In 2019, he has published a textbook, *Introduction to PHP Web Services: PHP, Javascript, MySQL, SOAP, RESTful, JSON, XML, WSDL* which is currently available on Amazon.com. He is an active researcher with more than 120 publications in peer-reviewed quality journals and conference proceedings. His research interests include agile software development, secure software engineering, and semantic web.