Journal of Electronic Testing (2024) 40:347–370
https://doi.org/10.1007/s10836-024-06121-x

A Novel Framework For Optimal Test Case Generation and Prioritization Using Ent-LSOA And IMTRNN Techniques

A. Tamizharasi[1] · P. Ezhumalai[2]

Received: 30 December 2023 / Accepted: 21 May 2024 / Published online: 20 June 2024

Abstract
Test Case Generation (TCG) generates various types of tests, including functional tests, performance tests, security tests, and reliability tests to ensure software quality, while Test Case Prioritization (TCP) prioritizes the generated tests. However, the previous studies had challenges, including resource constraints, detecting crucial requirements, and automating the Test Case (TC) process efficiently. Additionally, the process is costlier and takes a maximum time duration that affects the effective performance. Therefore, an effective framework is proposed to overcome such issues by optimizing TCG and TCP processes effectively. The proposed work starts with the generation of a Unified Modeling Language (UML) diagram from historical project source code, which is then converted into a Comma-Separated Value (CSV) format. Then, the feature extraction is performed on this CSV file, followed by optimal TCG using the Entropy-based Locust Swarm Optimization Algorithm (Ent-LSOA). Additionally, factors are extracted and reduced from the historical project source code using Pearson Correlation Coefficient-Generalized Discriminant Analysis (PCC-GDA). Finally, the optimal TCs and selected factors are prioritized with the highest accuracy and recall of 96.89% and 96.92%, respectively using an Interpolated Multiple Time scale Recurrent Neural Network (IMTRNN). Thus, the proposed work outperformed the existing techniques by providing an efficient solution for TCG and TCP in software testing.

Keywords Entropy-based Locust Swarm Optimization Algorithm (Ent-LSOA) · Pearson Correlation Coefficient-Generalized Discriminant Analysis (PCC-GDA) · Test Case Generation (TCG) · Interpolated Multiple Time scale Recurrent Neural Network (IMTRNN) · Test Case Prioritization (TCP) · Software Testing · Unified Modeling Language (UML)

1 Introduction

Recently, in everyone's life, a major role has been played by software applications like banks, ATMs, online marketing et cetera. Hence, in the development of such software or system life cycle, software testing is considered to be a significant phase as it ensures the excellence as well as consistency of any software application [24]. The TC, which is a tool to proffer several test activities like requirement gathering, planning, test execution, build creation, test analysis, along with defect logging, is the core of software testing. Therefore, in software testing, the generation of effective test codes is highly significant [9]. Testing the entire behavior manually is impossible since the software is becoming extremely complicated. Thus, to handle time restrictions as well as complexity, software test automation could be utilized [19]. When a set of tests like regression testing, compatibility testing, et cetera are needed to be repeated, an automated generation of TCs is needed [23]. For an automated generation of TCs, numerous methodologies have been adopted in recent years.

(i) Randomized searching, (ii) Search-based testing [29], (iii) Concolic testing, and (iv) Model-Based Testing (MBT) are the methodologies utilized for an automated generation of TCs; among these, MBT becomes highly popular. In MBT, to create TCs, UML models are utilized generally [17]. The UML illustrates the object-oriented model;

Responsible Editor: B. Arasteh.

A. Tamizharasi
tamizh05384@gmail.com

1 Assistant Professor, Department of Computer Science and Engineering, R.M.D. Engineering College, Tamil Nadu, Kavarapettai, Thiruvallur, India

2 Professor and Head, Department of Computer Science and Engineering, R.M.D. Engineering College, Tamil Nadu, Kavarapettai, Thiruvallur, India

moreover, it runs via the entire software development cycle. UML is a popular modeling language; however, its diagrams could create inconsistencies owing to the lack of formal semantics. The formal models have precise semantics, which accurately illustrates a system's behavior and test generation methodologies; thus, they could mitigate such issues [20].

Nevertheless, the number of TCs could be huge regarding the size of the code base; in addition, their execution often requires numerous servers. Also, they could take hours or even days to complete [2]. Moreover, initially, to execute TCs, selecting a more valuable TC as of the TC library is essential in a resource-constrained environment [27]. Thus, selecting the optimal TCs utilizing enhanced metaheuristic algorithms is considered to be more effective. Nevertheless, to prioritize TCs, the TCP is utilized; hence, in software testing, TCP is deemed to be a vital process. To prioritize the TCs regarding ML, various recent models like Additional Greedy, Genetic Algorithm (GA), Greedy Algorithm, along with hyper-heuristic algorithm have been developed [11]. However, in the generation of optimal TCs, there are certain limitations. Similarly, the factors required for the TCP were not studied well, affecting the testing process's efficacy. A vital factor to be deemed for generating effective test data is the quality of testing specifications. The test automation process is facilitated by utilizing semi-formal specifications; however, it is still costly and less preferred. Additionally, for utilizing only limited test information, the state-of-the-art (SoA) TCP algorithms are optimized. They are not quick enough for executing in industrial continuous integration settings with a huge number of test executions. Therefore, this work proposed a novel IMTRNN-based TCP with the optimal TCs.

Still, there are certain limitations even after developing the TCG as well as TCP models utilizing ML algorithms. Some of the limitations found are stated as follows,

- The TCs being generated could not be mitigated by the conventional TCP algorithm, suffering from the issue of time complexity in running all generated TCs.
- The computation efficiency is poor in Risk-centricTCP utilizing a fuzzy expert system technology in the prevailing research.
- There are complications in handling larger input datasets in the prevailing Input-based adaptive randomized TCP; moreover, it did not provide cost-effective outcomes.
- Only a fixed strength, not multiple strengths, is considered by the traditional TCP whilst selecting every single TC.

To overcome the aforementioned issues, developing suitable work for the selection of optimum TCs and an efficient TCP method utilizing ML is the intention of the proposed model. The proposed work's main contribution is listed below;

- A PCC-DGA technique is utilized to reduce the factors of the source code and minimize the classifier's processing time.
- An Ent-LSOA is employed to reduce the features of the project source code and the complexity problem due to the higher dimensionality.
- An enhanced neural network, namely IMTRNN is developed for improving the proposed model's classification accuracy.

1.1 Motivations of the Proposed Work

The main motivation of the proposed work is to address the persistent challenges and limitations of TCG and TCP processes in software testing. These challenges include resource constraints, inefficient risk-centric TCP due to fuzzy expert systems, inefficiencies in automation, time-consuming operations, and limitations in existing models, such as poor computation efficiency and inability to handle larger datasets. The proposed framework aims to overcome these issues by introducing innovative methods, such as feature reduction with PCC-GDA, optimal TCG with Ent-LSOA, and prioritization using an enhanced neural network model named IMTRNN. Thus, the proposed work significantly improves the efficiency, effectiveness, and quality of TCG and TCP in software testing by ultimately advancing state-of-the-art techniques in software testing practices. The proposed work is structured as: Related works are analyzed in Sect. 2. The proposed TCG and TCP methodologies are explained in Sect. 3 and the outcomes of the proposed model are comparatively analyzed in Sect. 4. Finally, the paper is winded up in Sect. 5.

2 Related Works

[22] presented a TCG model for a significant path regarding an enhanced amalgamated fitness function. By utilizing the Adaptive Particle Swarm Optimization (APSO), which engendered the TCs automatically, an Improved Combined Fitness (ICF) function was developed by the presented model. Better outcomes were obtained by the presented ICF function, which was applied to the APSO on the number of path coverage. Nevertheless, as the iterations increased, an internal threat was produced to the model by the APSO algorithm's random nature; thus, the model obtained a lower convergence rate.

[3] developed a model on Modified Condition/ Decision Coverage (MC/DC) TCG as well as TCP methodologies. Initially, a greedy algorithm was utilized by the model to augment the TCs. Next, by considering the Consideration Index (CI) along with Fault Exposing Potential (FEP) values, TCs were prioritized. A reduction rate of 49% was attained by the presented model, minimizing the test suit size. Nevertheless,

the presented model would result in poor coverage if the input program occurred in non-deterministic behavior.

[25] introduced the model-centric TCP model regarding the Alternating Variable Method (AVM). Here, for mutation testing, a model-centric development approach had been utilized; similarly, for TCP, the AVM model had been employed. In all the tested scenarios, the presented model outperformed the other techniques. However, the factors affecting the TCP were not considered by the presented approach, affecting the prioritization outcomes.

[1] proffered a prioritized T-way Test Suite (TS) grounded on the Bi-objective Dragonfly Algorithm (BDA). TC weight and TC priority had equal importance whilst generating the test suit. TC priority and TC weight were objectives for the BDA. The tested TS size outcomes exhibited the BDA methodology's efficacy. However, the presented model could not perform the complex TC scenarios effectively.

[8] presented the TCs automatically regarding Differential Evolution with Node Branch Archive (NBAr-DE). A relationship between node branch direction and the value of TC variables was recorded by the branch archive technique, which also enclosed more paths by the driven search-centric algorithms. The experimental outcomes exhibited that the number of redundantTCs was mitigated by the NBAr-DE model. However, the presented model's computational time was increased by the random initialization and limited computation in the differential algorithm.

[10] suggested a TCP model grounded on Particle Swarm Optimization (PSO) to investigate software fault detection. By using the PSO algorithm, the necessary features were extracted from the dataset, and by utilizing the K-Means clustering, the obtained features were clustered. Better outcomes were provided by the presented model for priority ranking for clusters and fault detection rate. Although the K-Means clustering had many advantages, problems arose when the cluster of varied sizes was considered as the K-Means that only deemed the data in spherical shapes.

[18] presented a model for the TCP as of user requirements for web-centric software. For developing software applications, test scenarios were prioritized by the requirement-centric test scenario prioritization regarding the requirements gleaned from end users. In early test scenario prioritization and the detection of faults, the model performed very effectively. Nevertheless, the aimed requirement coverage was not obtained here.

[13] developed correlating system methodologies for the risk-centric TCP. The presented model utilized an automated procedure that extracted risk factors like complexity, requirements modification information, and the models' size. Subsequently, the system models' risk values were computed. As per the experimental outcomes, in the presented model, the test efficiency was enhanced by spotting the defects earlier. The product-associated risk factors were considered by the presented model even though the risk process and project risk factors were neglected. Therefore, certain crucial TCs might have been prioritized incorrectly during the TCP process.

[4] developed discrete and combinatorial gravitational search algorithms aimed at TCP as well as minimization. In addition to this, the chaotic map developed an enhanced version that updated the gravitational constant. The simulation outcomes demonstrated that when analogized with the GA, the presented models were more effective for TCP as well as minimization. Nevertheless, at the terminal iterations, the TCP process was delayed by possessing slower convergence.

[5] presented a TCP model regarding a discrete Cuckoo Search Algorithm (CSA). Here, concerning asexual genetic reproduction, a new adaption strategy was presented for the transmutation of real numbers. The statistical examination proved that hybrid CSA outperformed the GA and PSO by 4.29% and 5.52%, respectively. However, the hybrid CSA's computational expense was higher than the other prevailing methodologies.

[21] illustrated a model for the Test Suite Generation (TSG) with the archive-centric multi-criteria Artificial Bee Colony (ABC) algorithm. Initially, for utilizing the available resources effectively, the presented model kept the covered targets in the archive. Additionally, the model was enhanced to provide more diversity to the population. Experimental outcomes demonstrated that in contrast to the basic ABC algorithm, a presented model provided fast convergence. However, in the exploitation stage, the complex problems were not processed by the model.

[12] recommended the utilization of an activity diagram along with a search-based technique for Automatic Test Data Generation (AutoTDGen). Based on an activity diagram, a model-centered approach had been developed; additionally, GS was utilized as the TCG technique. The experimental outcomes demonstrated the model's enhanced fault-detection performance by obtaining 11.1% better than the Data Flow Annotated Activity Diagram (DFAAD). However, the problem that arises during the process could make it difficult to correct the detected faults.

[16] demonstrated ontology-centered test generation for autonomous as well as automated driving functions. The presented model utilized ontologies that illustrated the autonomous vehicles' environment and transmuted them into systems for combinatorial testing. The experiential outcomes, which were reliant on the instance from the automotive industry, indicated that the model could be utilized in reality. Nevertheless, here, the modified domain was considered the least, affecting the TSGs' performance.

[6] introduced a model regarding nature-inspired algorithms for testing the regression. TCP, selection, and minimization that mitigated costs as well as efforts were the 3 techniques that tested the regression. The TCs were

prioritized by the nature-inspired algorithm on the code coverage conditions. Finally, the redundant TCs were taken away. Here, the TS was minimized by the model with full statement coverage; in addition, it also mitigated the negligible fault coverage loss. Nevertheless, there occurred time as well as computational complexity owing to the usage of multiple nature-inspired algorithms.

[30] presented an extremely simple as well as effective way for prioritizing the TCs via the introduction of the dispersity metric. Here, for the TCP, the presented model utilized GA as well as PSO hybridization. The experimental outcomes illustrated that the model was more effective than the Random Prioritization (RP). However, during the prioritization process, the introduction of the dispersity metric could produce misleading outcomes.

[26] introduced TCP on sample data sets with multiple iterations. African Buffalo Optimization (ABO) was applied for prioritizing the TCs. The comparative analysis was performed with the Ant Colony Optimization (ACO) and showed that ABO was highly efficient with lower execution time. However, this approach was not well suited for processing a large set of historical test execution data.

[7] exhibited Single and Multi-objective TCP for Self-driving Cars in a Virtual Environment. 2 black-box TCP approaches were employed for enhancing the regression testing's cost-effectiveness. Thus, the developed model significantly improved the capability to detect safety-critical failure in a very short time. However, the accuracy of the developed model was low due to training the model with a limited number of features.

[28] offered Scalable and Accurate TCP in Continuous Integration Contexts. A data model, which captured data sources and their relations, was defined. Secondly, a set of features was defined centered on this data model. The gathered features were inputted into the Machine Learning model. The experimental results demonstrated the effectiveness of ML-based TCP techniques. Nevertheless, the influence of removing FF TCs on the average number of TCs per build and the average regression testing time per build was negligible.

[15] introduced a framework to optimize TSs by focusing on path coverage-based optimizations in software testing. The working principle of this method involved iteratively improving test data using six meta-heuristic algorithms, namely the Hill-Climbing Algorithm, PSO, Firefly Algorithm, CSA, Bat Algorithm, and Artificial Bee Colony Algorithm to generate optimal TSs. Thus, the usage of these algorithms explored and refined the search space to achieve the desired path coverage and branch coverage in software testing. However, the algorithms used in this work took maximum duration due to extensive search exploration in path converge optimization that hindered the overall performance.

[14] accomplished a statistical model to assess the influence of various factors on testing cost, quality, and effort in software testing. This work studied the behavioral patterns of testing professionals and analyzed the relationship between various factors, such as project duration, Software Complexity, and testing tools. Regression analysis was employed to quantify the impact of automation on testing cost, quality, and effort while controlling for other variables. Nevertheless, the inadequate software test automation impacted the cost, quality, and effort of the entire work, which degraded the entire work.

The summary of related works is provided in Table 1.

2.1 Summary of the Related Works

Numerous studies were conducted to enhance various aspects of software testing in TCG and TCP. These works employ diverse optimization techniques, such as PSO, Genetic Algorithms, and Nature-inspired Algorithms to improve testing efficiency and effectiveness. However, many existing works suffer from drawbacks, such as computational inefficiency, scalability issues, non-detection of crucial requirements, and limited consideration of real-world testing scenarios. These drawbacks hinder the overall effectiveness of software testing optimization techniques, leading to suboptimal outcomes. Therefore, to overcome such issues, the proposed work is utilized, which reduces the features using PCC-GDA, generates the optimal TC using Ent-LSOA, and enhances the prioritization using IMTRNN. Thus, it improves the efficiency, effectiveness, and quality of TCG and TCP in software testing.

3 Proposed Test Case Generation And Prioritization Methodologies

To estimate whether varied features within the system are performing as expected, the TCs are produced. Nevertheless, these TCs require effective tools for the process; thus, the automated generation as well as prioritization of such TCs are complicated. Hence, in the proposed framework, a novel Ent-LSO-based optimal TCG and IMTRNN-based TCP is introduced. The proposed model consists of two phases, namely training and testing. During the training phase, the UML diagram is generated from the project source code, and then, the features are extracted and selected. On the other hand, the factors of source code are extracted. From that, optimal factors are chosen. Finally, the chosen factor, historical TCs, and optimal TCs are inputted into the trained model to prioritize the TCs for removing bugs from the software. At the testing phase, the real-time source code's optimal features are given as input into the trained optimal

Table 1 Summary of related works

Author	Objective	Method	Advantage	Disadvantage
[22]	To enhance TCG	APSO to automatically generate TCs. The model incorporated ICF to enhance path coverage	Better outcomes for path coverage when applied to APSO	Lower convergence rate
[3]	To develop TCG and TCP model	Greedy algorithm to augment TCs and prioritize them based on CI and FEP values	Reduced the size of the TS while maintaining coverage	Poor coverage in cases of non-deterministic program behavior
[25]	Model-centric TCP method for enhanced prioritization outcomes	A model-centric development method for mutation testing and the AVM model was employed for TCP enhancement	Prioritized TCs effectively using the AVM model	Did not consider factors that affect TCP, leading to suboptimal prioritization outcomes
[1]	To develop a prioritized T-way TS	Implemented BDA to assign equal importance to TC weight and priority during TSG	Efficiency in determining TS size outcomes	Struggled to handle complex TC scenarios
[8]	To automatically generate the TCs	Branch archive technique to record the relationship between node branch direction and TC variables. Driven search-centric algorithms to explore more paths	Reduced the number of redundant TCs with NBAr-DE	Increased computational time due to random initialization
[10]	Development of TCP model	Utilized PSO algorithm to extract necessary features from the dataset. Employed K-Means clustering to cluster the obtained features	Better outcomes in priority ranking for clusters and fault detection rate	Reduction in accuracy occurs when considering clusters of varied sizes
[18]	Development of TCP model based on user requirements for web-centric software	Prioritize test scenarios based on requirements gathered from end users	Effective performance in early test scenario prioritization and fault detection	Failed to achieve the aimed requirement coverage
[13]	Develop system methodologies for risk-centric TCP	Extract risk factors, such as complexity, requirements, modification information, and model size	Enhanced test efficiency by early defect detection	Incorrectly prioritizing crucial TCs due to the negation of certain risk factors
[4]	Develop discrete and combinatorial gravitational search algorithms for TCP and minimization	Enhance the gravitational search algorithm with a chaotic map and update the gravitational constant	Demonstrated superior effectiveness of the presented models for TCP and minimization compared to GA	Experienced delays in the TCP process during terminal iterations due to slower convergence
[5]	To develop a TCP model	Discrete CSA for optimal prioritization	Effectively outperformed in TCP	The computational expense of the hybrid CSA was higher
[21]	To develop a model for TS generation	Utilize an ABC algorithm that maintains covered targets in an archive for efficient resource utilization	Achieved faster convergence	The model struggled to process complex problems during the exploitation stage
[12]	Recommend the use of an activity diagram combined with a search-based technique for AutoTDGen	Utilize GS as the TCG technique	Improved fault detection performance	Difficulty in the correction of detected faults
[16]	Ontology-generated test generation for autonomous and automated driving functions	Utilize ontologies to represent the environment of autonomous vehicles and transform them into systems for combinatorial testing	Experimental outcomes suggested practical applications	Negation of modified domain adversely affected TSG performance
[6]	Focused on TCP, selection, and minimization techniques	Prioritize TCs based on code coverage conditions using nature-inspired algorithms	Successfully minimized the TS and mitigated negligible fault coverage loss	The usage of multiple nature-inspired algorithms led to time and computational complexity issues

Table 1 (continued)

Author	Objective	Method	Advantage	Disadvantage
[30]	A simple and effective method for prioritizing TCs	Utilize GA and PSO hybridization along with the dispersity metric for prioritizing TCs in TCP	Outperformed RP and demonstrated high effectiveness	Dispersity metric introduction during prioritization could lead to misleading outcomes
[26]	Developed a TCP model	ABO for prioritizing TCs and compared with ACO for efficiency	ABO showed higher efficiency and lower execution time compared to ACO	Not suitable for processing large sets of historical test execution data
[7]	Single and Multi-objective TCP for Self-driving Cars in a Virtual Environment	Employ two black-box TCPs to enhance regression testing's cost-effectiveness and improve safety-critical failure detection	Improved the capability to detect safety-critical failures in a short time	Accuracy was low due to training with a limited number of features
[28]	Provide Scalable and Accurate TCP in Continuous Integration Contexts	A data model, which captures data sources and relations	Demonstrated the effectiveness of ML-based TCP techniques	Removing FF-TCs had a negligible impact on the average number of TCs per build and regression testing time per build
[15]	To optimize TSs focusing on path coverage-based optimizations in software testing	Six-metaheuristic algorithms to improve test data	Algorithms explored and refined the search space to achieve the desired path coverage and branch coverage	Extensive search exploration in path convergence optimization led to maximum duration
[14]	A statistical model to assess the influence of various factors on TCs	Employ regression analysis to quantify automation's impact on testing cost, quality, and effort	Analyze relationships between factors for effective performance	Inadequate software test automation impacted cost, quality, and effort

prioritization model. The structural diagram of the proposed work is shown in Fig. 1.

3.1 Training Phase

To train as well as test the model, the historical projects' source codes are taken as input. Here, the historical source code utilized is generated with the historical project requirements. The source code of project requirements is normally understood as the mean project programming statements that are generated by a programmer with a visual programming tool; later, they are saved in a file. In addition, it was tested and verified with the generated TCs. The new project requirement is taken as input for testing. Then, to easily understand the concept of the code, the UML diagram is created. Then, it is exported into the CSV file, which has the compressing capability. Therefore, in the proposed system, the storage problem is avoided. After that, the features are extracted, and later, the optimal TC is chosen. For prioritizing the TCs, they are inputted into the trained model. Determining TC order that maximizes the probability for the prior discovery of faults in source code is the TCP's goal. In the software's prior working versions, it is verified that the bugs are fixed as well as the newly added features haven't created any issues. The flowchart of the proposed work is shown in Fig. 2.

Figure 2 explains the proposed methodology. The proposed work starts by collecting the historical project requirements. From the collected historical project requirements, the historical codes are generated. If the source code is available, then UML diagrams are generated for the source code and are then exported as a CSV file. After that, the features are extracted. Then, the optimal features are chosen along with the generation of test cases. Finally, the test cases are prioritized using IMTRNN. Then, the prioritized test cases are checked based on whether the bugs are fixed and the new features don't have any issues. Here, if the bugs are fixed and no issues appear in features, then end the process; otherwise, fix the bugs and issues in new features and then end the process.

Here, the TCs that are utilized to validate the historical project's source code are expressed in Eq. 1,

$$h_{test} = \{h_1, h_2, \ldots ..., h_n\} where\ test = 1,2, \ldots, n \tag{1}$$

where, (n) represents the total number of utilized historical project TCs (h_{test}) that train the proposed TCP classifier model.

Additionally, the historical source code data that are generated from historical project requirements are mathematically represented in Eq. 2,

$$S_a = [S_1, S_2, \ldots, S_a] where\ a = 1,2, \ldots ., a \tag{2}$$

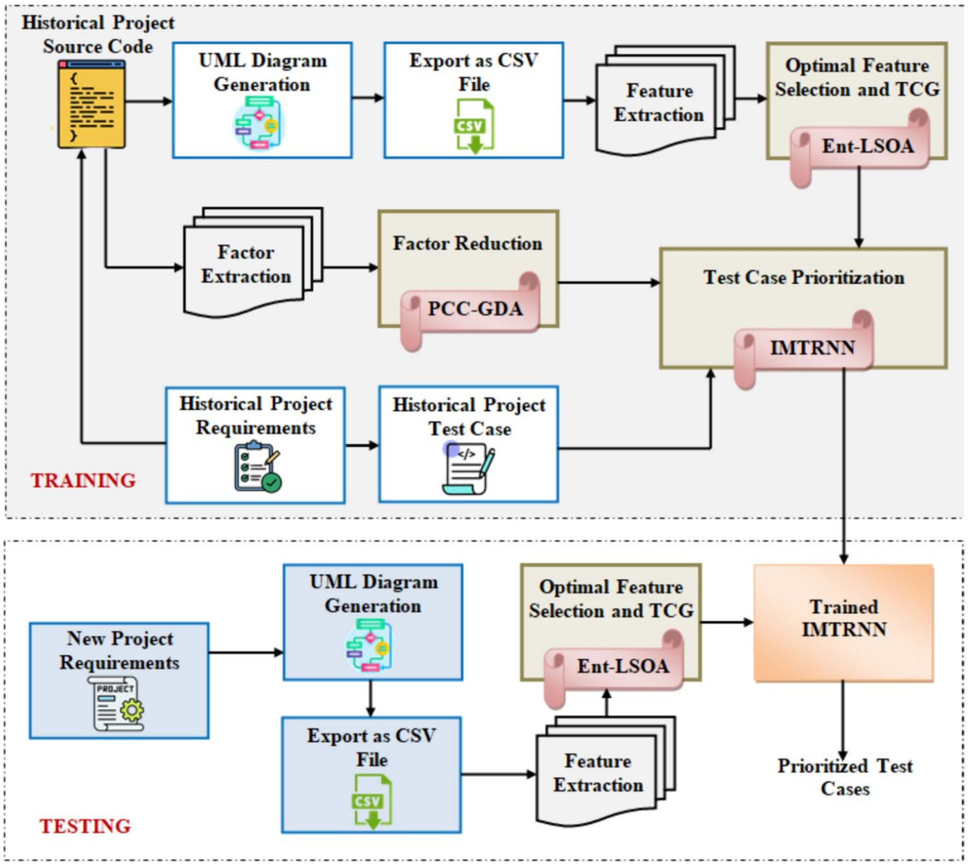


Fig. 1 Structural diagram of the proposed work

where, (a) represents the total numbers of historical source code data (S_a). Here, the (a^{th}) source code of the historical project such as ATM, banking, vehicle manufacturing, et cetera is specified as (S_a). After that, from (S_a), the UML diagram is generated, which is described further.

3.1.1 UML Diagram Generation and CSV File Exportation

UML is a general-purpose modeling language that visually represents the software system structure, behavior, and relationships. They aid in understanding code concepts, facilitating communication, testing, and feature extraction. Here, UML class diagrams are used, which aids in visualizing project structure, facilitating feature extraction, optimizing TCG, and prioritizing software testing. As of the source code, the UML is generated automatically using the code-based commands as shown in Eq. 3,

$$U_{gen} = reversecode(s_a) \tag{3}$$

where, (U_{gen}) represents the generated UML diagram and the command to generate the UML diagram is given as $reversecode\left(S_a\right)$.

Then,(U_{gen}) are exported as CSV files, which cover the diagram in a table format. The CSV conversion is given in Eq. 4 as,

$$C = csv.write\left(U_{gen}\right) \tag{4}$$

where, (C) the represents the CSV file and a command to convert the fields and data from the UML diagram to the CSV file is provided as $csv.write(U_{gen})$

3.1.2 Feature Extraction

Feature extraction plays a prominent role in the proposed work. It helps to gather the information of source code by removing redundant data. Also, it improves the accuracy during the classification of TCs. Therefore, the TCs' primary features like test name, test ID, objective, prerequisites, Form Name, Test Data, Test Scenario, Testcase Description, Step Details, etc. are extracted from the CSV file. Here, the feature of the objective represents the innovations of the project. Prerequisites state the requirements (integer, tools, parameters), which were needed to develop the source code. Step details denote the working strategy of the developed code. The taken features only have more relevant information about the TC. Also, it provides the bug information of each unit of the source code. Thus, by enriching test execution history data with lightweight code features, the prediction model's accuracy is improved. Also, centered on the actual failure data as well as execution times, the accuracy

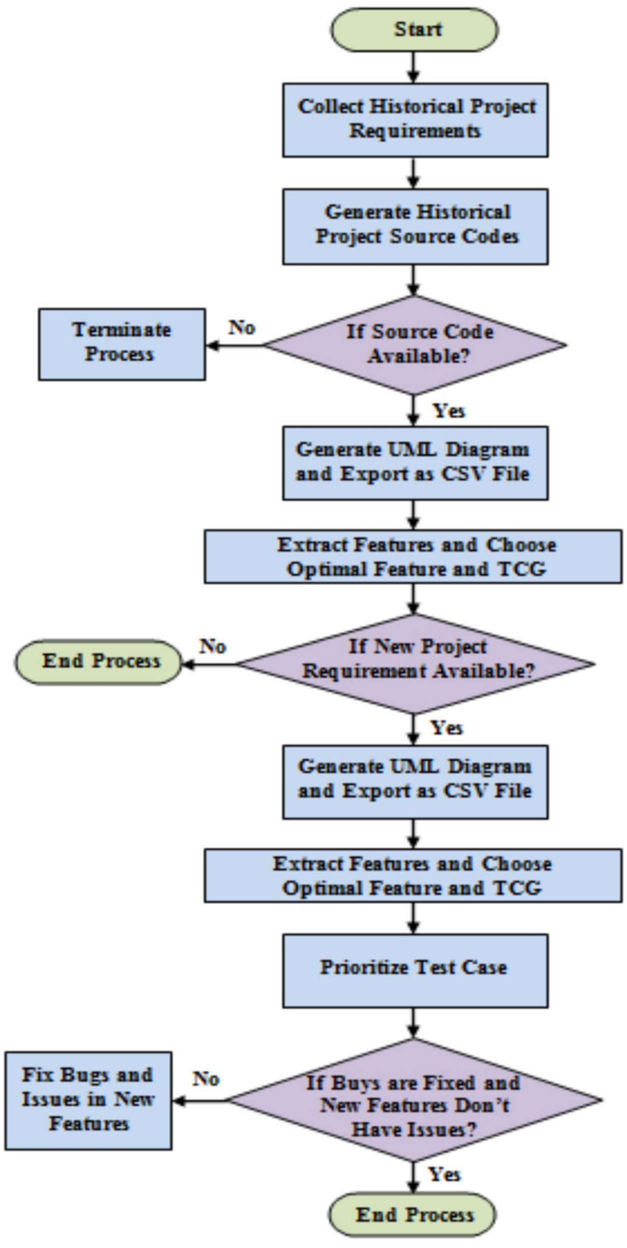

Start
Collect Historical Project Requirements
Generate Historical Project Source Codes
If Source Code Available?
No
Terminate Process
Yes
Generate UML Diagram and Export as CSV File
Extract Features and Choose Optimal Feature and TCG
If New Project Requirement Available?
No
End Process
Yes
Generate UML Diagram and Export as CSV File
Extract Features and Choose Optimal Feature and TCG
Prioritize Test Case
If Bugs are Fixed and New Features Don't Have Issues?
No
Fix Bugs and Issues in New Features
Yes
End Process

Fig. 2 Flowchart of the proposed work

is very close to the TCs' optimal ranking. The flowchart of the feature extraction process is shown in Fig. 3.

The flowchart is detailed below as,

Step 1: Start the extraction process.
Step 2: At first, the CSV file containing source code information is read.
Step 3: After a successful reading, the primary features, such as test name, test ID, objective, prerequisites, Form Name, Test Data, Test Scenario, Test case Description, and Step Details, are extracted from the CSV file as shown in Table 2.
Step 4: Then, redundant data is filtered out to reduce dimensionality and improve the accuracy of the feature set.
Step 5: After that, the relevant features for TC classification are identified and extracted from the filtered dataset.
Step 6: After extracting the relevant features, Lightweight code features are added to enrich the test execution history data, further improving the accuracy of the feature set.
Step 7: Bug information for each unit of the source code is then provided by enhancing the understanding of the code's behavior. The extracted features are then ready for use in TC classification. The working principle of feature extraction is given below.

The working principle of feature extraction revolves around identifying and isolating pertinent information from source code datasets. This process entails reading CSV files containing code information and extracting primary features like test names, IDs, objectives, prerequisites, and more. After that, the redundant data is filtered out to improve accuracy, and then the lightweight code features are added to enrich the test execution history. Bug information is provided for each code unit to refine the classification accuracy. The resulting feature set serves as input for optimization algorithms to select optimal features by ultimately facilitating effective test case prioritization and classification. The summarization of the feature extraction process is given in Table 2,

$$b_y = \left\{b_1, b_2, \ldots .., b_z\right\} where\ y = 1,2, \ldots , z \tag{5}$$

Here, the (z) represents the total numbers of (b_y). The pseudocode for Sect. 3.1.2 is given below in Algorithm 1,

Pseudo code of Algorithm 1

```
Input: CSV files
Output: Extracted features, $(b_y)$
Begin
    Start Process
    Initialize CSV file file with source code
    If CSV file is read,
        Extract primary features
    Else
        Return to read CSV file again
    End if
    Filter out redundant data
    If relevant features exists
        Enrich historical test data
    Else
        Return to filtering phase
    End if
    Provide bug information
    Extract features for TC classification
    End process
End
```

3.1.3 Optimal Feature Selection &TCG

From (b_y), the optimal features are selected and formed as optimal TCs. Here, the Ent-LSOA model has been proposed for the generation of optimal TCs via selected optimal features. (i) The solitary phase, and (ii) The social phase are

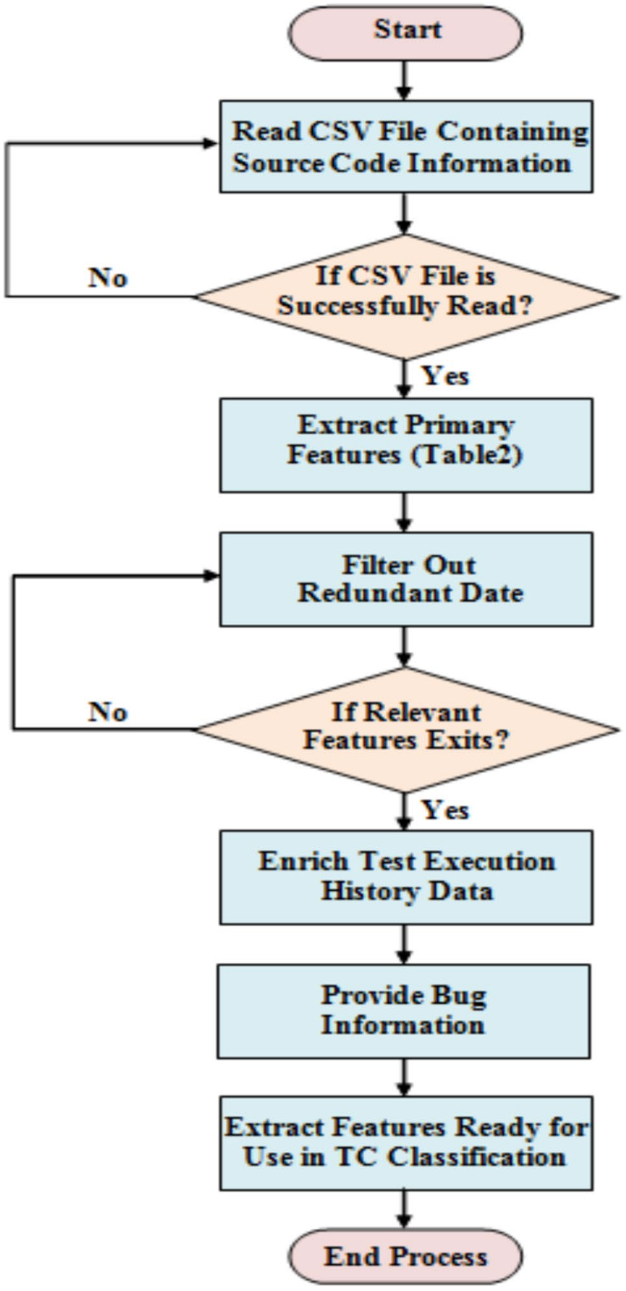

Start
Read CSV File Containing Source Code Information
If CSV File is Successfully Read?
No
Yes
Extract Primary Features (Table2)
Filter Out Redundant Date
If Relevant Features Exits?
No
Yes
Enrich Test Execution History Data
Provide Bug Information
Extract Features Ready for Use in TC Classification
End Process

Fig. 3 Flowchart regarding the feature extraction process

the 2 stages in the locust's search process. However, in the conventional LSOA, the social force exerted on every single locust will be modeled merely regarding the mean value, affecting the change of position at a lower level. The entropy strategy is subsumed in the social force phase in the conventional LSOA to resolve the aforementioned issue. The process of the Ent-LSOA algorithm is explained as follows,

The features $\left(b_y\right)$ are deemed to be the initial locust population. The locust population's initial position for kickstarting the evolutionary process is represented in Eq. 6,

$$b_y^I = \left\{b_1^I, b_2^I, ..., b_z^I\right\} \tag{6}$$

Here, the iteration number is indicated as (I) in $\left(b_y^I\right)$ and the population size is depicted as (z). (u, y, z) and so on are the different locust individuals enclosed in the locust population.

Initially, the locust $\left(b_y^I\right)$ is taken as of the locust swarm. The position of the locust $b_y^I = \left\{d_{y,1}^I, d_{y,2}^I, ..., d_{y,X}^I\right\}$ in the swarm given within the search boundary is given in Eq. 7,

$$B = \left\{b_y^I \in \lambda^X | Lb_e \leq b_{y,e}^I \leq Ub_e\right\} \tag{7}$$

where, the lower bound and upper bound in the dimension (d) are denoted as $\left(Lb_e, Ub_e\right)$, respectively. The boundary of the locust $\left(b_y^I\right)$ is notated as (λ). The locust that gets food is grounded on the fitness of the locust's position.

By utilizing $\left(b_y^I\right)$, the fitness is computed as $fit\left(b_y^I\right)$; subsequently, regarding the fitness, the locusts in Eq. (5) are ranked from 0 $to(z-1)$. Here, the fitness is deemed to be the classifier's accuracy.

Then, the exploration as well as exploitation phases are shown as follows,

(a) Exploration phase (Solitary stage)

During the search for food, the locusts change their position in the solitary stage. The new position $\left(b_y^*\right)$ of the locust $\left(b_y^I\right)$ in search of food is given in Eq. 8,

$$b_y^* = b_y^I + \partial b_y \tag{8}$$

where, the change of position of the locust is denoted as $\left(\partial b_y\right)$. The social interaction experienced by the locust $\left(b_y^I\right)$ with other elements in the group is the reason for the displacement of position. Here, by utilizing the entropy function, the social interaction betwixt the 2 locusts is computed. Thus, the social interaction of the locust $\left(b_y^I\right)$ and the locust $\left(b_u \in b_y\right)$ is formulated as shown in Eqs. 9 and 10,

$$\partial b_y = -\sum_{\substack{u=1 \\ u \neq y}}^{z} T_{yu}.\log\left(T_{yu}\right) \tag{9}$$

$$T_{yu} = \varepsilon\left(b_y^I, b_u^I\right) T\left(dis_{yu}^I\right) f_{yu} + r \tag{10}$$

where, the pairwise attraction or repulsion between the locust (y) and (u) is denoted as$\left(T_{yu}\right)$, the social factor is specified as $\left(T\left(dis_{yu}^I\right)\right)$, the fitness value between the locusts (y) and (u) is notated as$\left(\varepsilon\left(b_y^I, b_u^I\right)\right)$, the distance

Table 2 Summarization of feature extraction process

Total Number of Features	Feature Name	Data Type	Description
1	Test Name	String	Name of the TC
2	Test ID	Integer	Unique identifier for the TC
3	Objective	String	Purpose or objective of the TC
4	Prerequisites	String	Requirements or conditions needed for test execution
5	Formal Name	String	Name of the form or module being tested
6	Test Data	String	Data used as input for the TC
7	Test Scenario	String	Description of the scenario under which the TC is run
8	TC Description	String	StringDetailed description of the TC
9	Step Details	String	Step-by-step details of the TC execution
10	Bug Information	String	Information about any bugs encountered during test execution

From Table 2, the total number of features extracted was 9, and the records of those features depend on the size of the dataset that contains the source code information. These features provide essential information about each test case, including its name, purpose, prerequisites, input data, execution scenario, description, execution steps, and encountered bugs for effective test case classification and prioritization. The extracted features (b_y) are given in Eq. 5 as,

between two locusts is symbolized as$\left(dis_{yu} = \|b_y - b_u\|\right)$, the random number is depicted as(r), and the unit vector point from $\left(b_y^I\right)$ to $\left(b_u^I\right)$ is signified as $f_{yu} = \left(b_u^I - b_y^I\right) \Big/ dist_{yu}^I$. Then, the strength of the social vector $T\left(dis_u^I\right)$ is represented in Eq. 11,

$$T\left(dis_u^I\right) = \gamma \times \exp\left(-\frac{dis_{yu}^I}{R}\right) - \exp\left(-dis_{yu}^I\right) \tag{11}$$

where, the magnitude and length scale of attraction or repulsion are depicted as (γ, R). Also, regarding the rank of the locusts, the dominance value $\varepsilon\left(b_y^I, b_u^I\right)$ being gauged is given as a condition in Eq. 12,

$$\varepsilon\left(b_y^I, b_u^I\right) = \begin{cases} exp\left(-\frac{rn\left(b_u^I\right)}{z}\right) & if\ rn\left(b_u^I\right) > rn\left(b_y^I\right) \\ exp\left(-\frac{rn\left(b_y^I\right)}{z}\right) & if\ rn\left(b_y^I\right) > rn\left(b_u^I\right) \end{cases} \tag{12}$$

where, the rank of locusts, which is computed regarding the fitness, is denoted as $\left(rn\left(b_y^I\right), rn\left(b_u^I\right)\right)$, and the $\left(b_y^*\right)$ of locusts are updated according to the new positions as shown in Eq. 13,

$$b_y^* = \begin{cases} e_y\ if\ fit\left(b_y^*\right) > fit\left(b_y^I\right) \\ b_y^I else \end{cases} \tag{13}$$

where, the locusts in the better position in the swarm are denoted as $\left(e_y\right)$. Next, the optimal solution is selected from the social phase of the locusts.

(b) Exploitation phase (social behavior)

The process of refining existent locusts within a smaller swarm $Q = \left\{b_1^*, \dots b_z^*\right\}$ to enhance the solution quality is termed social behavior. The locusts in (Q) are the best locusts (features). Next, regarding the food quality index, the individuals in (Q) are sorted; then, from the sorted locusts, $\left(b_1^*\right)$ specifies the best position as well as $\left(b_z^*\right)$ depicts the worst position in the neighborhood. For every single $\left(b_y^* \in Q\right)$, a set of (lo) random locusts are generated $W^y = \left\{ra_1^y, ra_2^y, \dots, ra_{lo}^y\right\}$ within the respective subswarm $\left(O_u\right)$ of $\left(b_y^*\right)$ with radius (rad). The size of $\left(O_u\right)$ relies on the distance $\left(\mu_O\right)$, which is expressed in Eq. 14,

$$\mu_O = \frac{\phi \times \sum_{y=1}^{z}\left(Ub_y - Lb_y\right)}{z} \tag{14}$$

where, the lower and upper bound in the subspace are indicated as (Ub, Lb), and an adjustable parameter is notated as (ϕ). At last, the best solution from $\left(b_y^* \in Q\right)$ and its respective (lo) random solution is assigned as the position of the locust (y) in the iteration $(I+1)$ as shown in Eq. 15,

$$b_y^{I+1} = bst\left(b_y^*, ra_1^y, ra_2^y, \dots\dots, ra_{lo}^y\right) \tag{15}$$

where, the best solutions are symbolized as $bst\left(b_y^*, ra_1^y, ra_2^y, \dots\dots, ra_{lo}^y\right)$; then,$b_y^* \notin Q$ are removed in the social phase. The final position obtained after the social phase is expressed in Eq. 16,

$$b_y^{I+1} = \begin{cases} bst\left(b_y^*, ra_1^y, ra_2^y, \ldots, ra_{lo}^y\right) & if\left(b_y^* \in Q\right) \\ b_y^* i & f\left(b_y^* \notin Q\right) \end{cases} \tag{16}$$

The optimal features for the optimal TCG are obtained by updating the $\left(e_y^{I+1}\right)$ value. The optimal TCs generated the (g) optimal TCs $\left(t_g\right)$ regarding the optimal features; it is expressed in Eq. 17 as,

$$Ot = \left\{t_1, t_2, \ldots, t_g\right\} ort_j \tag{17}$$

The pseudocode for the proposed Ent-LSOA is given as follows,

Pseudo code of Ent-LSOA

```
Input: Extracted features $\beta = \{b_1, b_2, \ldots, b_z\}$
Output: Optimal test cases

Begin
    Initialize locust population $\{b_h^I\}$, $z$, initial iteration $I$, maximum iteration $I_{\max}$
    Calculate Fitness
    Set $I = 1$
    While $(I \le I_{\max})$ do
        Perform thesolitary phase using the $\partial b_y$
        Rank the locusts update $b_y^*$
        Divide into sub swarms $Q$
        Perform social behaviour
        For $(b_y^* \in Q)$ do
            Generate random locusts $W^y$ within the subswarm of $b_y^*$
            Update the position of the locusts in the subswarm
            If $(b_y^* \in Q)$ {
                Update position using $bst(b_y^*, ra_1^y, ra_2^y, \ldots, ra_{lo}^y)$
            } Else if $(b_y^* \notin Q)$ {
                Repeat For
            }
            End If
        End For
    End While
    $I = I + 1$
    Return optimal features $b_y^{I+1}$
    Generate optimal test cases $Ot$
End
```

In feature selection, the best-case scenario demonstrates a selection of features that are highly correlated with the effectiveness of test case prioritization, such as complexity metrics, historical failure rates, and code coverage. In contrast, the worst-case scenario illustrates the selection of features that may not directly impact the prioritization of test cases, such as coding style-related metrics and structural elements of the code.

The extracted features for test case generation and prioritization from the dataset include Test Name, Test ID, Objective, Prerequisites, Formal Name, Test Data, Test Scenario, TC Description, and Step Details. From the extracted features, a total of 7 features, namely Test Name, Test ID, Objective, Test Data, Test Scenario, TC Description, and Step Details are optimally selected. These optimally selected features provide insights into program behavior, communication patterns, developer intent, fault tolerance, and code quality. The loss of potentially relevant information in eliminated features, such as unused variables/methods, low-level code intricacies, and deprecated constructs made inadequate filtering criteria that lead to suboptimal features. This streamlined feature selection ensures focused analysis, which aids in accurate test case prioritization and efficient software testing strategies.

3.1.4 Factors Extraction

Parallel to the UML diagram generation, to train the classifier, factors like data flow, code coverage, customer-assigned priority, requirement implementation complexity, changes in the requirement, requirement volatility, traceability, fault proneness, and execution time from the source code $\left(s_\alpha\right)$ are extracted. The extracted factors are given in Eq. 18,

$$c_\delta = \left\{c_1, c_2, \ldots, c_x\right\} where\ \delta = 1,2, \ldots, x \tag{18}$$

Here, (x) represents the total number of extracted factors $\left(c_\delta\right)$.

3.1.5 Factors Reduction

Some of the insignificant factors, such as Tester Name, Tested By, Created by, Reviewed by, Test Date, Version, and Prerequisites are mitigated from $\left(c_\delta\right)$ to minimize the classifier's processing time. Here, by utilizing the PCC-GDA, the factors are mitigated. The Generalized Discriminant Analysis (GDA) is a methodology designed for feature mapping regarding the kernel function. The linearly separable data could be correctly discriminated by the conventional GDA. Hence, to discriminate the linearly separable data, the Pearson Correlation Coefficient (PCC) technique is applied within the class scatter matrix of GDA. The PCC-GDA procedure is given as follows,

Initially, the extracted factors' mean value (η) is computed as shown in Eq. 19,

$$\eta = \frac{\sum_{\delta=1}^{x} c_\delta}{x} \tag{19}$$

Then, every single factor gets divided and grouped into (Z) classes $\left\{cl_1, cl_2, \ldots, cl_Z\right\}$ from $\left(cl_t\right)$.

Next, the within-the-class and between-class scatter matrix is computed for the non-linearly mapped data. However, for enhancing the within-scatter matrix, the PCC is computed as shown in Eqs. 20 and 21,

$$Y_{win} = \frac{1}{Z}\sum_{t=1}^{Z}\frac{1}{G}\sum_{j=1}^{G}\frac{\lambda(c_{tj}-\eta).\lambda(c_{tj}-\eta)^T}{\sqrt{\Sigma_{ij}(c_{tj}-\eta)^2\Sigma_{tj}\left((c_{tj}-\eta)^T\right)^2}} \tag{20}$$

$$Y_{bn} = \frac{1}{Z}\sum_{t=1}^{Z}(\eta_t-\eta)^T(\eta_t-\eta) \tag{21}$$

where, (Y_{win}, Y_{bn}) denote the within the class and between the class scatter matrix, respectively, the total number of factors in the class t is signified as G, the factors in the class cl_t are symbolized as $(j \in c_\delta)$, and the j^{th} factor in the class (t) is notated as (c_{tj}). The mean of the factors in the class (t) is depicted as (η_t).

After that, in the GDA, to satisfy the condition, Eigenvector (k) and Eigenvalue (ϖ) are computed as shown in Eq. 22,

$$\varpi \times Y_{win} \times k = Y_{bn} \times k \tag{22}$$

As the solutions lie in the span $[\lambda(c_{11}), ..., \lambda(c_{tj})]$, the coefficient ϑ_{tj} that exists in this span could be illustrated utilizing the Eigenvalues, which are represented in Eq. 23,

$$k = \sum_{t=1}^{Z}\sum_{j=1}^{G}\vartheta_{tj} \times \lambda(c_{tj}) \tag{23}$$

Then, by utilizing the kernel function in GDA, the dot product of the two factors c_l, c_m from the two classes $cl_i, cl_o \in cl_t$ is performed for mapping the factors in the feature space. The feature mapping $(p_{lm})_{io}$ kernel function is expressed in Eq. 24,

$$(p_{lm})_{io} = \lambda(c_{il}).\lambda(c_{om}) = p(c_{il}, c_{om}) = \exp\left(-\frac{|c_{il}-c_{om}|^2}{\chi}\right) \tag{24}$$

where, the random number is depicted as χ, the factors l, m in the class i, o are indicated as c_{il}, c_{om}, respectively, and the kernel function is signified as $(p(c_{il}, c_{om}))$. Then, the $R \times R$ matrix L_{io} is constructed, which encompasses the dot product betwixt class i and o in the feature space M as given in Eq. 25,

$$L_{io} = (p_{lm})_{l=1,2,...,i,\ m=1,2,...,o} \tag{25}$$

Moreover, here, for identifying the optimal features, a $R \times R$ block diagonal matrix is introduced; it is expressed as shown in Eq. 26,

$$MT = (MT_t)_{t=1,2,.....,Z} \tag{26}$$

Here, the matrix (MT_t) is a $G \times G$ matrix with all elements equal to $(1/G)$; then, by substituting Eqs. (19), (20), and (23) in Eq. (21) and by taking the inner product with vector $\lambda(c_{lm})$ on both sides of Eq. (21), the solution could be obtained. The solution is shown in Eq. 27,

$$\rho LL\upsilon = L.MT.L\upsilon \tag{27}$$

where, the column vector with the elements $(\sigma_{tj}, t = 1,2,...,Z, j = 1,2,...,G)$ is depicted as (υ). By evaluating the Eigenvectors of the matrix $(LL)^{-1}L.MT.L$, the (υ) could be computed. Initially, by diagonalizing the matrix (L), significant eigenvectors are found if the matrix (L) is non-reversible.

A projection matrix (Pr) is then constructed after finding $(\Re)$ significant eigenvectors; it is modeled as shown in Eq. 28,

$$\Pr = [\upsilon_1\upsilon_2\upsilon_3.....\upsilon_\Re] \tag{28}$$

where, the significant eigenvectors obtained in the dimension of the features to be mapped are denoted as $(\Re)$.

Next, the projection of (c_δ) in the $(\Re)$ dimensional GDA space is computed in Eq. 29,

$$\varsigma = \Pr \times p_{c_\delta} \tag{29}$$

where, the feature mapping process is denoted as $p_{c_\delta} = [p(c_\delta, c_{11})......p(c_\delta, c_{tj}).......p(c_\delta, c_{ZG})]$. Hence, the factor set after reduction (A) is formulated in Eq. 30,

$$A = \{c_\delta\} \ where\ \delta = 1,2,3,...,x \tag{30}$$

3.1.6 Test Case Prioritization

Finally, the optimal TCs (t_j), optimal factors (c_δ), and historical TCs (h_{test}) are utilized to train the proposed IMTRNN classifier for generating prioritized TCs. Every single step input in IMTRNN relies on the preceding stage output. In MTRNN, there are 3 major stages: (i) the input state that captures the input data, (ii) the output state that captures the classifier's output results, and (iii) the recurrent state, which is a series of hidden states that captures all the computations betwixt the input and output states. Nevertheless, in a conventional Recurrent Neural Network (RNN), a neuron dies when its weights get tweaked; in the same manner, a weighted sum of its inputs is negative for all instances in the training set. Therefore, to minimize the error, only the interpolation-based Kullback-Leibler divergence technique is utilized. Thus, in Fig. 4, the working procedure of IMTRNN is explicated.

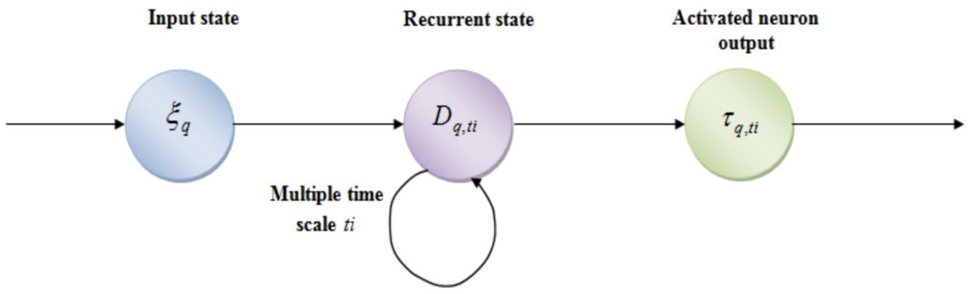


Fig. 4 The neural representation of the proposed IMTRNN

Input state: Here, the input data (t_j), (c_δ) and (h_{test}) are collectively termed as (ζ_q). Then, the input state prepares all the inputs (ζ_q) for further processing.

Hidden state: The present hidden node (D_q) is computed regarding the present input (ζ_q) and preceding hidden state output (D_{q-1}). The IMTRNN's hidden node that stores memory for sequential information processing is expressed in Eq. 31,

$$\omega_q.D_{q,ti} = -D_{q-1,ti} + \sum_{\ell=1}^{tot} \psi_{q,ti} \times \zeta_{q,ti} \tag{31}$$

where, the activity of the q^{th} neuron at the multi-time step (ti) is depicted as $(\zeta_{q,ti})$. The speed of adaption of neurons is determined by the time step. The weight value of the q^{th} neuron is depicted as $(\psi_{q,\ell})$ and a scale parameter that determines the decay rate of the neuron is notated as (ω_q). When the q^{th} neuron has a $(\aleph)$ number of connections, the Eq. (31) could be expressed as shown in Eq. 32,

$$D_{q,ti+1} = \left(1 - \frac{1}{\omega_q}\right) D_{q,ti} + \frac{1}{\omega_q}\left(\sum_{\ell \in \aleph} \psi_{q,ti} \times \zeta_{q,ti}\right) if\ ti > 0 \tag{32}$$

Here, If $(ti = 0)$, then the hidden node's activity is assumed to be 0 and the context neurons are set to the initial state.

Next, by utilizing sigmoid and softmax activations, the neural activity of the neurons is computed according to the level of the neurons. The activation calculation is formulated in Eq. 33,

$$\tau_{q,ti} = \begin{cases} \frac{\exp(\zeta_{q,ti})}{\sum_{q=1}^{\aleph} \zeta_{q,ti}} & if\ q \in i/o \\ \frac{\exp(\zeta_{q,ti})}{\zeta_{q,ti}+1} & else \end{cases} \tag{33}$$

The activated neuron output is termed $(\tau_{q,ti})$, i/o, which specifies the input as of the preceding neuron output. Here, by utilizing the interpolation with the Kullback–Leibler divergence technique, the error function is computed. The IMTRNN's error calculation is mathematically formulated in Eq. 34,

$$\varphi_{er} = \sum_{ti} \sum_{q=1}^{\aleph} \tau^*_{q,ti} + \left(\tau^*_{q,ti} - \tau_{q,ti}\right) \log\left(\frac{\tau^*_{q,ti} - \tau_{q,ti}}{\tau_{q,ti} - \tau^*_{q,ti}}\right) \tag{34}$$

where, the error function is depicted as (φ_{er}), and $\left(\tau^*_{q,ti}\right)$ indicates the expected activation at the time step (ti) for the neuron (q), which acts as the target value for the output $(\tau_{q,ti})$. If the error is less than or equal to the target threshold $(thresh)$, then the output is considered to be the final output, or else the training continues. In the IMTRNN, the next time step $(t + 1)$ of the input neuron is computed regarding the feedback rate of the $\left(\tau^*_{q,ti}\right)$ and $(\tau_{q,ti})$ as given in Eq. 35,

$$\zeta_{q,ti+1} = (1 - \wp) \times \tau_{q,ti} + \wp \times \tau^*_{q,ti} \tag{35}$$

Here, varied values are utilized for $(\wp)$ based on training and generating prioritized TCs. Moreover, to obtain an optimal solution, the weight values are updated for larger error values. Therefore, the updated weight computation for the input (ζ_q) is given in Eq. 36,

$$\psi^{Iter+1}_{q,ti} = \psi^{Iter}_{q,ti} - \xi_{q,ti} \frac{\partial \varphi_{er}}{\partial \psi_{q,ti}} = \psi_{q,\ell} - \frac{\xi_{q,ti}}{\omega_q} \sum_{ti} \frac{\partial \varphi_{er}}{\partial \psi_{q,ti}} \tag{36}$$

where, the updated weight value in the iteration is $\left(\psi^{Iter+1}_{q,ti}\right)$, $(Iter + 1)$, and $(\xi_{q,ti})$ is the learning rate for the weight value changes. Finally, the data from each neuron are summed and the output is provided. This classifier significantly classifies the TCs orderly. The trained prioritized TCs (τ_q) are expressed in Eq. 37 as,

$$\tau_q = \left\{\mathfrak{I}_1, \mathfrak{I}_2, \ldots\ldots, \mathfrak{I}_K\right\} \tag{37}$$

where, the (K^{th}) prioritized TC is depicted as $(\mathfrak{I}_K)$. Here, the TC with high priority is notated as $(\mathfrak{I}_1)$, and the TC with low priority is symbolized as $(\mathfrak{I}_K)$. Based on the prioritized TCs, the software is reconstructed by its ordering. It assists in meeting '2'vital constraints, namely time and budget in software testing for enhancing the fault detection rate as early as possible. The pseudocode for the proposed IMTRNN is given as,

Pseudo code of IMTRNN

```
Input: Data ($t_j$, $c_\delta$ and $h_{test}$) or $\zeta_q$
Output: Prioritized data $\tau_q$

Begin
    Initialize input state $\zeta_q, \psi_{q,ti}, Iter$, maximum iteration $Iter_{\max}$, $thresh$
    Set iteration $Iter = 1$
    While $(Iter \leq Iter_{\max})$ do
        Compute Hidden layer process $\omega_q D_{q,ti} = -D_{q-1,ti} + \sum_{\xi=1}^{tot} \psi_{q,ti} \times \zeta_{q,ti}$
        Update hidden layer input at the time step $t+1$
        Activate output neurons $\tau_q$
        Evaluate error $\phi_{er} = \sum_{ti}\sum_{q=1}^{\aleph} \tau^{*}_{q,ti} + (\tau^{*}_{q,ti} - \tau_{q,ti})\log\left(\frac{\tau^{*}_{q,ti} - \tau_{q,ti}}{\tau_{q,ti} - \tau^{*}_{q,ti}}\right)$
            If $(\phi_{er} \leq thresh)${
                Select output $\tau_q$
            } Else {
                Update weight values $\psi^{Iter+1}_{q,ti}$ at the time step $ti$
            }
            End If
    End While
    Return $\tau_q$
End
```

The criteria for optimal TCG involves selecting relevant features and factors from historical project source code and TCP criteria prioritize TCs based on their ability to detect faults efficiently, aiming for high accuracy, and recall in software testing. The testing phase is further described.

Table 3 displays the generated TCs with prioritized values, ranging from high to low. Also, the Historical Project dataset related to Table 3 is collected from open source repository. Additionally, it highlights that each test case corresponds to a specific functionality or scenario in a software application. The caption of Table 3 emphasizes the significance of prioritization in focusing testing efforts on critical functionalities first while ensuring efficient resource utilization and timely issue detection to enhance overall software quality. Prioritization enables focusing the testing efforts on critical functionalities first by ensuring the efficient use of resources and timely detection of potential issues. This prioritization scheme aids in optimizing testing processes by addressing high-risk areas early followed by considering lower-priority functionalities for comprehensive testing, thus enhancing overall software quality. For instance, validating login with valid credentials (TC001) holds the highest priority, followed by other critical functionalities like login validation and account balance retrieval as shown in Table 3. This prioritization scheme optimizes testing efforts by addressing high-risk areas first to ensure efficient resource utilization and timely issue detection. Lower-priority functionalities, such as user profile updates and email notification verification are subsequently tested to enhance the overall software quality improvement.

Table 3 Generated test case with prioritized values

Generated Test Case ID	Description	Prioritized Values
TC001	Validate login with valid credentials	High (1)
TC002	Validate login with invalid credentials	High (1)
TC003	Validate registration with valid data	High (1)
TC004	Validate registration with invalid data	Medium (2)
TC005	Verify account balance retrieval	High (1)
TC006	Test fund transfer functionality	High (1)
TC007	Test bill payment feature	Medium (2)
TC008	Verify the password reset feature	Medium (2)
TC009	Test user profile update	Low (3)
TC010	Verify email notification system	Low (3)

3.2 Testing Phase

When a new project requirement is given, a UML diagram is provided by the proposed methodology to provide both lower-level as well as higher-level insight into the concept along with the design of an application; then, the features are transmuted into a CSV file. The features, such as test name, test ID, objective, prerequisites, etc. are extracted from the CSV file. The extracted features are helpful for improving the classifier's accuracy by gathering the information from the project source code; subsequently, using Ent-LSOA, the optimal TCG is done. The main motive behind the optimal TCG phase is to reduce the computation burden and testing time of the classifier. Next, the optimal TCs are inputted into the IMTRNN for testing. Thus, the output is obtained as prioritized TCs regarding the optimal factors as well as historical TCs. The execution time of each section is given in Table 3,

Figure 5 describes the Time Complexity (TC) in each section based on input size. The TC for generating the test case

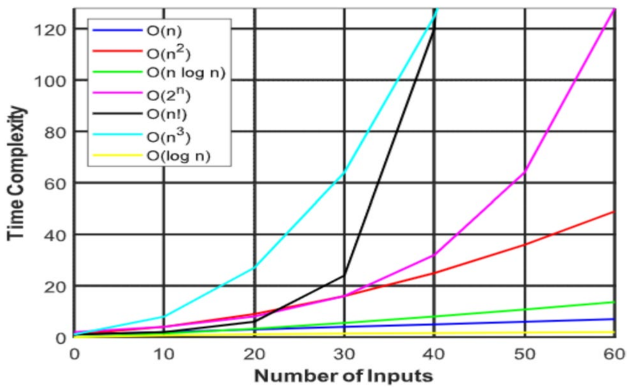


Fig. 5 Time complexity in each section with big O notation

generation and test case prioritization is very high, which is described as O(n^3) and O (n!) as shown in graph 5. Also, for generating the UML diagram O(2^n) and factor extraction O(n^2), the TCs are moderately high. Then, for feature selection O(n log n), factor reduction O(n), and feature extraction O(log n), the TCs are very low.

The overall proposed system starts with the collection of historical project requirements, which are then converted to source code for generating the UML diagram. From the UML diagram, the CSV files are exported. After that, the feature extraction is carried out and then the extracted features serve as the input for the Ent-LSOA algorithm, which generates optimal test cases by leveraging entropy-based social interaction techniques. These optimal test cases are then fed into the IMTRNN algorithm for test case prioritization. IMTRNN utilizes machine learning to prioritize test cases based on factors like precision, recall, and accuracy, resulting in enhanced bug detection capabilities and improved code coverage. After prioritization, the prioritized test cases are checked for bugs and issues to get effective and accurate prioritized test cases. Following prioritization, the framework undergoes evaluation and analysis by comparing its performance with existing methodologies, such as RNN, CNN, MLP, RBFN, and DBN. This analysis provides insights into the effectiveness of the proposed approach, highlighting its advantages in terms of precision, recall, accuracy, and other metrics.

4 Case Study of the Proposed Work

The proposed work starts by acknowledging the limitations of existing TCG and TCP methods, particularly in terms of time complexity, computational efficiency, and adaptability. Hence, to overcome the issues, the proposed work leveraged innovative techniques, such as Ent-LSOA, PCC-GDA, and IMTRNN for effectively prioritizing the generated TCs in software testing. This work begins with a thorough analysis of historical project source code to understand past testing patterns and to identify the areas for improvement. Here, the historical project requirements are translated into source code and then visualized using UML diagrams. Then, these diagrams are exported into CSV files to facilitate the feature extraction process. Feature extraction plays a crucial role in gathering pertinent information from the source code for filtering out redundant data and also for enhancing classification accuracy during TCP. Therefore, features like test names, objectives, prerequisites, and step details are extracted to provide insights into program behavior and fault tolerance. Meanwhile, factors, such as data flow, code coverage, customer-assigned priority, requirement implementation complexity, changes in the requirement, requirement volatility, traceability, fault proneness, and execution time are extracted from historical source code. Then, by utilizing a PCC-DGA technique, the project streamlined the selection of relevant source code factors while minimizing the computational overhead. Additionally, an Ent-LSOA is employed to optimize the TCG process by selecting the optimal features through social interaction techniques based on entropy principles. Thus, this work minimizes the computational burden and testing time to enhance efficiency. Subsequently, the IMTRNN algorithm prioritizes TCs based on factors like precision and recall, optimizing bug detection capabilities, and code coverage. This comprehensive approach effectively prioritizes the TCs in real-time scenarios. Through these efforts, the proposed work seeks to revolutionize software testing practices by optimizing efficiency and enhancing software quality.

Here, the UML diagram is generated using the Pylint package from the source code. The UML diagrams are then opened and read using (with open(uml_file, "r") as f: and uml_data = f.read()). These diagrams are then converted to csv files using (csv.writer) function. Then, the class names and attributes are then split and separated using (tokens = line.split()) function. After that, based on high accuracy, the optimal features are selected using Ent-LSOA algorithmic codes. Also, these are theoretically explained below,

- UML Diagram Generation

  This part of the code is responsible for converting UML (Unified Modeling Language) diagrams into a CSV (Comma Separated Values) format. It starts with reading the UML file by splitting it into lines and then parses each line to extract class names, attributes, and methods. These extracted data are then written into a CSV file.
- Optimal Feature Selection using Ent-LSOA

  This section implements a swarm optimization algorithm called the Entropy Locust Swarm Optimization Algorithm (Ent-LSOA). It consists of two main classes: `EntropyLocust` and `EntropyLocustSwarm`. The locusts move randomly within a bounded space, and their fitness is evaluated based on maximum classification accuracy. The swarm updates the locusts' positions, calculates their fitness, and then finds the best locust with the highest fitness.
- Testcase Prioritization using IMTRNN

  Here, a Recurrent Neural Network (RNN) model is defined using PyTorch to prioritize test cases. The RNN architecture consists of an input layer, an RNN hidden layer, and a fully connected layer. RNN is trained using a binary cross-entropy loss function and optimized using the Adam optimizer. The model is then trained for a specified number of epochs that are evaluated for accuracy on a test dataset.
- Factors Reduction using Pearson Correlation Coefficient-Generalized Discriminant Analysis (PCC-GDA)

This part involves reducing factors/features using the Pearson Correlation Coefficient (PCC) and Linear Discriminant Analysis (LDA). It calculates the correlation between features and the target variable and then fits an LDA model to the data. Finally, it makes predictions and calculates the accuracy.

- Historical Project Testcase Generation

  This section defines the functions to generate test cases for a given function and inputs. It randomly selects the inputs, calculates expected outputs using the function, and generates a test case consisting of input–output pairs. Additionally, it provides a function to generate multiple test cases at once.

  Each component serves a specific purpose, from data manipulation (UML to CSV conversion, feature reduction) to optimization (feature selection, test case prioritization) and testing (test case generation). Together, these functionalities cover a range of tasks involved in software development and testing.

5 Results and Discussion

Here, the proposed framework's performance is experimentally evaluated in comparison with the prevailing methodologies. The experiments are executed in JAVA with the synthetically created dataset, which is the collection of source code of historical projects.

5.1 Dataset Description

The proposed work is implemented in the working platform of JAVA, which is selected for its versatility and widespread adoption across industries. The JAVA programs created for various applications like banking, healthcare, and finance will be downloaded from publicly available sources on the Internet. Moreover, Java libraries like DL4J enable the implementation of the neural network and also help in assessing how efficiently the model prioritizes the generated Test cases. At last, regarding fitness, TCG time, precision, accuracy, f-measure, recall, training time, along with specificity, the proposed model's performance is analyzed experimentally. Thus, the outcomes exhibited that the proposed methodologies outperformed the state-of-the-art techniques. Here, the Historical Projects Dataset is collected from SourceForge and open source repository. The dataset totally contains 8,00,000 projects with more than two million registered users. For TCG, the JAVA program in the proposed work considered a total of 100 projects. Thus, the dataset link is given below, https://zenodo.org/records/268466.

5.2 Tools Required

Tools required in the proposed work for prioritizing the test cases include:

1. Eclipse IDE
2. UML Diagram Integration with IDE

   Here, Eclipse IDE provides a comprehensive development environment for coding and project management, while UML Diagram Integration enhances code visualization for better understanding

1. Eclipse IDE

   Eclipse Integrated Development Environment (IDE) provides a robust platform for software development by offering features, such as code editing, debugging, and project management. Eclipse IDE's extensibility through plugins makes it suitable for various programming languages and development tasks for facilitating efficient coding and project organization.
2. UML Diagram Integration with IDE

   Integrating UML diagram functionality into the IDE streamlines the software development process by enabling the developers to visualize system architecture directly within their coding environment. This integration enhances code comprehension, promotes better design practices, and facilitates communication for better understanding. Thus, UML diagram integration ultimately leads to more efficient and reliable software.

5.3 Performance Analysis

The experiments are conducted on two segments, namely optimal TCG and TCP to verify the proposed model.

5.4 Performance analysis of the optimal test case generation

In this segment, regarding the TCG time, the proposed optimal TCG algorithm Ent-LSO model's performance is comparatively analyzed with the prevailing TCG algorithms like LSOA, Sooty Tern Optimization Algorithm (STOA), Dwarf Mongoose Optimization Algorithm (DMOA) and Emperor Penguin Optimization Algorithm (EPOA) models. Also, to verify the proposed test generation techniques, the fitness vs iteration analysis is described in Table 4.

The experimental outcomes obtained during the fitness vs iteration analysis for the proposed Ent-LSOA and the existing algorithms are signified in Table 5. The fitness evaluation is performed to validate how fit the algorithm selects

Table 4 Execution time in each section

Sections	Execution Time (ms)
UML Diagram Generation	**12476 ms**
Feature Extraction	**10453 ms**
Feature Selection	**9463 ms**
Factor Extraction	**11435 ms**
Factor Reduction	**8563 ms**
Testcase Generation	**13457 ms**
Testcase Prioritization	**32457 ms**

optimal TCs that are needed for prioritization. Here, in the 30th iteration, the proposed algorithm's fitness is 3454, which is greater than the fitness of the prevailing models like LSOA (2965), STOA (2354), and EPOA (1454). The addition of entropy in the social interaction of locusts made this enhancement in the outcome. Hence, it is evident that better optimal TCs were generated by the proposed model than the prevailing techniques.

The proposed Ent-LSOA is analogized with the prevailing LSOA, STOA, DMOA, and EPOA models regarding TCG time in Fig. 6. The time taken to generate optimal TCs should be as low as possible for an effectual TCG system. Here, to complete 50 iterations, the proposed algorithm takes only 38654 ms, which is 14870 ms faster than the DMOA and 4004 ms faster than the LSOA. The existing LSOA attained the TCG time of 23654ms (iteration 10), 27658ms (iteration 20), 32458 ms (30 iterations), 37546ms (iteration 40), and 42658ms (50 iterations). The time taken for the TCG generation by STOA and DMOA at the iteration of 50 is 47548ms and 53524ms, respectively. Likewise, the EPOA takes the time for the iterations of 38654 (iteration 10), 42654ms (iteration 20), 47541ms (iteration 30), 52658ms (40 iterations), and 58652 (50 iterations). The improvement of the proposed Ent-LSOA is attained by calculating the position changing at a low level using the entropy technique. Thus, it is evident that the proposed model generated optimal TCs faster than the prevailing algorithms.

Table 5 Fitness vs iteration analysis

Iterations	Proposed Ent-LSOA	LSOA	STOA	DMOA	EPOA
10	2547	1974	1452	905	445
20	2965	2447	1864	1345	1175
30	3454	2965	2354	1975	1454
40	3845	3437	2854	2444	1745
50	4568	3965	3454	2841	2236

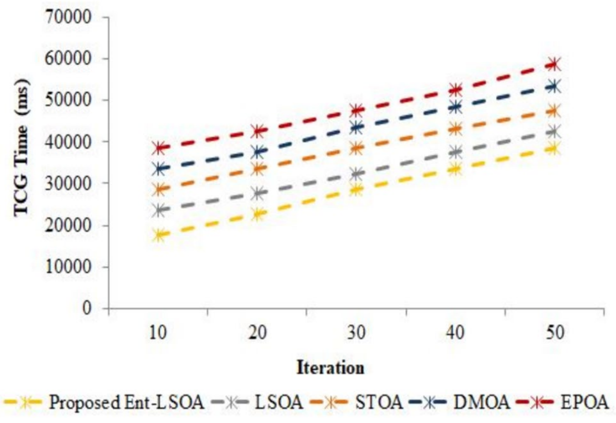


Fig. 6 TCG time for 50 iterations

The optimal number (most prioritized) of TCs selected out of 100, 200, 300, 400, and 500 TCs by the proposed and prevailing optimization algorithms is demonstrated in Table 5. The optimal features selected out of 300 TCs are illustrated here. In the proposed Ent-LSOA approach, out of 300 TCs, 204 were selected as the most prioritized TCs; conversely, in the existing LSOA, STOA, DMOA, and EPOA models, 218, 237, 258, and 278 TCs were selected as the most prioritized TCs in which some insignificant TCs were also selected as optimal TCs. Thus, it is proved that for optimal TCG, the proposed Ent-LSOA was suitable as it selects the most prioritized TCs better than the other existing algorithms.

Figure 7 describes the optimal number of test cases (TCs) that are selected from varying pools of 100, 200, 300, 400, and 500 TCs by both the proposed Ent-LSOA and existing LSOA, STOA, DMOA, and EPOA algorithms (Table 6). Focusing on the selection process from a pool of 300 TCs, the Ent-LSOA approach prioritized 204 TCs as optimal. In contrast to that, the other models, such as LSOA, STOA, DMOA, and EPOA identified 218, 237, 258, and 278 TCs, respectively, as most prioritized. Notably, the Ent-LSOA method demonstrated superior performance by reinforcing its suitability for optimal TCG when compared to existing algorithms. Ent-LSOA also quantifies uncertainty that aids in the selection of the most critical TCs. Therefore, the software is tested by optimizing the TCG.

The convergence curve in feature selection illustrates the relationship between the number of iterations and the convergence of the optimization algorithm. It provides insight into how quickly the algorithm converges towards an optimal solution and aids in determining the appropriate stopping criterion for assessing the algorithm's performance and efficiency. Therefore, as shown in Fig. 8, the convergence curve for the proposed Ent-LSOA reaches the

Fig. 7 Prioritization of Optimal Test Cases

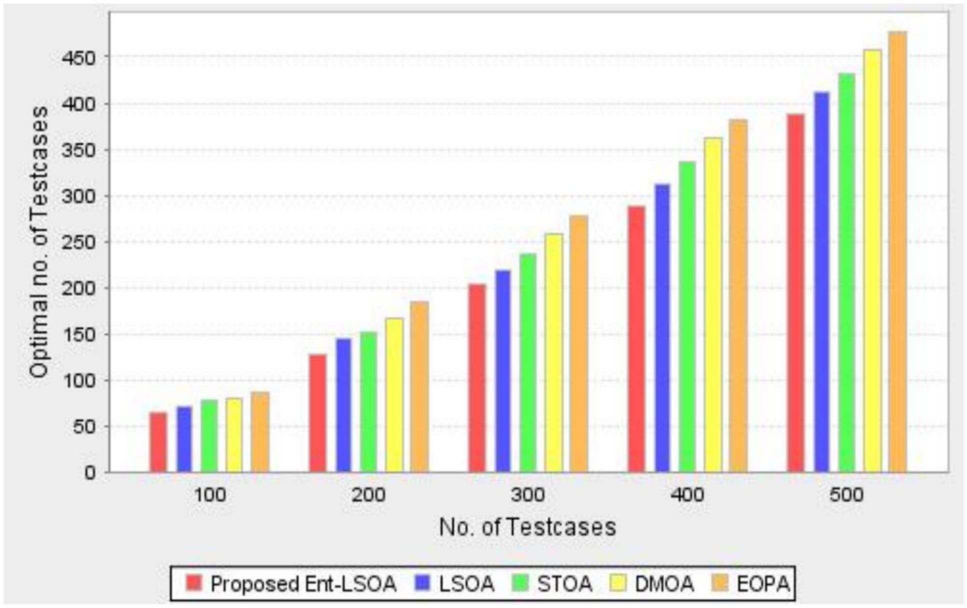


Table 6 Comparative analysis of selected optimized test cases

Number of test cases	100	200	300	400	500
Proposed Ent-LSOA	64	128	204	289	389
LSOA	71	145	218	312	41
STOA	78	152	237	337	431
DMOA	80	168	258	362	458
EOPA	87	184	278	381	476

peak value (converges towards an optimal solution) in the 400th iteration due to the enhanced inclusion of the Ent technique. But, the traditional LSOA, STOA, DMOA, and EPOA reach the peak value in the 595th, 690th, 710th, and 1300th iterations, respectively. Hence, when compared to the proposed work, the existing works take maximum duration to converge towards the best solution, thus degrading the overall performance.

Fig. 8 Convergence curve

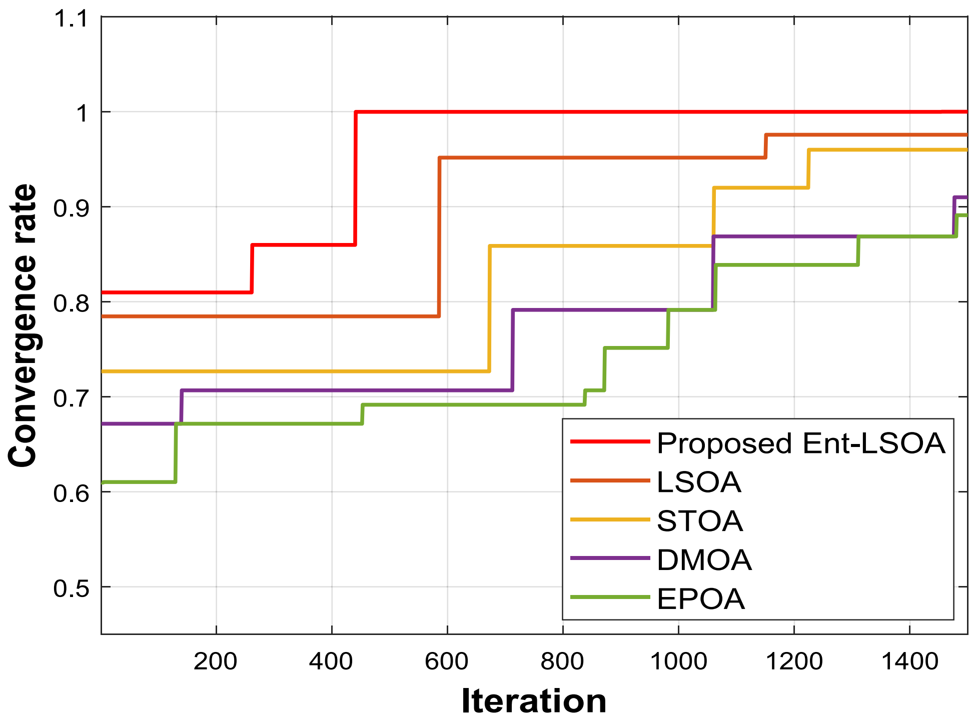


Table 7 Training time of the TCP classifiers

Techniques	Training Time
Proposed IKL-MTRNN	89654
RNN	95874
CNN	106589
MLP	112475
RBFN	128754
RBM	135474
DBN	145874
DNN	153258

5.5 Performance Analysis of the Test Case Prioritization

Here, Deep Belief Network (DBN), Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), and Deep Neural Network (DNN) are the prevailing methodologies with which the proposed ML TC prioritization algorithm IMTRNN is compared regarding, precision, accuracy, f-measure, recall, training time, along with specificity.

The time taken to train the proposed IMTRNN classifier and the existing RNN, DBN, CNN, MLP, RBFN, RBM, and DNN classifiers is shown in Table 7. For training, the RNN takes less time (95874ms) than the other prevailing algorithms. Nevertheless, the proposed IMTRNN takes 89654ms less time than the RNN scheme. Also, the training time of the existing models, such as CNN(106589ms), MLP(112475ms), RBFN(128754ms), and RBM (135474ms) is higher than the proposed model. The implementation of the interpolation technique in the MTRNN classifier obtained better enhancement in the outcome. Thus, it is proved that the lesser time taken by the IMTRNN makes it appropriate for the proposed TCP model.

In Fig. 9, the proposed TCP algorithm IMTRNN is analogized with the prevailing models regarding precision outcomes. The metric utilized to evaluate the number of TCs perfectly prioritized by classifier algorithms is termed precision. The metric 'precision' depicts the positive classes of the test cases, which have higher priority as same. The precision of the proposed method is analyzed to correctly prioritize the test cases by the software testers so that any critical issues in the code can be addressed earlier. Here, the proposed IMTRNN's precision is 2.13% higher than the existing RNN, 13.4% higher than the DBN algorithm, and 15% higher than the DNN algorithm.RBFN and RBM attained a precision level of 88.6% and 85.6%, respectively. Likewise, the MLP attained a 90.8% precision level. Thus, it is evident that in contrast to the prevailing algorithms, the proposed IMTRNN prioritized the TCs more correctly.

The graphical representation of the recall result obtained during the experimental analysis is illustrated in Fig. 10. To verify whether the prioritized TCs as well as the number of TCs definitely have the correct priority, the recall for the classifier algorithms is evaluated. The recall measure specifies the correct prediction of highly prioritized test cases among the entire historical source code, which consists of both the high and low-prioritized test cases. So, the recall score is analyzed for the proposed test case prioritization method to execute the test cases efficiently and prevent the risk of failures due to errors during the software testing scenario. Here, the proposed IMTRNN algorithm's recall is 96.92%, which is higher than the prevailing RNN (94.8%), CNN (91.5%), RBFN

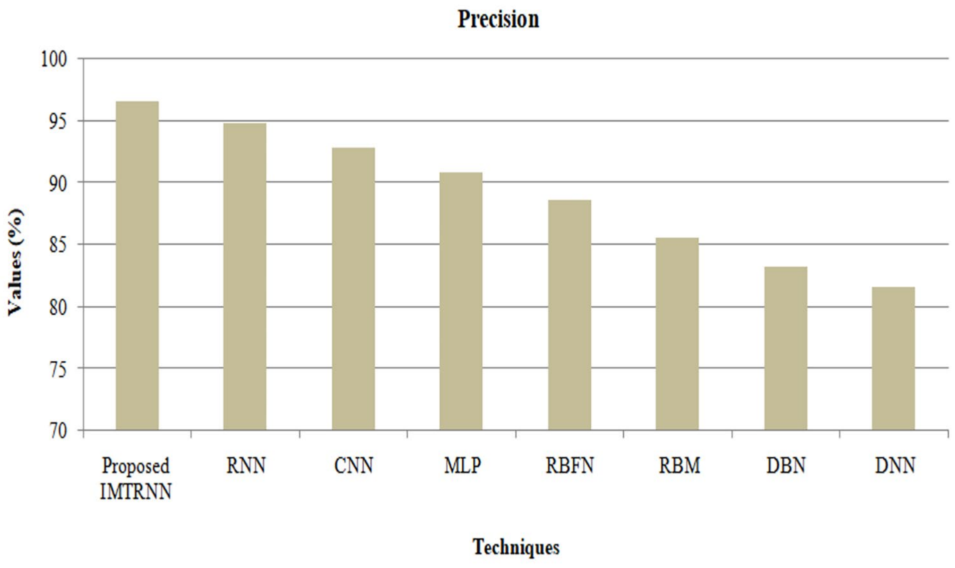


Fig. 9 Precision analysis of the proposed IMTRNN

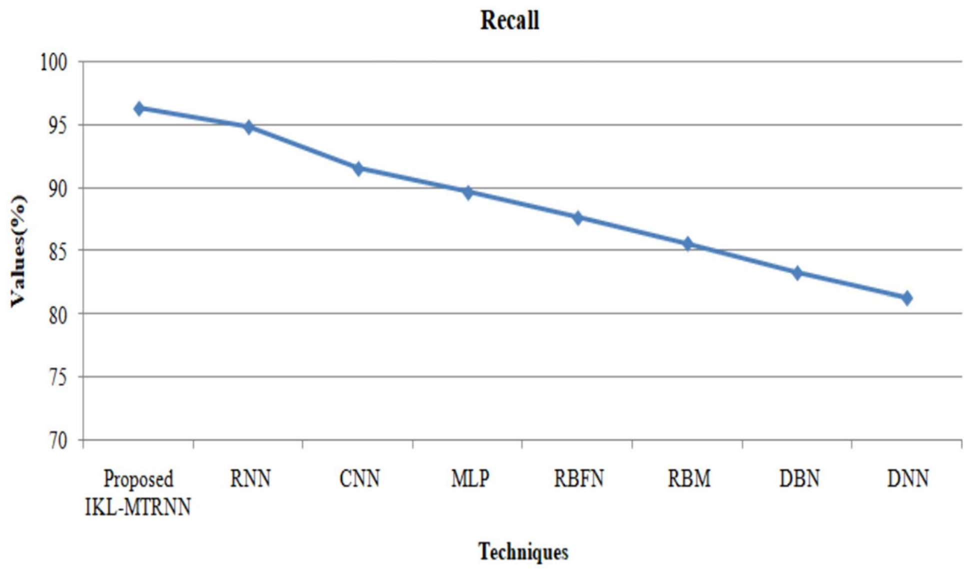


Fig. 10 Performance analysis of algorithms based on recall metric

(8.6%), RBF (85.6%), MLP (89.6%), and DNN (81.3%) algorithms. Thus, it is proved that in the proposed algorithm, a greater number of TCs are prioritized correctly than by any other algorithms.

Table 8 exhibits the experimental outcomes attained for the metrics like accuracy as well as f-measure. The accuracy is analyzed to estimate how accurately the TCP provides higher priority to the mandatory test. The accuracy obtained by RNN is 2.36% higher than the DBN, which is the highest accuracy gain amongst the prevailing algorithms. Nevertheless, the proposed IMTRNN gained an accuracy, which is 10% higher than the RNN algorithm. As the test case priority is identified with improved accuracy using the proposed method, the test cases are well-executed and the presence of bugs can be predicted earlier during the testing period. Hence, the accuracy of the proposed method for prioritization of test cases is assessed to verify the performance. Here, the proposed algorithm attained the f-measure value of 96.6%, which is better than the other prevailing methodologies. The highest f-measure achieved by the proposed method exhibits how appropriately the model generates the prioritized test cases. Thus, it is evident that more accurate outcomes were obtained by the proposed IMTRNN for TCP than the prevailing methodologies.

Table 8 Experimental analysis of the proposed and prevailing schemes based on accuracy and F-Measure

Techniques	Accuracy	F-Measure
Proposed IMTRNN	96.89458	96.65325
RNN	94.84755	94.65458
CNN	92.65898	92.65898
MLP	90.32565	89.32565
RBFN	88.64578	87.65325
RBM	86.3257	85.65898
DBN	84.87457	83.26574
DNN	82.87457	81.25447

The pictorial representation of the specificity evaluation of the proposed IMTRNN in comparison with conventional TCP algorithms is exhibited in Fig. 11. By analyzing the specificity, it is verified that the TCs to be executed at last are given lesser priority. The proposed model is evaluated in terms of specificity to examine the model's capability of recognizing the non-prioritized or low-prioritized test cases. This in turn helps the model to sequence the test cases correctly at the respective priority level and enhance the functionality of the source code. Here, the proposed IMTRNN obtained a specificity of 97.88%, which is higher than the prevailing RNN (95.6%), DBN (83.2%), MLP(90.6%), RBFN (87.6%), RBF (85.9%), and DNN

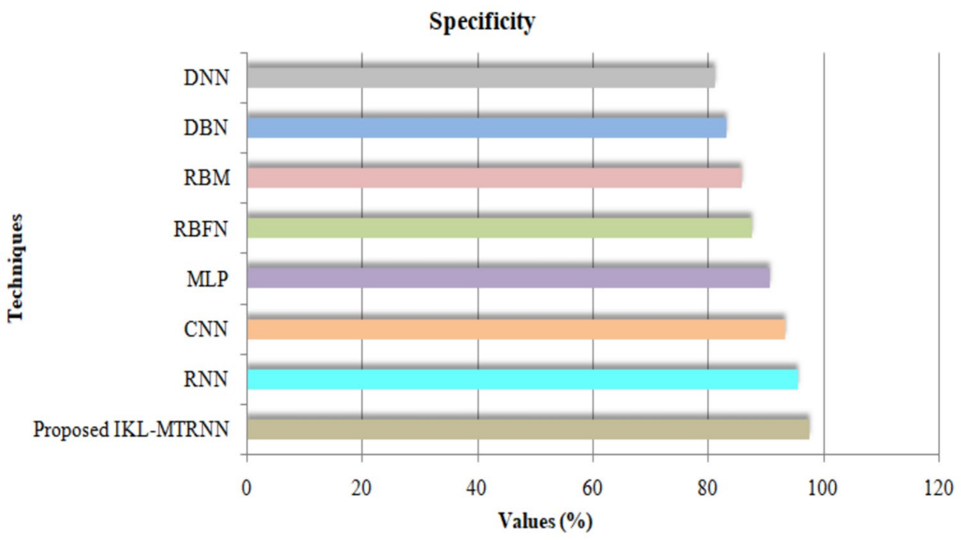


Fig. 11 Comparative analysis based on specificity

(81.12%) algorithms. Thus, it is verified that lesser priority was provided by the proposed IMTRNN to the TCs that are executed at last.

From Fig. 12, it is clear that the bug detection capabilities of the proposed IMTRNN are 99.2356%, which is higher when compared to existing RNN, CNN, MLP, and RBFN techniques that had the lower bug detection capabilities of 95.2659%, 91.5874%, 87.1245% and 83.2568%, respectively. Here, the aim of bug detection capacity is to enhance software reliability by identifying and addressing flaws effectively, thereby improving overall system performance and user experience. The proposed IMTRNN had Interpolated Multiple time scale requirements that enhanced the model's performance. But, the existing systems had lower bug detection coverage, which hindered the effectiveness of the work.

The Code coverage of the proposed IMTRNN and the existing RNN, CNN, MLP, and RBFN are shown in Fig. 13. As shown in the graph, the proposed IMTRNN had higher code coverage of 96.2348% while the existing systems had 92.6547%, 85.2486%, 81.3245%, and 75.6247% respectively. The inclusion of IMT in the RNN

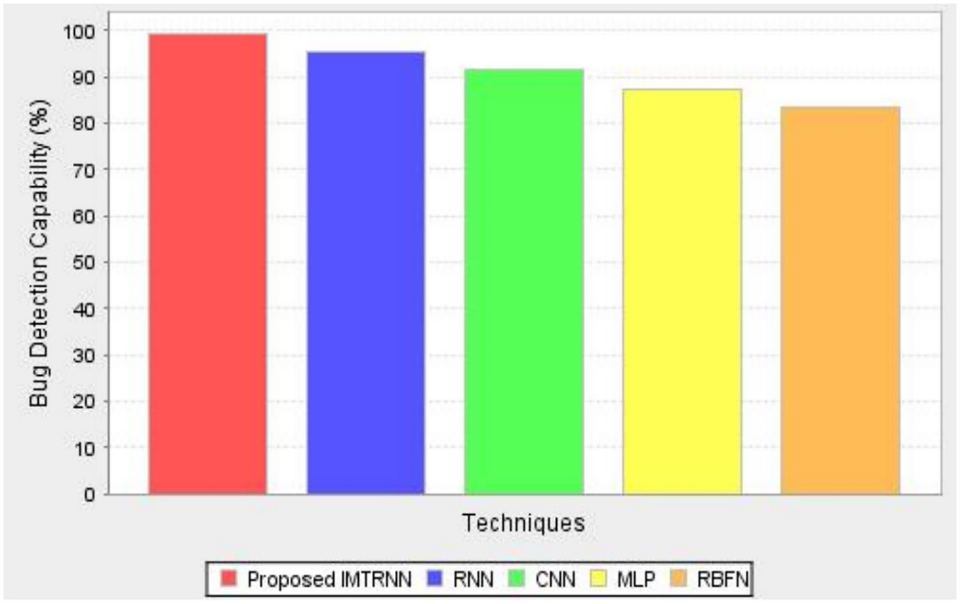
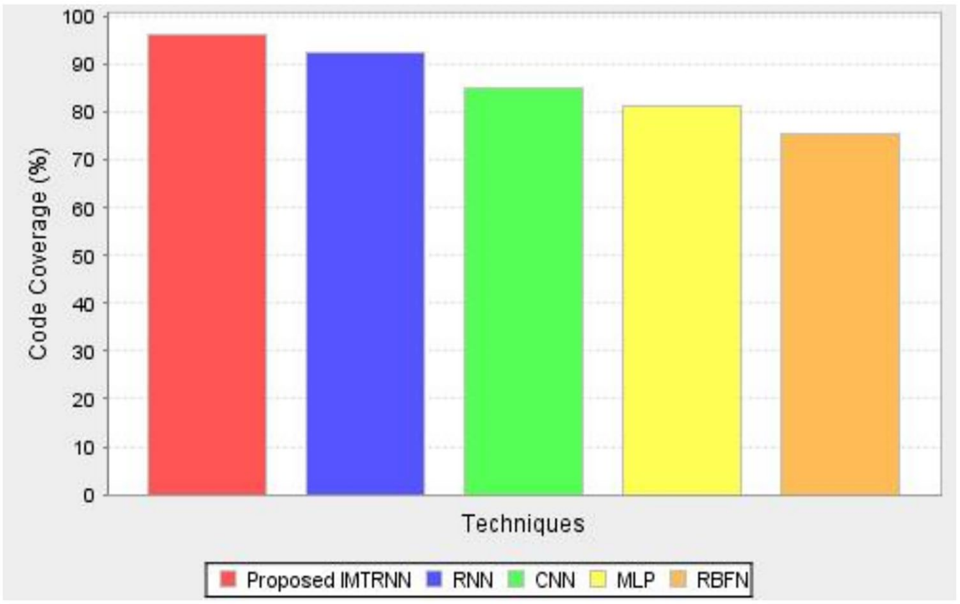


Fig. 12 Bug detection capability

Fig. 13 Code coverage

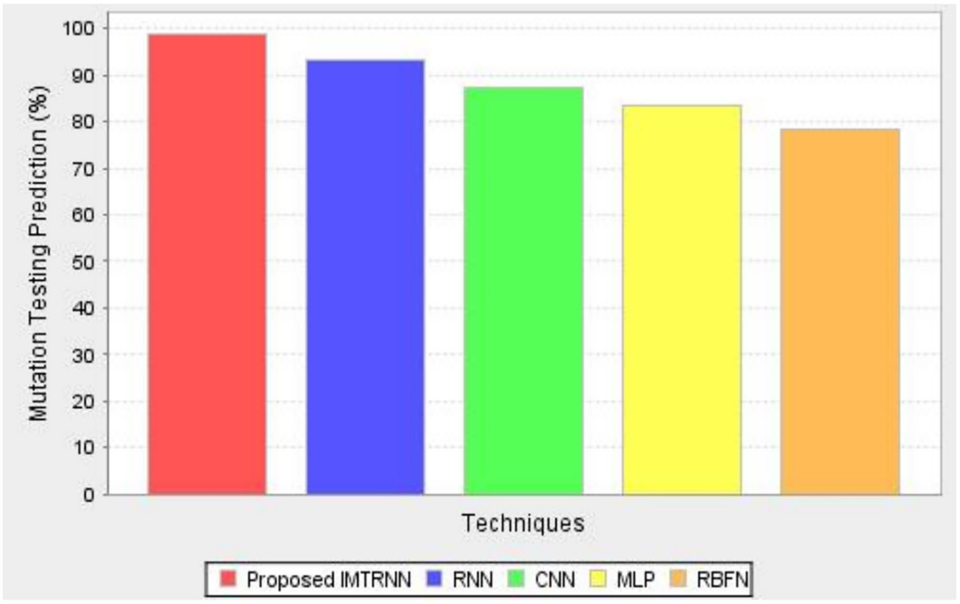


technique enhanced the neural network that enhanced the model's effective code coverage. Thus, when compared to the proposed work, the existing works had less code coverage efficiency, hindering the entire work. Also, in the worst-case scenario, the proposed IMTRNN achieved a code coverage of 90%, and across multiple executions, the average code coverage achieved by IMTRNN is 93.5%. Finally, in the best-case scenario, the proposed IMTRNN, which is shown in Graph 12, achieves a code coverage of 96.2348%. When compared to the proposed IMTRNN, the traditional techniques have limited code coverage in all worst, average, and best-case scenarios, thus degrading the performance of traditional techniques.

Figure 14 describes the mutation testing predictions of the proposed and existing techniques. The proposed IMTRNN had 98.7812% while the existing RNN, CNN, MLP, and RBFN techniques had lower mutation testing predictions of 93.1256%, 87.4578%, 83.3694%, and 78.1549%, respectively. Thus, from the graph, it is clear that the proposed IMTRNN had a higher prediction capacity than existing techniques due to the enhancement of the model using IMT techniques. Thus, the proposed work surpassed the existing works in mutation testing predictions.

Here, mutation testing serves as a critical evaluation metric for assessing the effectiveness of test case prioritization. By introducing mutations to the source code and measuring

Fig. 14 Mutation testing prediction



the ability of prioritized test cases to detect these changes, the framework evaluates the quality and reliability of the prioritization algorithm. High mutation detection rates indicate effective targeting of critical code areas for enhancing software reliability and fault detection capabilities. This underscores the importance of mutation testing in validating the proposed framework's efficacy in software testing.

6 Conclusion

Based on ML, a novel TCG and TCP have been proposed in this study. Here, based on the Ent-LSOA approach, the optimal TCs are generated; additionally, by utilizing the PCC-GDA methodology, the assessment factors being extracted are mitigated. Eventually, the TCs are prioritized regarding the selected factors, historical TCs, along with optimal TCs. Here, a novel IMTRNN classifier algorithm has been proposed for the TCP. The proposed work thus starts with UML diagram generation from historical project source code, followed by feature extraction and optimal TCG. Additionally, factors are extracted and reduced using PCC-GDA. The prioritization phase achieves high accuracy and recall, surpassing existing techniques with 96.89% and 96.92% respectively. This research contributes to advancing software testing methodologies, enhancing fault detection, ultimately improving software reliability, and reducing time complexity.

6.1 Future Recommendation

The proposed work has limitations as it extracted only a limited number of factors and features. Additionally, the classification process lacked parameter fine-tuning. These constraints may impact the comprehensiveness and optimization of the proposed framework for TCG and TCP. Hence, in the future, to minimize the complexity of the proposed TCP, the model will be ameliorated with an enhanced optimal feature selection process and parameter fine-tuning methodology.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10836-024-06121-x.

Funding The authors received no specific funding for this study.

Data Availability Dataset for Test case Generation and prioritization https://zenodo.org/records/268466; https://drive.google.com/drive/folders/1ifx1QOoyouqV99w_un_e8w9du3_Q4-v9?usp=drive_link

Declarations

Conflicts of Interest An authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Ahmed M, Nasser AB, Zamli KZ (2022) Construction of Prioritized T-Way Test Suite Using Bi-Objective Dragonfly Algorithm. IEEE Access 10:71683–71698
2. Bagherzadeh M, Kahani N, Briand L (2022) Reinforcement Learning for Test Case Prioritization. IEEE Trans Software Eng 48(8):2836–2856
3. Barisal SK, Chauhan SP, Dutta A, Godboley S, Sahoo B, Mohapatra DP (2022) BOOMPizer: Minimization and prioritization of CONCOLIC based boosted MC/DC test cases. J King Saud Univ - Computer Inf Sci. pp 1–20
4. Bajaj A, Sangwan OP (2021) Discrete and combinatorial gravitational search algorithms for test case prioritization and minimization. Int J Inf Technol (Singapore) 13(2):817–823
5. Bajaj A, Sangwan OP (2021) Discrete cuckoo search algorithms for test case prioritization. Appl Soft Comput 11:1–18
6. Bajaj A, Sangwan OP (2021) Tri-level regression testing using nature-inspired algorithms. Innovations Syst Softw Eng 17(1):1–16
7. Birchler C, Khatiri S, Derakhshanfar P, Panichella S, Panichella A (2023) Single and multi-objective test cases prioritization for self-driving cars in virtual environments. ACM T Soft Eng Meth 32(2):1–30
8. Dai X, Gong W, Gu Q (2021) Automated test case generation based on differential evolution with node branch archive. Comput Ind Eng 156:1–13
9. Dandan H (2020) A research on automated software test case generation based on control flow. Procedings - 2020 International Conference on E-Commerce and Internet Technology. ECIT, pp 204–207
10. Gokilavani N, Bharathi B (2021) Test case prioritization to examine software for fault detection using PCA extraction and K-means clustering with ranking. Soft Comput 25(7):5163–5172
11. Han J, Li Z, Guo J, Zhao R (2020) Convergence based Evaluation Strategies for Learning Agent of Hyper-heuristic Framework for Test Case Prioritization. In proceedings of 2020 IEEE 20th Proc. International Conference on Software Quality, Reliability, and Security, QRS 2020:394–405
12. Jaffari A, Yoo CJ, Lee J (2020) Automatic test data generation using the activity diagram and search-based technique. Applied Sciences (Switzerland) 10(10):9–13
13. Jahan H, Feng Z, Mahmud SH (2020) Risk-Based Test Case Prioritization by Correlating System Methods and Their Associated Risks. Arab J Sci Eng 45(8):6125–6138
14. Khari M (2019) Empirical Evaluation of Automated Test Suite Generation and Optimization. Arab J Sci Eng 45(4):2407–2423
15. Khari M, Sinha A, Verdu E, Crespo RG (2020) Performance analysis of six meta-heuristic algorithms over automated test suite generation for path coverage-based optimization. Soft Comput 24(12):9143–9160
16. Li Y, Tao J, Wotawa F (2020) Ontology-based test generation for automated and autonomous driving functions. Inf Softw Technol 117:1–43
17. Minhas NM, Masood S, Petersen K, Nadeem A (2020) A systematic mapping of test case generation techniques using UML interaction diagrams. J Softw: Evol Process 32(6):1–21
18. Panda N, Mohapatra DP (2021) Test scenario prioritization from user requirements for web-based software. Int J Syst Assur Eng Manag 12(3):361–376
19. Paiva AC, Restivo A, Almeida S (2020) Test case generation based on mutations over user execution traces. Software Qual J 28(3):1173–1186

20. Rocha M, Simao A, Sousa T (2021) Model-based test case generation from UML sequence diagrams using extended finite state machines. Softw Qual J 29(3):597–627
21. Sahin O, Akay B, Karaboga D (2021) Archive-based multi-criteria Artificial Bee Colony algorithm for whole test suite generation. JESTECH 24(3):806–817
22. Sahoo RR, Ray M (2020) PSO based test case generation for critical path using improved combined fitness function. J King Saud Univ - Comput Inf Sci 32(4):479–490
23. Sankar SS, Chandra VC (2020). An Ant colony optimization algorithm based automated generation of software test cases. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, 12145, pp 231–239
24. Shah SA, Bukhari SS, Humayun M, Jhanjhi NZ, Abbas SF (2019) Test case generation using unified modeling language. In proceedings of 2019 International Conference on Computer and Information Sciences. ICCIS, pp 1–6
25. Shin KW, Lim DJ (2020) Model-based test case prioritization using an alternating variable method for regression testing of a UML-based model. Applied Sciences (Switzerland) 10(21):1–23
26. Singhal S, Jatana N, Subahi AF, Gupta C, Khalaf OI, Alotaibi Y (2022) Fault coverage-based test case prioritization and selection using african buffalo optimization. CMC 74(3):6755–6774
27. Su W, Li Z, Wang Z, Yang D (2020) A meta-heuristic test case prioritization method based on hybrid model. In proceedings of 2020 International Conference on Computer Engineering and Application. ICCEA, pp 430–435
28. Yaraghi AS, Bagherzadeh M, Kahani N, Briand LC (2023) Scalable and accurate test case prioritization in continuous integration contexts. IEEE Trans Software Eng 49:1615–1639
29. Zamani S, Hemmati H (2020) A cost-effective approach for hyper-parameter tuning in search-based test case generation. In proceedings of 2020 IEEE International Conference on Software Maintenance and Evolution. ICSME, pp 418–429
30. Zhou ZQ, Liu C, Chen TY, Tse TH, Susilo W (2021) Beating random test case prioritization. IEEE Trans Reliab 70(2):654–675

A. Tamizharasi is Assistant Professor in Department of Computer Science and Engineering, R.M.D. Engineering College, Chennai. She received a Bachelor's degree in Computer Science and Engineering from Pavendhar Bharathidasan College of Engineering and technology and M.E in Systems Engineering and Operation Research from College of Engineering, Anna University, Chennai. Currently pursuing PhD at Anna University, Chennai. Her areas of speciality and interest are Software Testing, Machine Learning, Artificial Intelligence, Data Science and Wireless Sensor Networks.

P. Ezhumalai is a Professor and Head at Department of Computer Science and Engineering, R.M.D. Engineering College, Chennai. He got his Master's from Jawaharlal Nehru Technological University, Hyderabad and PhD degree from Anna University, Chennai. He has 25 years of working experience in teaching profession and 10 years of research experience. His research focuses on Cloud computing, Multicore Architecture, Artificial Intelligence and Machine Learning. Patent has been granted for his research "VIRTUAL JEWELLERY" He has published 65 papers in various International Journals and Conferences. He has got National Citizenship Gold Medal Award for excellence from Global Economic Progress and Research Association, New Delhi.