Soft Computing (2022) 26:6953–6968
https://doi.org/10.1007/s00500-022-07174-w

FOUNDATIONS

Analytic hierarchy process-based regression test case prioritization technique enhancing the fault detection rate

Soumen Nayak[1] · Chiranjeev Kumar[1] · Sachin Tripathi[1]

Accepted: 8 May 2022 / Published online: 11 June 2022

Abstract
Regression testing is a testing method conducted to ensure that improvements do not affect the software's current behavior. Test cases play a significant role in software testing activities since it detects faults in the software. The selection of test cases for execution based on the priority is a considerable decision-making step since testing needs to be carried out with limited computing resources like cost, effort, and time. Prioritized selection of test cases involves considering specific test case parameters or criteria; prioritizing test cases can be conceived as a multi-criteria decision-making problem. In this paper, using the Analytic Hierarchy Process (AHP) method, we suggest an approach for prioritizing the selection of the test cases. AHP provides a rational framework for structuring a decision-making problem. It is used to determine the priority of a set of criteria and to calculate the consistency ratio of those criteria using pair-wise comparisons of various measures. Using a new priority regression test algorithm proposed in this study, regression test prioritization is transformed into a multi-criteria decision-making problem. The suggested strategy has a higher Average Percentage of Faults Detected (APFD) value when many determinants are taken into account. The experimental results indicate that our proposed prioritization approach would improve the probability that faults will be detected early and enhance performance compared to other prioritization strategies.

Keywords Software testing · Test case prioritization · Regression testing · AHP technique · APFD metric

1 Introduction

Over a decade, evaluation and prioritization of test cases in regression testing have received a tremendous focus from academics, the information technology sector, and researchers. With computing resource constraints like budget, time, effort, etc., there is a need to decide which test cases to design, what test cases need to be selected, and how they will be executed in a particular order. In practice, setting the prioritized test cases requires evaluating different testing criteria, making the test case prioritization problem a multi-criteria decision-making problem (Tahvili and Bohlin, 2016). As the decision criteria grow in number, there is a need to make a systematic decision that leads to an efficient solution. Prof. Thomas L. Saaty devised the AHP which is a multi-criteria decision-making process (Saaty, 1987). It provides a rational framework for structuring a decision-making problem. It is used to find the priority and derives the consistency ratio of particular criteria by pair-wise comparison among various measures. AHP has been considered a popular decision-making tool that can be implemented to solve a wide variety of decision-making problems. Several researches have discussed methods for extracting a subset of test instances from a larger set of test cases. But there always is a place for further exploration and getting a result that enhances the test cases' rate of fault detection.

To address cost reduction in regression testing, several approaches have been proposed and developed (Rothermel et al. 2001): Regression Test Selection (RTS), Regression Test Minimization (RTM), and Regression Test Prioritization (RTP). When it comes to RTS and RTM, there comes a point where a few test cases must be permanently

✉ Soumen Nayak
soumen.nayak@gmail.com

Chiranjeev Kumar
chiranjeev@iitism.ac.in

Sachin Tripathi
sachin2781@iitism.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology (ISM), Dhanbad 826004, India

removed from the test suite (Rotherrmel et al. 1998; Rotherrmel et al. 1997). Because RTP strategies do not eliminate any test cases, the disadvantages of the two methods can be avoided. However, prioritization strategies can be used in conjunction with RTS or minimization techniques in circumstances where test cases can be discarded. Furthermore, priority strategies increase the possibility that if a regression testing activity is interrupted unexpectedly, the time spent testing will be more productive than if test cases are not prioritized. Regression testing is aided by RTP strategies, which rank the test cases. During regression testing sessions, the highest priority test cases run first, followed by the lower priority test cases, based on specified testing criteria. Test cases can be scheduled in a way that achieves faster code or statement coverage, or based on the frequency with which features are used, or by components that have a history of failing.

In the literature, Nayak et al. (2017) proposed a RTP technique where they have considered four factors and found an optimal ordering. The authors have not mentioned any relationship for any factor at the time of obtaining the results. In other words, there might be some factors that might have a severe impact on the outcome than the rest. In practice, this multi-criteria software testing problem using software metrics can be solved. Still, it isn't easy to comprehend and compute due to the usage of several multiplicative factors. It is because the values of the metrics are volatile. These values depend on certain factors like the number of faults detected by a test case, total time taken to detect defects, risks associated with the flaws, etc.

On the other hand, these factors depend on different criteria like test case organization in the test suite, number of test cases, number of faults available in the system, the clock speed of the processor, etc. In this manuscript, to avoid the dominance of any factor in deciding the result mentioned in Nayak et al. (2017), using the AHP technique, we proposed a method for ranking test cases. Before acquiring test cases, the results are normalized to eliminate parameter dominance and provide an accurate solution. With the use of average percentage of faults detected (APFD) metric values, the outcome is compared to various prioritization strategies such as no order prioritization (NOP), random order prioritization (ROP), reverse order prioritization (REOP), and previous works (Kavitha and Sureshkumar 2010; Tyagi et al. 2015; Nayak et al. 2016).

The goal of this work is to fill in the gaps in the current research. The following are the research questions on which this research is focused:

RQ1 Is it possible to improve the fault detection rate of test cases in test suites utilizing RTP approaches that use AHP?

RQ2 How does the proposed algorithm solve the test case prioritization problem by normalizing the obtained result values?

RQ3 In terms of fault detection rate, how is the suggested algorithm superior to the different existing prioritization techniques?

The three issues addressed in this study help to reduce the computing cost and improve the accuracy of regression testing. During the regression test, we also pick the algorithm to use for prioritization in order to get the best fault coverage with the least amount of time.

The RTP is transformed into an MCDM problem using a new priority regression test algorithm proposed in this study. Taking into account many determinants, the suggested strategy has a higher APFD value. The following points can be applied to the existing literature in the manuscript:

(1) A new algorithm for test case prioritization at the maintenance level,
(2) A framework that requires several inputs to have a higher APFD value still,
(3) By normalizing the outcomes of multiple factors, a better priority algorithm is constructed in comparison to traditional prioritization algorithms, and
(4) Test cases highly reliant on test suites are used to assess the fault-finding process.

The remainder of the paper is arranged in the following manner: Sect. 2 presents the associated prior works on RTP. Section 3 discusses the problem statement, and concepts and fundamentals required to understand the work in brief. In Sect. 4, the proposed methodology is presented. Section 5 deals with experimental evaluations. Section 6 deals with results and discussions. At last, Sect. 7 offers the conclusion, and future enhancements are discussed.

2 Related work

Several studies have emphasized different measures and methods for prioritizing regression test cases to optimize regression test effectiveness for several years. Prioritization methods order the execution sequence of the test cases so that meeting some performance goals is increased. The ability for error detection is improved in the early stages of the development cycle. It encourages sufficient pro-vision for quicker reviews on the SUT and gives an overview of the consistency targets that have not been achieved. It can allow software engineers to test until the software is released under a restriction time limit. Yoo and Harman (2012) surveyed different techniques for prioritization, where they have studied open issues and categorized the existing approach into several groups.

The test cases were arranged by Avritzer and Weyuker (1997) by modeling them using Markov chains, which increased the likelihood of identifying flaws early, but they never used the term "prioritization". Rothermel et al. (2001) also suggested nine prioritization methods, and by analyzing the Siemens suite, empirically clarified the trade-off of each strategy. Qasim et al. (2021) compiled the work in the domain of TCP from 2010 to 2020, focusing mainly on the nature-inspired algorithms.

Hettiarachchi et al. (2016) used the status change criteria, complexity, reliability, and scale of the program specifications as risk measures. They used a fuzzy expert model to quantify the risks of the requirements. The new method will spot faults early and detect more defects sooner in the high-risk system components. Jahan et al. (2020) suggest a semi-automatic, risk-driven approach to prioritizing test cases based on experience about program alteration and invocation methods. The experimental findings indicate that by detecting defects early overall, the proposed solution increases the test performance.

Wei et al. (2020) have a Hidden Markov Model (HMM)-based test case prioritization (TCP) technique to detect faults as early as possible and lower the alteration costs. The proposed approach has four significant portions like the UML sequence diagram is converted to HMM, the fault urgency is estimated according to the priority and probability of the fault, the fault severity is assessed by considering the weight factors of the states available in the HMM, and the priority is generated from fault severity and urgency. They have performed this TCP operation on the flight control system in unmanned aerial vehicles (UAV). The obtained result is promising in detecting the faults earlier. Emam and Miller (2015) suggest an expanded form of the digraph using a novel methodology based on reinforcement-learning (RL) and HMM. Test cases will be given priority in this model for regression testing goals. Based on RL concepts applied to an application's digraph model, they describe a method for initializing and training an HMM. The model selects test cases using forwarding probabilities, which is a novel method for TCP. The experiment's findings suggest that the proposed technique can uncover flaws early.

Mohanty et al. (2020) proposed a RTM technique using the Ant Colony Optimization algorithm (ACO-min). The algorithm seeks a representative set of test cases that meet all the criteria and take the least time to execute. The representative set also has the same ability to detect faults as the original test suite. Bajaj and Sangwan (2019) systematically reviewed the literature on TCP using a genetic algorithm. This analysis aims to analyze and identify the current state of use of the genetic algorithm to prioritize the test cases. Arrieta et al. (2019) suggest a search-based methodology that aims to automate the test cycle of CPS product lines cost-effectively by prioritizing the test cases performed at different test grades of similar products. The target test suite seeks to reduce the discovery time for defects, the simulation period, and the resources needed to address practical and non-functional needs. The study shows that the algorithms for the local search were more efficient than global search algorithms.

Khatibsyarbini et al. (2019) proposed a new TCP method based on the firefly algorithm and a similarity distance model. This method is applied to three benchmark programs, resulting in the conclusion that this algorithm provides a better APFD value. Panwar et al. (2018) suggested a test case prioritization approach based on a hybrid of the Cuckoo-ACO algorithm. The results obtained are promising in terms of defect detection. The hill-climbing algorithm, greedy algorithm, additional greedy algorithm, genetic algorithm, and a two-optimal greedy algorithm are among the TCP meta-heuristics described by Li et al. (2007). The greedy algorithms may produce a sub-optimal result that can be resolved by meta-heuristic or evolutionary search techniques. The result of their analytical investigation, which included six programs, provides light on the search space in regression research, highlighting its multi-modal existence. Despite the fact that the evolutionary algorithm performs well, greedy tactics win the race due to the multi-modal nature of the environment. The discrete cuckoo search method is proposed by Bajaj and Sangwan (2021) for test case prioritization. The prioritization problem is concerned with the order in which the test cases should be run. To convert real numbers into permutation sequences, a new adaption technique based on asexual genetic reproduction is developed. The effectiveness of the cuckoo search algorithm is further enhanced by the mutation operator of the genetic algorithm, which balances the trade-off between exploration and exploitation.

Klindee and Prompoon (2015) have done a case study on TCP using AHP. They have not implemented any real-world project and hence have not considered the different factors that impact TCP. Ahmad and Pirzada (2014) have used AHP for prioritizing the functional strategies in auto parts manufacturing. The analysis will help policymakers understand this sector's market practices and devise strategies to improve their efficiency. Afzal and Sadim (2018) have used the AHP technique for the selection of software requirements. They consider Institute Examination Systems' ten technical criteria and measure them on a cost-based basis.

Srivastava (2008) has proposed a software metric that can enhance TCP efficiency. They have taken only one metric for constructing a TCP technique. Luo et al. (2018)

have done an empirical study by contrasting different TCP techniques. They have applied this to both real-world and mutation faults. The result shows that the best approach performed on a set of mutants may not work efficiently when applied to actual defects. Noor and Hemmati (2017) suggested a technique using a logistic regression model to forecast faulty test cases in the latest release based on a series of consistency check metrics (Similarity-based metric). The regression model ranks are more accurate in the current version, prioritizing fault disclosing test cases.

To arrange test cases, Cho et al. (2016) carefully examine the history of failure for each test case. Correlation data from test cases gathered from previous test results is used to reorder test cases in the next step. On two open-source Apache software projects, an observational analysis is performed (i.e., Tomcat and Camel). This study shows this technique provides failure information more efficiently than other methods. Shin et al. (2018) suggest a novel methodology for prioritizing research cases, incorporating mutation-based approaches with diversity-aware approaches. The diversity-aware mutation-dependent method is based on mutant differentiation, which seeks to differentiate one mutant's behavior from the other instead of the original system. The study reveals that all the flaws analyzed are without a single dominant technique. To this end, the explanation why each of the mutation-based prioritization parameters performs poorly is addressed using a statistical model called Mutant Distinguishment Chart that displays the distribution of test cases for mutant distinguishing and killing. For DNN-based classifiers, a test case prioritization strategy is utilized by Yan et al. (2022), which gives high priority to situations that might lead to incorrect classifications. The test instances are ranked in order of importance based on the pattern identified from the neuron outputs. The priority of an input is determined by the difference between the pattern gained from the training set and the neurons matched from that input.

Several TCP approaches are structured to take factors into account. Kavitha and Sureshkumar (2010) have suggested a prioritization strategy that considers two components: the rate of identification of errors and the impact of faults. They have taken two industrial projects and have proven that the prioritized ones are better than non-prioritized ones as far as performance is considered. Tyagi and Malhotra (2015) took three things into account and showed better results than Kavitha et al.'s work. Jeffrey and Gupta (2005) suggested using appropriate slices for prioritization. Korel et al. (2007) proposed a model-based prioritization strategy that would consider knowledge about the device configuration and its behavior against TCP. Maheswari et al. (2013) suggested the use of a Hamming distance for a TCP technique. The fault revealed will be binary in shape and match to strings of equal length. Calculate the unknown locations by combining the respective symbols. The obtained results outperform other methods. Nayak et al. (2017) have proposed a TCP technique based on a weighted sum of four factors. The higher APFD value is obtained, and this paper opened an insight that by adding factors, the rate of fault detection can be improved. Nayak et al. (2016) have proposed a novel technique considering two factors for prioritizing the test cases. The result obtained is better than the rest.

For regression testing, Chi et al. (2020) recommend ratio-based TCP. They claim in this study that the dynamic function call can better guide TCP. They have presented a novel method dubbed the additional greedy method call sequence (AGC), which takes use of dynamic relationship-based coverage and applies it to eight real-world Java applications. The result is compared to 22 TCP approaches in terms of bug discovery capability. The solution suggested provides a better outcome than the others. Dash and Acharya (2022) performed a thorough literature assessment of TCP research undertaken between the years 1999 and 2020. Huang et al. (2022) used the notion of data fusion to present data-fusion-driven DTCP (DDTCP), a novel family of DTCP algorithms that tries to employ different information granularities for selecting test cases based on dissimilarity. Shrivathsan et al. (2019) proposed two TCP approaches based on fuzzy-based clustering techniques using the resemblance coefficient and the metric of superiority. There is an improvement in the likelihood of choosing more appropriate test cases when analyzing the data from the Software-artifact Infrastructure Repository (SIR).

3 Background

In this section, a glimpse of the idea of the AHP that is the basis for TCP in regression testing in this manuscript, the motivation behind this study, research questions that are related to the survey of which we are focusing and the concepts and definitions of some of the terms used in this study for a better understanding of the paper.

3.1 Analytic hierarchy process

The AHP is a measurement theory through pair-wise comparisons. It focuses on expert judgments to deduce priority scales. In relative terms, it is these scales that evaluate intangibles. The contrasts are made using a scale of complete reviews reflecting a given feature and how much one variable exceeds another. The decisions can be inconsistent. The AHP's problem is how to assess inconsistencies and strengthen findings wherever feasible to achieve greater consistency. The derived priority scales are synthesized by multiplying them by their parent nodes' priority and adding them.

To decompose the decision or judgments into the following phases are required to decide a systematic way to produce priorities.

(1) Defines the issue or problem and determines what type of knowledge is being sought.
(2) Lay out the decision hierarchy from the highest level, with the purpose of the judgment, down to the lowest level, with the priorities from a specific point of view (which is generally a collection of substitutes).
(3) Create a set of pair-wise comparison matrices. That element at the top level is used to equate the aspects directly below concerning it.
(4) Using the priorities of the comparisons, calculate the expectations at the stage immediately below. Do it for each element. Alternatively, add the weighted values to each component below and gain the absolute or global priority. Start this calculation process until the final requirements of the alternatives are reached at the lowest point.

To make comparisons, a scale of numbers is required to state how much more significant or superior one element is over another element in terms of the criteria or property in respect to which it is measured. The scale is seen in Table 1. The whole number is entered in its respective position, and the reciprocal is placed in the transpose position.

The AHP has various sets of advantages when it comes to its implementation. Some of them can be listed as:

(1) Combining multiple inputs from several persons to a consolidated outcome.
(2) Has a higher probability of finding a desirable output.
(3) Considers complex situations where there are numerous variables or criteria to consider when prioritizing and selecting alternatives or projects.
(4) Takes different intensities of the criterion (not only the extremities) into account before concluding

Designed for multi-criteria When there are important decisions to make, there are always conflicts between criteria. Analytic Hierarchy Process allows users to take all essential measures into account and organize them into a hierarchy.

3.2 Problem statement

The fundamental goal of test prioritization is to organize the test cases according to certain test adequacy criteria. Moreover, officially, the prioritization problem is defined by Rothermel et al. (2001) as such:

Definition: TCP_Problem_Statement.
Given: A set of test cases, T, the permutation set of T, like PT, along with f as a function from PT to the collection of real numbers.
Problem: Find $T' \in PT$ such that $((T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.

According to the previous definition, PT denotes all potential arrangements (orderings) in a set of T. In addition, for each order, the function 'f' returns a value known as "reward value". From the above definition, the goals of prioritization like increasing the rate of fault detection, code coverage in the system under test (SUT) at a faster rate, enhancing SUT dependability at a speedier pace, detecting high-risk faults earlier in the testing process, and

Table 1 Saaty rating scale

Intensity of importance	Definition	Explanation
1	Equal amount of importance	Equal contribution of two activities to the objective
2	Weak importance	
3	Moderate amount of importance	Slight favor of one activity based on experience and judgement over another
4	Moderate plus	
5	Strong importance	Strong favor of one activity based on experience and judgement over another
6	Strong plus	
7	Demonstrated importance	An action is favored very strongly over another; its superiority illustrated in reality
8	Very very strong	
9	Extreme importance	Evidence promoting one operation above another is in the highest possible order of affirmation

Table 2 Fault matrix of project 1

Test case/faults	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
F1								×	×	
F2		×	×		×					
F3				×		×				×
F4		×	×							
F5								×		
F6								×	×	
F7				×	×		×			
F8	×					×				
F9				×	×	×				×
F10	×							×		

Table 3 Number of faults and execution time for each test case

Test Cases	No. of faults detected	Time (ms)
T1	2	9
T2	2	8
T3	2	14
T4	3	9
T5	2	12
T6	3	14
T7	1	11
T8	4	10
T9	2	10
T10	2	13

maximize the probability of disclosing defects created due to changes in the code earlier, etc. can be stated quantitatively using the representation of 'f.'

The prioritization problem may become unmanageable or undecidable depending on the function 'f.' The solution to the TCP problem will yield a solution to the traveling salesman problem. Hence Analytic Hierarchy Process is implemented to solve Combinatorial Optimization Problems like the TCP problems.

3.3 Concepts and definitions

There are several factors used in the literature (Nayak et al. 2017) that have been used in this paper for implementing AHP are as follows:

(1) Rate of fault detection (RFT)—It is known as the average number of defects found per minute by the test case. The RFT of the T_k test case can be determined using the T_k defect count and the T_k time criterion to identify defects for each test case of the test suite. The equation is the following:

$$\mathrm{RFT}_k = \frac{\mathrm{NF}_k}{\mathrm{TT}_k} \times 10 \tag{1}$$

where, NF_k is the count of faults uncovered by T_k and TT_k depicts requirement of time by test case T_k.

(2) Number of faults detected (NFD)—We have suggested this element and considered the prioritization of the test case as it will improve the RFT. The T_k NFD can be defined as the number of faults (NF_k) identified in the T_k test case. Mathematically, this may be expressed as

$$\mathrm{NFD}_k = \mathrm{NF}_k \tag{2}$$

(3) Test case ability of risk detection (TARD)—TARD of the test case T_k is defined as the capacity of the test case to determine the magnitude of the faults per time. Efficiency and efficacy can be enhanced by taking into account test cases that show a higher proportion of significant defects.

$$\mathrm{TARD}_k = \frac{(S_k \times \mathrm{NF}_k)}{\mathrm{TT}_k} \tag{3}$$

(4) Test case effectiveness (TCE)—This indicates that a series of test cases is sufficiently successful in the event of discovered defects. This can be defined as the percentage of the number of faults observed in the test cases and the overall number faults.

$$\mathrm{TCE}_k = \frac{\mathrm{NF}_k}{\mathrm{TNF}} \times 100 \tag{4}$$

There are several standard techniques used in this paper in the comparative study section that is explained here as follows (Rothermel et al. 2001):

NOP: No prioritization technique has been used in this approach. The test cases in the test suite remain untreated and unchanged throughout the testing process. This acts as a control for experimentation.

REOP: In this Prioritization approach, the test cases are arranged in the reversed order in the test suite compared to the initial unchanged test suite. This also acts as another control used for our study.

ROP: Test cases are randomly organized in this case. This acts as an additional control in our studies.

Optimal prioritization: This prioritization approach provides the test suite's optimal ordering in the test case prioritization method. The optimal arrangement never performs worse than the orders that have been calculated in this paper.

4 Proposed work

Test case prioritization in regression testing is prioritizing the test case and forming an order that comes as close as possible to the optimum ordering solution. This is a crucial phase that the software needs to go through during the maintenance phase. This section uses AHP to get an order that can provide us with a relatively higher APFD value. Since this multi-criteria decision-making can prioritize the test cases in a test suite according to each test case's merit rather than just selecting one test case or ranking them, this approach can be used to find an order that can enhance the fault detection rate.

The proposed steps are as follows:

Steps: After obtaining the rate of fault detection (RFT), the number of faults detected (NFD), the test case ability of risk detection (TARD), and the test case effectiveness (TCE) values as explained in the previous part, the advance analysis is done using the AHP technique. The explanation of the whole process is given in the steps below:

Step 1: From the given values in Table 4, the maximum value in each column can be recorded and it is termed as MAX(Factor).

Step 2: Using the values of Table 4 and MAX(Factor), a normalized matrix P is constructed. The obtained normalized values are termed as NORM(Factor). Using Eq. (5), P can be created as follows:

$$\mathrm{NORM}(\mathrm{Factor})_{\mathrm{TC}x} = \frac{\mathrm{Factor}(\mathrm{TC}_x)}{\mathrm{MAX}(\mathrm{Factor})} \tag{5}$$

The NORM(Factor) values that acquired are depicted in Table 5.

Step 3: The relative significance of the goal to the individual factor is assessed. The main motive here is to choose the best test case that can detect more undetected faults. Accordingly, a significant number is allotted to each factor. These values are allocated according to Saaty (1987). The Saaty scale is shown in Table 6.

For each factor, the significance number is represented as IN(Factor).

Since the test cases present in the test suite are monitored, the favoring of one factor over another is avoided. For ordering the test cases from best to worst, equal importance is given to all the factors. Therefore, the significant number is assigned to be 1 to all the factors, which designates the equal importance of all the parameters for our goal, as given in Table 5. In this case, the significance numbers for the four factors are shown in Table 6.

Step 4: M, an importance matrix is created using the significance numbers allotted to each factor according to Eq. (6). In the matrix M, the values are referred to as IMP($\mathrm{Factor}_{\mathrm{mn}}$). In Table 7, the importance matrix M is depicted.

$$\mathrm{IMP}(\mathrm{Factor}_{mn}) = \frac{\mathrm{IN}(\mathrm{Factor}_m)}{\mathrm{IN}(\mathrm{Factor}_n)} \tag{6}$$

where, Factor_m, Factor_n = RFT, NFD, TARD, and TCE. IMP(Factor_{mn}) = Significance of factor m compared to factor n.

The IMP(Factor_{mn}) for all the factors are computed and after that for each factor in matrix M, a column-wise sum is computed. The value obtained is referred to as SUM (IMP(Factor_{mn})).

Step 5: Using the IMP(Factor_{mn}) and SUM(IMP(Factor_{mn}))values, a weight matrix W is constructed according the Eq. (7), where $W(\mathrm{Factor}_{mn})$ = weight of Factor_m compared to Factor_n.

$$W(\mathrm{Factor}_{mn}) = \frac{\mathrm{IMP}(\mathrm{Factor}_{mn})}{\mathrm{SUM}(\mathrm{IMP}(\mathrm{Factor}_{mn}))} \tag{7}$$

Using Eq. (7), W(Factor) values are then obtained which represents the final weights for each factor, where X = number of factors (i.e. four)

$$W(\mathrm{Factor}) = \frac{\sum W(\mathrm{Factor}_{mn})}{X} \tag{8}$$

In this case, the value of weights obtained for all Factors is 0.25.

Step 6: Using the weights of the factor and the normalized values calculated in matrix P, a rank matrix R is constructed. The value in each cell of matrix R is known as PRIORITIZED_ORDER(Factor) as represented in Eq. (9). In Table 7, the matrix R is shown.

$$\begin{aligned}&\mathrm{PRIORITIZED\_ORDER}(\mathrm{Factor})\\&\quad = W(\mathrm{Factor}) * \mathrm{NORM}(\mathrm{Factor})\end{aligned} \tag{9}$$

Step 7: Finally, the test cases in the test suite are ordered by summation of PRIORITIZED_ORDER(Factor) values that can be computed using Eq. (10)

Table 4 Factors namely RFT, NFD, TARD, TCE values for test cases T1…T10

	RFT	NFD	TARD	TCE
T1	2.22	2	1.33	20
T2	2.5	2	1.5	20
T3	1.43	2	0.86	20
T4	3.33	3	3.33	30
T5	1.66	2	1.33	20
T6	2.14	3	2.14	30
T7	0.91	1	0.36	10
T8	0.40	4	8	40
T9	0.20	2	2.4	20
T10	1.53	2	0.92	20

$$\text{PRIORITIZED\_ORDER}(\text{TC}) = \sum_{\text{Factor}_m=\text{RFT,NFD,TARD,TCE}} \text{PRIORITIZED\_ORDER}(\text{Factor}_m) \tag{10}$$

The final prioritizing orders of the test cases are shown in Table 8 for Project 1.

Similarly, the following steps can be followed for prioritizing the test cases in the test suite for the Project 2. The final ordering of test cases is depicted in Table 14.

The flow chart for the above-mentioned algorithm is represented in the Fig. 1.

5 Experiment and discussion

This section illustrates how an experimental appraisal is used to prove the efficacy of an approach. The data that was chosen to guide the experiments is listed below. Furthermore, based on the experimental assessment values, this section presented a quick discussion.

5.1 Experiment for project 1

The experimental setting and test suite for our empirical study were adapted from existing literature (Nayak et al. 2017). In terms of the project, they did an experiment on a PC with a 3 GHz Intel Pentium 4 processor and 8 GB RAM, and the application is running on Windows XP. The study's project is a VB project completed at CCSQ, Chennai, India's Competency Center for Software Quality. Manual testing and the QTP 9.5 tool are used to thoroughly test the programs. They have injected ten faults those are varying in severity level scales. The prioritized test cases are executed for the faulty programs, and the total number of test cases is used to detect the defects. Using random number generation in 'C,' ten alternative random test cases are generated. Finally, the time it takes for each test case to find errors is recorded. The fault matrix was derived from the current source (Nayak et al. 2017). Table 2 shows the amount of flaws discovered by each test case, whereas Table 3 shows the time required to uncover faults and the number of faults detected by each test instance.

Table 4 is the matrix which has all the attributes for each test case, namely RFT, NFD, TARD, and TCE.

The values are found out using the formulas given above. Table 4 shows the output of each value for each test case.

The maximum value in each attribute is found out. The maximum values are:

$$\text{Max(RFT)} = 3.33 \quad \text{Max(NFD)} = 4$$
$$\text{Max(TARD)} = 8 \quad \text{Max(TCE)} = 40$$

The above values are then used to normalize the above Table which can be taken as a matrix too. To normalize the values the formula used is:

$$\text{NORM(Factor)}_{\text{TCx}} = \frac{\text{Factor}(\text{TC}_x)}{\text{MAX(Factor)}}$$

where TC_X is the collective notation for every test case of the data set.

And Factor are RFT, NFD, TARD, TCE.

The Normalized matrix is compared to the maximum value of each factor when the max value is selected from Table 5.

Table 6 shows the availability of each test case and its corresponding values for the attributes. '1' denotes that the test cases are required. And here, as it is observed, the elements of the matrix are '1'; hence, it says that for the final result, the entire set of ten test cases are to be considered.

Table 7 will hold the calculated ranked values for each factors (RFT, NFD, TARD, TCE) using the formula below and which will be arranged accordingly to find out the ranks of each test case.

$$\text{rank value} = 0.25 \times \text{Normalized value of each attribute}$$

Table 5 Normalized Matrix

	RFT	NFD	TARD	TCE
T1	0.667	0.5	0.166	0.5
T2	0.751	0.5	0.188	0.5
T3	0.429	0.5	0.108	0.5
T4	1	0.75	0.416	0.75
T5	0.498	0.5	0.166	0.5
T6	0.643	0.75	0.268	0.75
T7	0.273	0.25	0.045	0.25
T8	0.120	1	1	1
T9	0.060	0.5	0.3	0.5
T10	0.459	0.5	0.115	0.5

Table 6 Availability matrix

	RFT	NFD	TARD	TCE
T1	1	1	1	1
T2	1	1	1	1
T3	1	1	1	1
T4	1	1	1	1
T5	1	1	1	1
T6	1	1	1	1
T7	1	1	1	1
T8	1	1	1	1
T9	1	1	1	1
T10	1	1	1	1
SUM	10	10	10	10

Table 7 Rank matrix

	RFT	NFD	TARD	TCE
T1	0.167	0.125	0.042	0.125
T2	0.188	0.125	0.047	0.125
T3	0.107	0.125	0.027	0.125
T4	0.25	0.188	0.104	0.188
T5	0.125	0.125	0.042	0.125
T6	0.161	0.188	0.067	0.188
T7	0.068	0.063	0.011	0.063
T8	0.03	0.25	0.25	0.25
T9	0.015	0.125	0.075	0.125
T10	0.115	0.125	0.029	0.125

Table 7 is used to calculate the rank of each test case so that the prioritization can be done based on the rank obtain by each test cases.

Table 8 will hold the ranks for each test case found by summing the values of individual test cases under RFT, NFD, TARD, and TCE.

The final order obtain is: $T8 > T4 > T6 > T2 > T1 > T5 > T10 > T3 > T9 > T7$.

Hence, T8, T4, T6, T2, T1, T5, T10, T3, T9, T7 is the optimal order.

5.2 Experiment for project 2

The experimental test suite was taken from the "Student admission system" as mentioned in the literature (Nayak et al. 2017). The problem specification can be accessed at the website http://www.planet-source-code.com. We have conducted an experiment on a PC with a CPU of 2 GHz Intel Pentium dual-core processor and 2 GB RAM. The program runs on the Windows XP operating system. A VB project is being considered for the investigation. Manual testing and the testing tool QTP 9.2 are used to test the application software. The ten faults are injected into the application by compile-time injections. To find all defects, the prioritized test cases are executed, and the total number of test cases executed on the faulty code is computed. Random number creation in the 'C' language is used to generate the eight different random order test cases. The test cases are run to find all of the defects that have been detected. We have kept track of how long it takes to find each fault. The fault matrix is shown in Table 9.

Table 10 shows the matrix where calculated values of RFT, NFD, TARD and TCE are given. The same will contain the maximum values of each attribute.

Maximum values from Table 10:

$$\text{Max(RFT)} = 10 \qquad \text{Max(NFD)} = 4$$
$$\text{Max(TARD)} = 16.5 \qquad \text{Max(TCE)} = 40$$

The normalized values found in the same way as denoted in Table 11 (called as the Normalized Matrix).

Availability Matrix has all its values like '1' because, to find the optimal solution, all the ten tests and fault cases are to be considered, as shown in Table 12. Else if not, then the values must have been '0'.

Table 13 is the matrix that contains the ranked values of each factor of the respective test cases that are calculated using a formula:

$$\text{rank value} = 0.25 \times \text{ Normalized value of each attribute}$$

The rank of each test case is noted in Table 14 which is the sum of each value for the corresponding attributes.

Order we obtain is: $T3 > T4 > T5 > T8 > T1 > T6 > T2 > T7$.

Hence, T3, T4, T5, T8, T1, T6, T7 is the order that is optimal.

Table 8 Sum of rank of each test case

Test case	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Rank	0.459	0.485	0.384	0.730	0.417	0.604	0.205	0.780	0.340	0.394

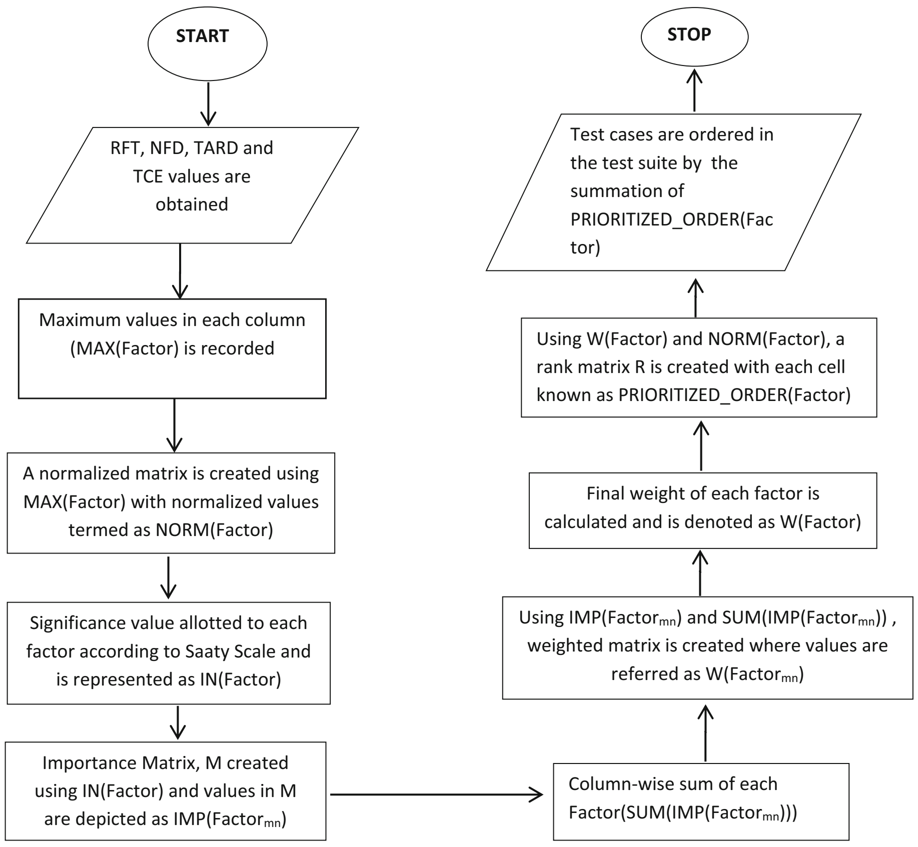
START
RFT, NFD, TARD and TCE values are obtained
Maximum values in each column (MAX(Factor) is recorded
A normalized matrix is created using MAX(Factor) with normalized values termed as NORM(Factor)
Significance value allotted to each factor according to Saaty Scale and is represented as IN(Factor)
Importance Matrix, M created using IN(Factor) and values in M are depicted as IMP(Factor$_{mn}$)
Column-wise sum of each Factor(SUM(IMP(Factor$_{mn}$)))
Using IMP(Factor$_{mn}$) and SUM(IMP(Factor$_{mn}$)), weighted matrix is created where values are referred as W(Factor$_{mn}$)
Final weight of each factor is calculated and is denoted as W(Factor)
Using W(Factor) and NORM(Factor), a rank matrix R is created with each cell known as PRIORITIZED_ORDER(Factor)
Test cases are ordered in the test suite by the summation of PRIORITIZED_ORDER(Factor)
STOP

Fig. 1 Flow chart of the proposed algorithm

6 Results and discussion

In this section, the comparative studies between different prioritization techniques have been discussed.

6.1 APFD measures

The effectiveness of a prioritized test suite is checked using a metric called Average Percentage of Faults Detected (APFD) (Elbaum et al. 2000). It calculates the weighted average of the percentage of faults detected during the test suite execution. A higher APFD score indicates that a test suite can detect all faults in the SUT as quickly as feasible. Its value ranges from 0 to 100. On the *x*-axis, if the test suite executed percentage is taken, and on the *y*-axis, faults detected percentage is considered, then the APFD value is the area under the curve. Mathematically, it can be calculated as given below in Eq. (11).

$$\mathrm{APFD} = 1 - \left(\frac{\mathrm{TF}_1 + \mathrm{TF}_2 + \ldots + \mathrm{TF}_m}{m * n}\right) + \left(\frac{1}{2 * n}\right) \quad (11)$$

where TF_i = location of the first test in test suite T that reveals fault, m = total count of faults surfaced by a T, n = total count of test cases in a T.

6.2 Comparative study

This part of the paper explains the comparative analysis between the above technique and the various other established techniques.

6.2.1 Comparative study of project 1

The given method has been compared with many other prioritization techniques, for example, no prioritization, reverse prioritization, random prioritization, and existing techniques found in the literature. The computing APFD value of each methodology is compared with each other.

6.2.1.1 Comparison to earlier research work This section focuses on the comparative study of the proposed prioritization order with the previous work of Kavita and

Table 9 Dataset of project 2

Test cases/faults	T1	T2	T3	T4	T5	T6	T7	T8
F1		×	×			×		
F2	×			×				×
F3		×			×		×	
F4	×			×				
F5			×					
F6					×		×	
F7	×		×			×		
F8			×				×	
F9	×			×			×	
F10					×			
No. of faults detected	4	2	4	3	3	2	3	2
Execution time (ms)	7	4	5	4	4	3	4	2

Table 10 Factors namely RFT, NFD, TARD, TCE values for test cases T1…T8

	RFT	**NFD**	**TARD**	**TCE**
T1	5.714	4	3.429	40
T2	5	2	16	20
T3	8	4	8	40
T4	7.5	3	16.5	30
T5	7.5	3	13.5	30
T6	6.667	2	14.001	20
T7	7.5	3	1.5	30
T8	10	2	16	20

Table 11 Normalized matrix

	RFT	NFD	TARD	TCE
T1	0.5714	1	0.208	1
T2	0.5	0.5	0.970	0.5
T3	0.8	1	0.484	1
T4	0.75	0.75	1	0.75
T5	0.75	0.75	0.818	0.75
T6	0.667	0.5	0.849	0.5
T7	0.75	0.75	0.091	0.75
T8	1	0.5	0.970	0.5

Sureshkumar (2010), Tyagi and Malhotra (2015), and Nayak et al. (2016). Table 15 provide all the prioritization order for the given dataset.

6.2.1.2 Comparison with other approaches for prioritization Based on the other existing methods of prioritization, for example, no prioritization, random prioritization, and reverse prioritization, we compared these existing techniques with our proposed method, which is given in Table 16.

The APFD value of the proposed technique is compared with the other prioritization techniques. Table 17 summarizes all the APFD percentage that proves that the given algorithm is better than the previous works.

The APFD percentage of the above-mentioned techniques is shown in Fig. 2a–e respectively.

6.2.2 Comparative study of project 2

This section of the chapter deals with the comparative study between the proposed technique and other existing techniques.

6.2.2.1 Comparison with other approaches for prioritization The proposed approach is compared with different prioritization techniques such as no prioritization, reverse prioritization and random prioritization, as represented in Table 18. These approaches will be analyzed by computing the APFD value for each method as given in Table 19.

The APFD percentage of the above mention techniques is shown in Fig. 3a–d respectively.

Table 12 Availability matrix

	RFT	NFD	TARD	TCE
T1	1	1	1	1
T2	1	1	1	1
T3	1	1	1	1
T4	1	1	1	1
T5	1	1	1	1
T6	1	1	1	1
T7	1	1	1	1
T8	1	1	1	1
SUM	8	8	8	8

Table 13 Rank matrix

	RFT	NFD	TARD	TCE
T1	0.143	0.250	0.052	0.250
T2	0.125	0.125	0.243	0.125
T3	0.200	0.250	0.121	0.250
T4	0.188	0.188	0.250	0.188
T5	0.188	0.188	0.205	0.188
T6	0.167	0.125	0.212	0.125
T7	0.188	0.188	0.023	0.188
T8	0.250	0.125	0.243	0.125

6.3 Discussion

The efficiency of the suggested prioritization strategy and the algorithms outlined above can be measured by the rate of fault detection (both control and prior work techniques). The speed with which an algorithm can detect all faults in SUT is the subject of this component. Defects can be introduced into a system in two ways: intentionally (by purposefully injecting or seeding faults) or naturally (by accidentally introducing defects) (present due to program errors committed by the programmers unknowingly). Manually injecting the faults is used in all of the investigations in this chapter, followed by an experiment to test the algorithms' efficiency. The APFD measure has been used to evaluate a prioritized test suite's efficacy. The higher the value of this metric, the better the algorithm is in detecting faults.

Table 16 Various prioritization techniques and their test case sequences

Prioritization Techniques	Test case sequence
No prioritization	T1, T2, T3, T4, T5, T6, T7, T8, T9, T10
Random prioritization	T2, T9, T5, T7, T4, T6, T8, T10, T3, T1
Reverse prioritization	T10, T9, T8, T7, T6, T5, T4, T3, T2, T1
Proposed order	T8, T4, T6, T2, T1, T5, T10, T3, T9, T7

For Project 1, the APFD percentage of the proposed method is contrasted with six other existing prioritized techniques such as no prioritization, random prioritization, reverse prioritization, prioritized order by Kavitha and Sureshkumar (2010), prioritized order by Tyagi and Malhotra (2015), prioritized order by Nayak et al. (2016), and the proposed technique are given in Table 17. The APFD score of the proposed prioritization technique is 84% which is better than the others, and also it provides the optimal ordering by normalizing the values of each factor. For Project 2, the proposed method is compared with three other existing prioritized techniques. The obtained APFD percentage of all the prioritization techniques is represented in Table 19.

Nayak et al. (2017) have obtained the sequence of test cases for Project 1 as T8, T4, T6, T2, T1, T5, T9, T10, T3, and T7 and for Project 2 got the sequence as T3, T4, T5, T1, T8, T2, T6, and T7. By implementing the suggested

Table 14 Sum of rank of each test case

Test case	T1	T2	T3	T4	T5	T6	T7	T8
Rank	0.695	0.618	0.821	0.814	0.769	0.629	0.587	0.743

Table 15 Test cases ordering for previous work and proposed approach

Prioritization techniques	Test case sequence
Prioritized order by Kavita and Sureshkumar (2010)	T8, T4, T9, T6, T5, T2, T1, T10, T3, T7
Prioritized order by Tyagi and Malhotra (2015)	T8, T4, T6, T9, T2, T1, T5, T10, T3, T7
Prioritized order by Nayak et al. (2016)	T8, T4, T9, T6, T2, T1, T5, T10, T3, T7
Proposed order	T8, T4, T6, T2, T1, T5, T10, T3, T9, T7

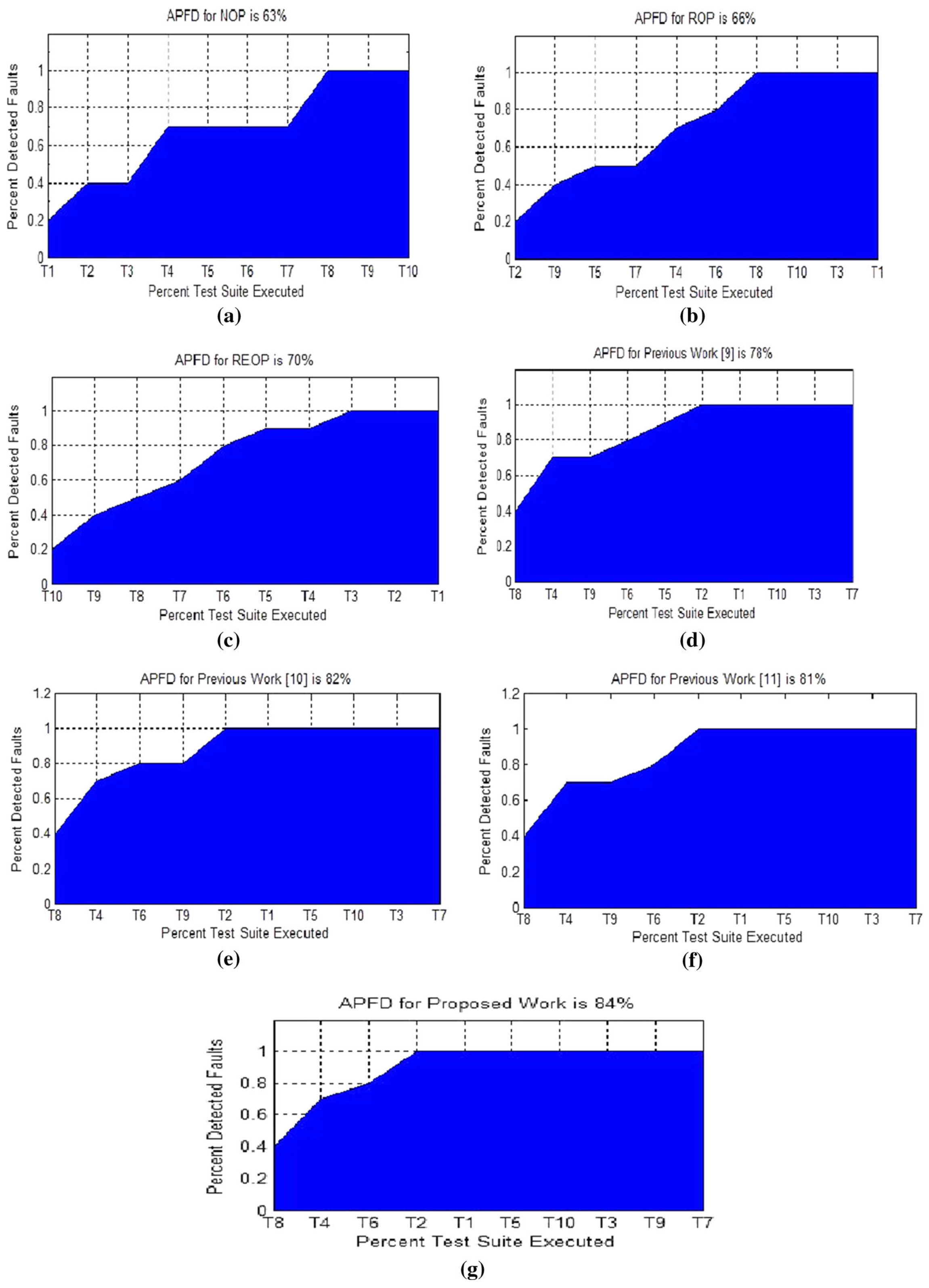

Fig. 2 Resemblance of various prioritization methods: **a** No order, **b** Random order, **c** Reverse order, **d** Prioritized order by Kavitha and Sureshkumar (2010), **e** Prioritized order by Tyagi and Malhotra (2015), **f** Prioritized order by Nayak et al. (2016), and **g** Proposed work

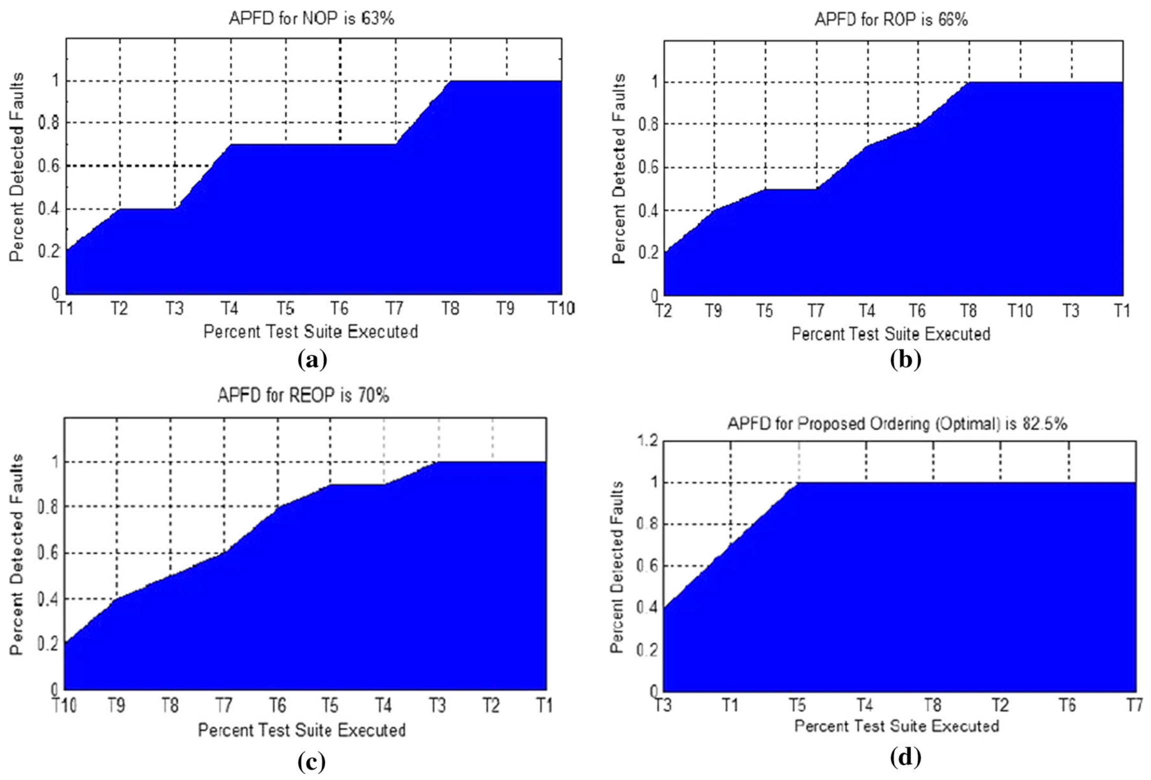

Fig. 3 Resemblance of various prioritization methods: **a** No order, **b** Random order, **c** Reverse order, and **d** Suggested work

prioritization technique, we obtained for Project 1 and Project 2 the sequences as T8, T4, T6, T2, T1, T5, T10, T3, T9, T7 and T3, T4, T5, T8, T1, T6, T2, T7 respectively. This shows after normalization using AHP, the proposed sequence by Nayak et al. (2017) changes since the dominance of the factors influencing the result is neutralized. Our suggested approach arranges the test cases in the most efficient manner.

This section also addresses the research questions mentioned in the Sect. 1 of this paper.

RQ1 The regression test case prioritization techniques can enhance the rate of fault detection of test cases in the test suite using AHP. As the number of choice criteria grows, it becomes more important to make a methodical decision that leads to an effective solution. AHP stands for Analytic Hierarchy Technique, and it is a multi-criteria decision-making process. It gives a logical framework for arranging an issue of decision-making. It is used to determine the priority of a set of criteria and to calculate the consistency ratio of those criteria using pair-wise comparisons of multiple metrics.

RQ2 Nayak et al. (2017) published a test case prioritization methodology, where they took four criteria into account and came up with an optimal ordering. At the time of getting the results, the authors made no mention of any association for any aspect. In other words, certain factors may have a more significant influence on the outcome than others. This multi-criteria software testing challenge may be handled in reality utilizing software metrics. Despite this, due to the use of numerous multiplicative factors, it is difficult to grasp and compute. The reason behind this is that the measures' values are highly variable. These values depend on certain factors like the number of faults detected by a test case, total time taken to detect defects, risks associated with the flaws, etc. The AHP approach avoids any factor having a dominant role in determining the result specified in for prioritizing the test cases. The results are normalized before deciding the order of test cases in order to avoid parameter dominance and provide an accurate answer by giving a rational framework for designing a decision-making challenge.

RQ3 The suggested methodology produces a higher APFD score, which can aid in the faster identification of defects from the aforementioned data as given in Table 17 for Project 1 and Table 19 for Project 2. It saves the software company time, money, and effort. The proposed strategy outperformed all previous methods and resulted in the best possible ordering. The performance is enhanced when the AHP technique is incorporated to design an efficient prioritization algorithm.

6.4 Limitations based on our work

There are some limitations found in this work as follows:

Table 17 APFD percentage for various prioritization techniques

Prioritization Techniques	APFD percentage
No Prioritization	63
ROP	66
REOP	70
Previous work Kavita and Sureshkumar (2010)	78
Previous work Tyagi and Malhotra (2015)	82
Previous work Nayak et al. (2016)	81
Proposed work (Optimal)	**84**

Table 18 Comparative study of other prioritization techniques

Prioritization Techniques	Test case sequence
No prioritization	T1, T2, T3, T4, T5, T6, T7, T8
Random prioritization	T5, T7, T1, T3, T6, T2, T8, T4
Reverse prioritization	T8, T7, T6, T5, T4, T3, T2, T1
Proposed order	T3, T4, T5, T8, T1, T6, T2, T7

Table 19 APFD percentage for various prioritization techniques

Prioritization techniques	APFD percentage
No Prioritization	76
ROP	75
REOP	68
Proposed work	**82.5**

(1) The APFD value is utilized to measure the rate of faults detection, and the APFD metric has its limitations by compromising the accuracy of the result. So to overcome this limitation, ANOVA and ANCOVA are better alternatives.

(2) We have taken a smaller number of test cases for the experimentation compared to the test cases available to us to minimize the computational complexity. So the result cannot be generalized until implemented on a real-time dataset.

(3) The AHP method has its issues. The AHP techniques consist of a higher level of mathematics (concept of Eigen vectors). Therefore we should be aware of the AHP process and the calculations.

7 Conclusion and future work

In this paper, a new prioritization algorithm has been formulated using AHP. It gives a logical framework for structuring an issue of decision-making. It is used to determine the priority of a set of criteria and to calculate the consistency ratio of those criteria using pair-wise comparisons of multiple measurements. Therefore, AHP can be used to solve the test case prioritization problem. The proposed approach is implemented on two VB-based projects. Desirable results have been acquired after applying the method on these two projects.

The proposed approach yielded high APFD results among the comparison algorithms. It is detected in the experiment that the applied technique is better than the other existing prioritization techniques based on the fault identification rate. The APFD score of the proposed prioritization technique for Project 1 is 84% which is better than the others, and also it provides the optimal ordering by normalizing the values of each factor. For Project 2, the proposed method is compared with three other existing prioritized techniques. The obtained APFD percentage of all the prioritization techniques like NOP, ROP, REOP, and proposed approach are 76, 75, 68 and 82.5% respectively. The proposed approach outclasses all other approach. The suggested approach can select efficient test cases and arranges them accordingly for execution resulting in enhanced fault detection in minimum time. This algorithm will be checked on the data collected from the repositories like real-world projects coded with different programming languages to adapt to the proposed approach.

Author contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Soumen Nayak, Chiranjeev Kumar and Sachin Tripathi. The first draft of the manuscript was written by Soumen Nayak and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding No funds, grants, or other support was received.

Data Availability Enquiries about data availability should be directed to the authors.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent This research does not involve any human participants or animals.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1007/s00500-022-07174-w.

References

Afzal N, Sadim M (2018) Software requirements selection using AHP. Int J Comput Sci Commun 9(2):47–52

Ahmad Y, Pirzada DS (2014) Using analytic hierarchy process for exploring prioritization of functional strategies in auto parts manufacturing SMEs of Pakistan. Sage Open. https://doi.org/10.1177/2158244014553560

Arrieta A, Wang S, Sagardui G, Etxeberria L (2019) Search-based test case prioritization for simulation-based testing of cyber-physical system product lines. J Syst Softw 149:1–34

Avritzer A, Weyuker EJ (1997) Monitoring smoothly degrading systems for increased dependability. Empir Softw Eng 2:59–77

Bajaj A, Sangwan OP (2019) A systematic literature review of test case prioritization using genetic algorithms. IEEE Access 7:126355–126375

Bajaj A, Sangwan OP (2021) Discrete cuckoo search algorithms for test case prioritization. Appl Soft Comput. https://doi.org/10.1016/j.asoc.2021.107584

Chi J, Qu Y, Zheng Q, Yang Z, Jin W, Cui D, Liu T (2020) Relation-based test case prioritization for regression testing. J Syst Softw 163:1–18

Cho Y, Kim J, Lee E (2016) History-based test case prioritization for failure information. In: 23rd Asia-Pacific software engineering conference (APSEC), IEEE

Dash U, Acharya AA (2022) A systematic review of test case prioritization approaches. In: Proceedings of international conference on advanced computing applications, Springer, pp 653–666

Elbaum S, Malishevsky A, Rothermel G (2000) Prioritizing test cases for regression testing. In: Proceedings the 2000 ACM SIGSOFT international symposium on software testing and analysis, Portland, Oregon, USA, pp 102–112

Emam SS, Miller J (2015) Test case prioritization using extended digraphs. ACM Trans Softw Eng Methodol 25(01):1–41

Hettiarachchi C, Do H, Choi B (2016) Risk-based test case prioritization using a fuzzy expert system. Inf Softw Technol 69:1–15

Huang R, Towey D, Xu Y, Zhou Y, Yang N (2022) Dissimilarity-based test case prioritization through data fusion. Softw Pract Exp. https://doi.org/10.1002/spe.3068

Jahan H, Feng Z, Mahmud SMH (2020) Risk-based test case prioritization by correlating system methods and their associated risks. Arabian J Sci Eng 45:6125–6138

Jeffrey D, Gupta R (2005) Test suite reduction with selective redundancy. In: 21st IEEE conference on software maintenance (ICSM’05), pp 549–558

Kavitha R, Sureshkumar N (2010) Test case prioritization for regression testing based on severity of fault. Int J Comput Sci Engg (IJCSE) 2(5):1462–1466

Khatibsyarbini M, Isa M, Jawani D, Hamed H, Suffian M (2019) Test case prioritization using firefly algorithm for software testing. IEEE Access 7:132360–132373

Klindee P, Prompoon N (2015) Test case prioritization for software regression testing using analytic hierarchy process. In: 12th International joint conference on computer science and software engineering (JCSSE), pp 168–173

Korel B, Koutsogiannakis G, Tahat LH (2007) Model-based test prioritization heuristic methods and their evaluation. In: Proceeding A-MOST’07 Proceedings of the 3rd international workshop on advances in model-based testing, ACM, pp 34–43

Li Z, HarmanHierons M (2007) Search algorithms for regression test case prioritization. IEEE Trans on Softw Eng 33(4):225–237

Luo Q, Moran K, Poshyvanyk D, Penta M (2018) Assessing test case prioritization on real faults and mutants. In: IEEE international conference on software maintenance and evolution (ICSME)

Maheswari R, Mala D (2013) A novel approach for test case prioritization. In: IEEE International conference on computational intelligence and computing research

Mohanty S, Mohapatra SK, Meko SF (2020) Ant colony optimization (ACO-min) algorithm for test suite minimization. Adv Intell Syst Comput (AISC) 1119:55–63

Nayak S, Kumar C, Tripathi S (2017) Enhancing efficiency of the test case prioritization technique by improving the rate of fault detection. Arab J Sci Eng 42(8):3307–3323

Nayak S, Kumar C, Tripathi S (2016) Effectiveness of prioritization of test cases based on faults. In: Proceedings third int’l conference on recent advances in information technology (RAIT-2016) IEEE. https://doi.org/10.1109/RAIT.2016.7507977

Noor T, Hemmati H (2017) Studying test case failure prediction for test case prioritization. In: Proceedings of the 13th international conference on predictive models and data analytics in software engineering (PROMISE), pp 2–11

Panwar D, Tomar P, Singh V (2018) Hybridization of Cuckoo-ACO algorithm for test case prioritization. J Stat Manag Syst Taylor and Francis 21(04):539–546

Qasim M, Bibi A, Hussain SJ, Jhanjhi NZ, Humayun M, Sama NU (2021) Test case prioritization techniques in software regression testing: an overview. Int J Adv Appl Sci 8(5):107–121

Rothermel G, Harrold MJ (1997) A safe, efficient regression test selection technique. ACM Trans Softw Eng Methodol 6(2):173–210

Rothermel G, Untch R, Chu C, Harrold M (2001) Prioritizing test cases for regression testing. IEEE Trans Softw Eng 27(10):929–948

Rotherrmel G, Harrold MJ, Ostrin J, Hong C (1998) An empirical study of the effects of minimization on the fault detection capabilities of test suites. In: Proceedings of the international conference on software maintenance, pp 34–43

Saaty RW (1987) The analytic hierarchy process-what it is and how it is used. Math Model 9(3–5):161–176

Shin D, Yoo S, Papadakis M, Bae D (2018) Empirical evaluation of mutation-based test case prioritization techniques. Softw Test Verif Reliab. https://doi.org/10.1002/stvr.1695

Shrivathsan A, Ravichandran K, Krishankumar R, Sangeetha V, Kar S, Ziemba P, Jankowski J (2019) Novel fuzzy clustering methods for test case prioritization in software projects. Symmetry 11(11):1–22

Srivastava, P. (2008) Test case prioritization. Journal of Theoretical and Applied Information Technology (JATIT), 178–181

Tahvili S, Bohlin M (2016) Test case prioritization using multi criteria decision making methods. Danish Soc Oper Res 26:9–11

Tyagi M, Malhotra S (2015) An approach for test case prioritization based on three factors. Int’l J Inf Technol Comput Sci 4:79–86

Wei D, Sun Q, Wang X, Zhang T, Chen B (2020) A model based test case prioritization approach based on fault urgency and severity. Int J Softw Eng Knowl Eng World Sci 30(02):263–290

Yan R, Chen Y, Gao H, Yan J (2022) Test case prioritization with neuron valuation based pattern. Sci Comput Progr 215:1–15

Yoo S, Harman M (2012) Regression testing minimization, selection and prioritization: a survey. Softw Test Verif Reliab 22(2):67–120