Journal of Electronic Testing (2022) 38:471–479
https://doi.org/10.1007/s10836-022-06024-9

Achieving Agility in Projects Through Hierarchical Divisive Clustering Algorithm

Janani Varun[1] · **R. A. Karthika**[2]

Received: 30 January 2022 / Accepted: 3 September 2022 / Published online: 23 September 2022

Abstract
Software products cannot be delivered to the market without proper testing. Only with the help of Testing, accuracy and quality of the product improves. Test personnel cannot compromise on the quality of the product and cannot afford to miss any defects. As the product's functionality expands, so does the testcase suite, and executing all of them takes more time and work. In this discussion, we'll look at how to use a machine learning approach called Hierarchical Divisive Clustering to optimise the test suite. With this approach, all the testcases are being considered as a single cluster in the beginning and during every iteration they are separated based on the similarity. This would help execute unique testcases without compromising on the quality which would help during any regression or sanity testing.

Keywords Testing · Regression · Smoke · Sanity · Test suite · Machine Learning · Hierarchical Divisive Clustering

1 Introduction

Machine Learning is a branch of artificial intelligence that allows a system to learn and develop without having to be explicitly programmed. Machine learning is the process of interpreting data structures and converting them into models that people can comprehend and use. Machine learning allows computers to construct models from data they are given as input, allowing them to automate decision-making. Unsupervised and supervised learning are the two types of machine learning tasks. In Supervised Learning, the desired outcome is labelled on the sample input. In this supervised learning strategy, patterns are utilised to predict label values on further unlabeled data. The model can be tested with new data after it has been trained to assess how well it works. Classification and Regression are two different sorts of supervised learning. When it is a categorised output variable (e.g., to categorise into "Red," "Blue," and "Green"), it is a classification problem; when it is a real number output variable, it is a regression problem (eg: predicting house price, predicting employee salary). The system is supplied with unlabeled and uncategorized input in unsupervised learning, and the model must be able to distinguish them based on the underlying structure. Clustering and Association. In Clustering, intrinsic groupings are discovered (for example, customers are grouped based on purchasing behaviour). In Association, rules that describe a substantial chunk of data are discovered (e.g.: People who buy x are more inclined to buy y as well.). Reinforcement learning is a sort of learning that involves interacting with the environment. When an agent performs successfully, he is rewarded, and when he performs wrong, he is penalised.

2 Literature Review

1. An Improved K-means Algorithm for testcase Optimization; Tan et al. [20]

Existing Methodology To improve the K-means algorithm and build a fuzzy clustering method, the degree of membership function was introduced. Developed a fuzzy mathematic

Communicated by V. D. Agrawal.

✉ Janani Varun
janani.rajalakshmi@gmail.com

[1] Research Scholar, Department of Computer Science and Engineering, Vels Institute of Science, Technology and Advanced Studies, Chennai, India

[2] Associate Professor, Department of Computer Science and Engineering, Vels Institute of Science, Technology and Advanced Studies, Chennai, India

approach to simplify software testcases while maintaining software fault detection reliability.

Challenges Using the Existing Methodology The clustering process fails to completely examine the many attributes of test instances, resulting in a discrepancy between clustering results and reality. Determination of the value of the fuzzy control factor. The number of testcases employed in the article was small.

2. Test-Suite Reduction based on K-Medoids Clustering Algorithm; Liu et al. [12]

Existing Methodology This method employs a greedy algorithm to process the streamlined test suite, ensuring that testcases are covered and errors are detected, resulting in a minimal test suite.

Challenges Using the Existing Methodology Every time the code coverage and complexity of each testcase were calculated, there were two steps/methods involved. Calculate the distance between sample points and centre points in the second phase, and then select the smallest test suite.

3. Test Suite Reduction via Evolutionary Clustering; Xia et al. [24]

Existing Methodology To combine comparable testcases into the same cluster, use the K-means algorithm. The evolutionary approach is then used to remove duplicate test cases based on the clustering results, using optimization objects representing coverage, fault, and cost criteria.

Challenges Using the Existing Methodology Code Coverage, execution speed, and fault location capabilities are the only factors considered, and redundancy elimination is not considered.

4. User -session- based testcases Optimization Method based on Agglutinate Hierarchy Clustering; Liu et al. [13]

Existing Methodology User Session Clustering based on Hierarchical Clustering estimates the distance between user sessions first, then clusters the initial testing cases and generates many test suites using the bottom-up agglutinate hierarchical clustering technique.

Challenges Using the Existing Methodology Mining user sessions for web application testing is a complex and systematic activity. The information about the user saved in the server database is also significant for mining, and USCHC should be utilised to test a lot more online app.

5. Testcase Prioritization Incorporating Ordered sequence of Program Elements; Wu et al. [23]

Existing Methodology Adaptive random testing (ART) is based on the idea of rearrangement of testcases in order to maximise the diversity of testcases in terms of similarity measure. When variety is increased, the number of defects is likely to increase as well.

Challenges Using the Existing Methodology Prioritizing testcases based on coverage rearranges them to optimise code coverage, but it does not ensure a high rate of fault discovery.

6. Effective testcase Prioritization method based on Fault severity; Wang et al. [22]

Existing Methodology Fatal defects include software crashes, software collapses, and software anomalous exits. Serious Error: Software prerequisites were not met. General Fault: The software's execution does not match the software's instructions. Minor Error: Has a minor impact on software functionality.

Challenges Using the Existing Methodology Counting the coverage rate of each testcase and selecting the one with the highest coverage rate takes time and does not focus on fault detection.

7. Requirement based testcase Prioritization; Kavitha et al. [11]

Existing Methodology Customer Priority is allocated to each demand on a scale of 1 to 10, with 10 being the highest customer priority. Complexity of Implementation: Each need is assessed and given a score between 0 and 10. Changes in Requirements: The number of times a requirement has been changed over the development cycle on a 10-point scale.

Challenges Using the Existing Methodology This strategy prioritises testcases based on customer needs, has a big test suite to complete, and is more time demanding.

8. An Insight into testcase optimization: Ideas and trends with future perspectives; Gupta et al. [10]

Existing Methodology The first group of projects focuses on optimization strategies in testing domains. Non-traditional adequacy criteria and related research are the focus of the second category. The third category looks for optimization tactics that have never or very rarely been used in software testing but have the potential to be used in the future.

Challenges Using the Existing Methodology A technique based on general purpose computing on graphics processing units (GPGPU) has been used in the optimization disciplines.

9. Optimization of Testsuite- testcase in Regression Test; Ansari et al. [5]

Existing Methodology Prioritize and select testcases based on the level of risk: Testcases should be prioritised in order of risk exposure.Select only those testcases that pose an intolerable, significant, or moderate risk. Testcase minimization based on specifications: Choose the test cases that cover all of the functionality if the regression pool's testcases don't.

Challenges Using the Existing Methodology In this model, the testcase minimization technique is based on risk exposure, and not all testcases in the test suite are mapped to risk – an unimportant element for Test personnel to attribute risk factors to testcases.

10. Empirical evaluation of Pareto efficient multi- objective regression testcase prioritization; Epitropakis et al. [9]

Existing Methodology Test case selection techniques Enhance the retest-all technique by selecting a subset of the entire test suite based on test criteria. Prioritizing testcases aims to organise testcases in a way that maximises test adequacy as quickly as feasible.

Challenges Using the Existing Methodology Pareto analysis is based on the 80–20 principle, which states that 20% of causes result in 80% of effects, which does not guarantee quality assurance.

3 Quality Assurance

Because structural quality is based on the engineering team's ability, it is ensured by code review, analysis, and rewriting. Quality assurance, quality control, and testing are examples of quality management duties that can aid in the functional aspect's preservation [4]. The Software Testing Life Cycle (STLC) is a collection of methods for ensuring that software quality goals are met. The STLC strategy includes both verification and validation. A methodologically organised series of activities for certifying a software product.

3.1 Phases of STLC

1. Requirement Analysis

   The Quality Assurance team meets with stakeholders to get a complete picture of the demand. And the team understands what is to be tested.
2. Test Planning

   The most efficient phase for defining test plans and calculating the projected testing effort and cost.
3. Test Case Development

   The phase in which the testing team creates testcases and test data for them. The same is examined by the quality assurance team.
4. Test Execution

   After the testcases have been generated [15, 17] and the environment and test data have been set up, the test execution phase begins. In this phase, testcases are run, and if the product passes the test, the testcase is passed; otherwise, the testcase fails, and a defect is recorded.

4 Issue with the Large Test Suite

The test suite continues to develop as the product's functionality grows, as do the testcases for the relevant requirements [4, 14]. It's never easy to manually run all of the testcases for each regression/smoke/sanity test [5, 16]. Simultaneously, Test personnel are unable to perform random testcases or compromise on product quality. Rerunning all of the testcases is impractical due to a lack of resources. There has to be a mechanism to optimise the testcases [27] that will be run once the build is provided, ensuring that no faults are missed [19].

a) Based on the execution counts of programme pieces in an ordered sequence

   Two approaches are discussed in this technique: generic testcase prioritisation [7] and version specific testcase prioritisation. While coverage-based testcase prioritisation rearranges testcases to maximise code coverage using code coverage statistics as a proxy, it does not ensure a high rate of fault discovery. Adaptive random testing (ART) [23] generates a candidate set of not-yet-selected test cases, with the test case farthest from the prioritised test suite picked as the next. It increases the likelihood of an increase in the number of faults [6]. The Farthest First algorithm is introduced in this publication. The essential method is prioritisation, which involves choosing a test scenario that produces the highest code coverage. The procedure SelectNextTestCase is then called on a regular basis based on Pareto Methodology [9] to choose unordered test cases into a prioritised collection until all testcases have been re-ordered [18, 26].

b) Optimize test case based on Fault Severity

Four fault levels are:

Fatal defects include software crashes, software collapses, software anomalous exits, data loss, and difficulty in repair [3].

Serious Fault: Software criteria aren't being met, the software can't be utilised regularly, and data is being lost but is easily recoverable.

General Fault: The software's execution does not match the software's instructions.

Little Fault: Has a minor influence on software functionality or is inconvenient to use.

Minor Fault: Has a minor impact on software functionality or is inconvenient to use [22].

The steps for determining the testcase execution sequence are as follows:

1. For each testcase, calculating the coverage rate
2. Choose the testcase with the highest coverage rate.
3. Go to step 4 when all programme codes have been covered or when the remaining testcases can't cover the uncovered programme code.
4. For the remaining testcases, update the code coverage rate. Repeat steps 2–4 until all of your testcases have been prioritised.

c) Requirement based testcase Prioritization

Customer priority, requirement modifications, and implementation complexity are the three criteria used in this methodology [11].

a) Customer Priority: Each demand is assigned a number between one and ten, with ten being the highest. To improve customer satisfaction, the top priority [2] needs are extensively evaluated early on.
b) Difficulty of Implementation: Each need is rated from 0 to 10, with higher numbers indicating greater complexity. Expect more issues with requirements with a high level of implementation complexity.
c) Requirement Changes: The number of times a requirement has been altered over the development cycle on a 10-point scale [11].

5 Related Work Using Hierarchical Divisive Clustering

i) Clustering Methodology

Data clustering is an unsupervised machine learning technique for organising data such that patterns can be found. Objects in a cluster have a high degree of resemblance or correlation with one another and are distinct from those in other groups [1]. Clustering can be done in two ways: hierarchical and partitional clustering. In hierarchical approaches, an n-object dataset is dissected into a hierarchy of groups, and the result is represented by a dendrogram, a tree structure diagram in which the root node represents the entire dataset and each leaf node represents a single dataset object. There are usually two general approaches for the hierarchical clustering: agglomerative and divisive.

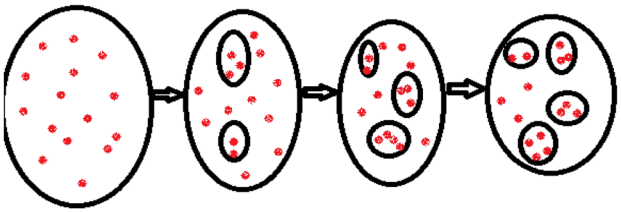

Fig. 1 explaining the hierarchical divisive clustering.

ii) Hierarchical Divisive Clustering

The Divisive Hierarchical Clustering technique is shown in Fig. 1 is a top-down strategy that treats the entire data set as a single cluster or root node at first, then gradually separates the data into multiple clusters downstream based on particular data qualities [8]. Top-down clustering requires a mechanism for breaking a cluster containing all of the data, then recursively splitting clusters until all of the data is broken into singletons [25].

iii) Finding testcase Similarities

To determine the similarity score and put the comparable testcases in a cluster, find the similarity between the testcases [21]. For testcase optimization, one testcase from a cluster can be executed for any regression/ smoke testing [10]. If certain testcases are unique and are dissimilar from the remaining, then those testcases must be run, and the similarity score will be zero. In this approach, Jaccard similarity score is being calculated and the steps are being mentioned below:

1. Lowercase all text
2. Tokenize
3. Remove Stop words
4. Remove punctuation
5. Lemmatize
6. Calculate intersection/ union between testcases

Jaccard Distance—The Jaccard Index is used to determine how similar two finite sets are. Jaccard Index can be used to calculate Jaccard Distance.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Where A and B are two different test cases represented here, and the similarity score is zero if the

testcase is unique and does not match others, otherwise the testcase is represented with a similarity score until it reaches 100.

iv) Input Data

1. Input is a CSV file with testcases. It can be of 3 styles:
   Type 1: One testcase per row.

ID	Testcase_No	Title	Description	Expected Result
1	TC_01	Test_Verify_whether_FIS_Email_entry_is_avilable_in_CDCONTMETHTP	Query the CDCONTMETHTP table using the following query: select * from CADDBO.CDCONTMETHTP where	The value in the NAME column should be 'FIS Primary Email' and the value in the DESCRIPTION column should be 'FIS Primary Email'.
2	TC_02	Test_Verify_code_table_XCDSRCPARTYTP	Query the XCDSRCPARTYTP table using the following query: Select * from XCDSRCPARTYTP where XSRCPARTY_TP_CD in (100009, 100010, 100011, 100012)	The following value combinations should be present in the NAME and DESCRIPTION columns : 'I' - 'Individual Client', 'C' - 'Corporation Client', 'RI' - 'Related Individual', 'RC' - 'Related Corporation' .

   Type 2: One testcase split in multi rows (no ID repetition).

SL.No	Test Name	Step Name	Description	Expected Result
1	Hadoop_Upg_TC001_JC_Award	1	Open CAISO PortalURL: https://portalmap.caiso.com-MST-	CAISO screen should be displayed.
		2	Click on 'CMRI'	CMRI page should be displayed
		3	Navigate to CMRI > Day-Ahead> Select Day Ahead Generation market Results	Day Ahead Generation market Results screen should be displayed

   Type 3: One testcase split in multi rows (ID repetition).

ID	Scenario	Steps	Expected Result
POS-22794	Sale_Tax Exempt_Verify the mandatory fields when tax information is added to guest's profile	Search a loyalty customer	Guest maintanace screen is displayed
POS-22794		Click on Save & Continue	Tax Exempt screen is displayed
POS-22794		Click on Select and Continue button	Tax is exempted from the transaction
POS-22794		Select the tender method as Cash ,and complete the t	Transaction completed successfully and the receipt is generated
POS-22794		Verify the Receipt generated	The total amount is updated correctly and the tax is shown as 0.00
POS-22522	Manager Functions_Paid Out_Verify that the maximum amount limit for OTHER paid out is $100.00	Login to Register with Manager Id and password	Manager is able to login
POS-22522		Skip Customer Search	User is navigated Scan Upc Page
POS-22522		Click manager functions and select paid out	Paid out option are displayed
POS-22522		Select OTHER Paid Out options and proceed	A page is displayed for entering amount
POS-22522		Enter the amount $101.00 and proceed	Validation prompt is displayed

2. Other inputs as required from the user are:

a) Unique column (index column): This is the column that must be provided as input that has unique ids in it.

ID	Scenario	Steps	Expected_Result	ClustIndex	similarity
TC-20493	TC01_Verify the service HTTP and HTTPS	Trigger the HTTP Swagger Account URL	URL triggered successfully and all the request details as expected	51	99.83
	TC01_Verify the service HTTP and HTTPS	Trigger the HTTPS Swagger Account URL	URL triggered successfully and all the request details as expected	51	99.83
TC-20503	TC01_Verify the Service HTTPS and HTTP	Trigger the HTTP Swagger URL	URL triggered successfully and all the request details as expected	51	99.83
	TC01_Verify the Service HTTPS and HTTP	Trigger the HTTPS Swagger URL	URL triggered successfully and all the request details as expected	51	99.83
TC-20515	TC01_Verify the Service HTTPS and HTTP	Trigger the HTTP Swagger Policy URL	URL triggered successfully and all the request details as expected	51	99.83
	TC01_Verify the Service HTTPS and HTTP	Trigger the HTTPS Swagger Policy URL	URL triggered successfully and all the request details as expected	51	99.83
TC-20542	TC01_Verify the Service HTTPS and HTTP	Trigger the HTTP Swagger URL	URL triggered successfully and all the request details as expected	51	99.83
	TC01_Verify the Service HTTPS and HTTP	Trigger the HTTPS Swagger URL	URL triggered successfully and all the request details as expected	51	99.83

Fig. 2 Test cases with High Similarity with a score of 99.83

b) Clustering columns: This is the column that must be provided for the optimisation to happen (either test scenarios alone or test scenario and expected result or test scenario, test steps and expected result)
c) Threshold Value: This is the value provided by the user based for the similarity to be looked upon. If the optimisation has to be done for a regression/ smoke testing, then the user chooses the threshold value in the range of 70%. If redundant test cases are to be removed from the suite, then the user chooses the threshold value in the range of 90%

3. Upload the input file in the format of '.csv' file.
4. Output will also be in the format of '.csv' file containing the cluster ids- to which cluster the test case belongs to and the percentage of similarity in testcases for each cluster

v) **Algorithm explained in steps**

1. Pre-processing:

a. The data and all the parameters from the job request are fetched from DB.
b. Row merging takes place if the testcases are split into multiple rows.
c. Column merging takes place for all the columns selected in 'Optimize' field during job request creation.

2. Clustering algorithm:

a) For each individual testcase, the tokens (words) are extracted.
b) The numbers and symbols are also filtered out from the tokens, such that only alphabets remain.
c) The tokens are then compared with a common stopwords list (a, an, that, this) and any matching words are removed.
d) The tokens are then placed in a Set data structure. Individual sets are maintained for individual TCs.
e) Once all the testcases are reduced to their respective sets of tokens, Jaccard Similarity is

ID	Scenario	Steps	Expected_Result	Clu	simila
TC-20487	TC02_Verify the submission API request response in DB for LOB=D&O	Login to Escape application	Login Successful	17	84.44
	TC02_Verify the submission API request response in DB for LOB=D&O	Fill all the mandatory details in account screen, perform OFAC search	Account created successful	17	84.44
	TC02_Verify the submission API request response in DB for LOB=D&O	Select LOB=D&O, Eff date, Expiration date should be within 30 days fron	Submission created successfully	17	84.44
	TC02_Verify the submission API request response in DB for LOB=D&O	Select Bound LOB=D&O, Eff date, Expiration date should be within 30 da	Policy created successfully	17	84.44
	TC02_Verify the submission API request response in DB for LOB=D&O	Open the swagger url - http://enappwst1/SubmissionTypeAPI/swagger	Swagger details entered successfully	17	84.44
	TC02_Verify the submission API request response in DB for LOB=D&O	Click on 'Execute' button Response should come	Submission Type, Submission num and pol	17	84.44
	TC02_Verify the submission API request response in DB for LOB=D&O	Open server ENICDB-DEV\ENIC_QA in DB and run the query	Swagger requested reponse details are log	17	84.44
TC-20490	TC05_Verify the submission API request response in DB for LOB=Surety	Login to Escape application	Login Successful	17	84.44
	TC05_Verify the submission API request response in DB for LOB=Surety	Fill all the mandatory details in account screen, perform OFAC search	Account created successful	17	84.44
	TC05_Verify the submission API request response in DB for LOB=Surety	Select LOB=Surety, Eff date, Expiration date should be within 30 days fr	Submission created successfully	17	84.44
	TC05_Verify the submission API request response in DB for LOB=Surety	Select Bound LOB=Surety, Eff date, Expiration date should be within 30	Policy created successfully	17	84.44
	TC05_Verify the submission API request response in DB for LOB=Surety	Open the swagger url - http://enappwst1/SubmissionTypeAPI/swagger	Swagger details entered successfully	17	84.44
	TC05_Verify the submission API request response in DB for LOB=Surety	Click on 'Execute' button Response should come under below format	Submission Type, Submission num and pol	17	84.44
	TC05_Verify the submission API request response in DB for LOB=Surety	Open server ENICDB-DEV\ENIC_QA in DB and run the query	Swagger requested reponse details are log	17	84.44
TC-20589	TC05_Verify the submission API request response in DB for any LOB	Login to Escape application	Login Successful	17	84.44
	TC05_Verify the submission API request response in DB for any LOB	Fill all the mandatory details in account screen, perform OFAC search	Account created successful	17	84.44

Fig. 3 Test cases with Medium Similarity with a score of 84.44

A	B	C	D	E	F
ID	Scenario	Steps	Expected_Result	Clu	similar
TC-20585	TC01_Verify the Request Uri in DB for Control number API 'POST' reques	Trigger Swagger Control number API URL	Swagger Control number API URL triggered successful	30	80
	TC01_Verify the Request Uri in DB for Control number API 'POST' reques	Click on 'POST' request	Details of POST request populated successful	30	80
	TC01_Verify the Request Uri in DB for Control number API 'POST' reques	Click on 'Try it Out' option	Control number request body should get enable	30	80
	TC01_Verify the Request Uri in DB for Control number API 'POST' reques	Enter the payload with all the Control number details in POST	Details entered as expected	30	80
	TC01_Verify the Request Uri in DB for Control number API 'POST' reques	Click on 'Execute' button	Response details should get display	30	80
	TC01_Verify the Request Uri in DB for Control number API 'POST' reques	Verify the response with Control number details	Control number details verified successful	30	80
	TC01_Verify the Request Uri in DB for Control number API 'POST' reques	Verify Request Uri details in Database	Request Uri details logged and populated successful i	30	80
TC-20586	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Trigger Swagger Control number API URL	Swagger Control number API URL triggered successful	30	80
	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Click on 'PUT' request	Details of PUT request populated successful	30	80
	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Click on 'Try it Out' option	Control number request body should get enable	30	80
	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Enter existing Control number in control number field.	Entered existing Control number.	30	80
	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Enter the payload and modify Control number details in PUT r	Details entered as expected	30	80
	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Click on 'Execute' button	Response details should get display	30	80
	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Verify the response with Control number details	Control number details verified successful	30	80
	TC02_Verify the Request Uri in DB for Control number API 'PUT' request	Verify the Request Uri details in Database	Request Uri details logged and populated successful i	30	80
TC-20587	TC03_Verify the Request Uri in DB for Control number API 'PATCH' reque	Trigger Swagger Control number API URL	Swagger Control number API URL triggered successful	30	80

Fig. 4 Test cases with Low Similarity with a score of 80

calculated by comparing the token sets of each testcase with all the others.

f) The above step gives a 2D linkage matrix where the similarity of each testcase with all the other testcase is stored.

g) The linkage matrix is passed to the Hierarchical Clustering algorithm which clusters the testcases according to the Similarity Threshold.

h) Now, if there are N clusters, then the hierarchical clustering is further performed N times, cluster by cluster.

i) At a time only the testcases belonging to the Nth clusters are used for clustering.

j) Rest all the steps are same, except for the similarity calculation technique.

k) For all tokens in a test case, an average of all the word vector is taken as the similarity score.

l) Post Processing- JSON and CSV files are created from the above result and saved in DB.

Thus, we can cluster as:

- Highly Similar: In this paper, similarity threshold has been set as 80, hence testcases similarity greater than 94 are considered to be highly similar is shown in Fig. 2.
- Medium Similar: For similarity threshold set as 80, test cases similarity greater than 87 will be considered as Medium similar is shown in Fig. 3.
- Low Similar: For similarity threshold set as 80, test cases similarity greater than 80 will be considered as low similar is shown in Fig. 4.
- Unique: For similarity threshold set as 80, test cases similarity less than 80 will be considered as unique and the similarity score will be zero

Below is the output for the testcases that are optimized. Scenario, Steps and Expected Result are the fields chosen for optimization.

Test case optimization has been tried out for industry standard projects and the results of optimization for various modules has been provided below Tables 1 and 2:

As the Table 1 states, the total number of testcases in the testsuite is 658. Scenario, steps and expected result were chosen as the fields for optimization with a similarity threshold level of 80%. From the clusters that had high, medium

Table 1 explaining test case optimization numbers

Summary	High Similarity	Med Similarity	Low Similarity	Unique Test cases	Total
Original number of test cases	243	357	52	6	658
% test cases selected from cluster	57%	35%	69%	100%	47%
Sample selected from cluster	140	128	36	6	310

Table 2 explaining effort saved out of test case optimization

Test Suite Reduction	TE Effort pre-clustering	TE Effort post-clustering	Effort Savings -Test Execution
53%	5 person days	3 person days	40%

and low similarity with unique testcases, a total of 47% of testcases were chosen for regression execution.

Table 2 explains on the percentage of reduction in number of testcases in the testsuite and overall effort savings during a regression testing.

6 Conclusion and Future Work

The Hierarchical cluster divisive algorithm is used to optimise test cases, resulting in a decrease of the number of test cases in the larger test suite as well as on the removal of redundant and duplicate testcases. Further prioritisation of testcases can be done using the optimised test suite to report issues faster.

Data Availability Statement Test case optimization has been tried out for industry standard projects with different functionalities and sizes. The one quoted here in this paper refers to web testing project with 600 h of testing effort estimation and the results of optimization for various modules has been provided.

Declarations

Conflict of Interest The authors declare that we have no conflict of interest.

References

1. Abraham J, Radhamani G (2014) Fuzzy C Means (FCM) Clustering Based Hybrid Swarm Intelligence Algorithm for Test Case Optimization. Res J Appl Sci Eng Technol 8:76–82. https://doi.org/10.19026/rjaset.8.943
2. Ahmed A A, Shaheen M, Kosba E (2012) Software testing suite prioritization using multi-criteria fitness function. In Proc 23rd IEEE Int Conf Comput Theory Appl (ICCTA) 160–166
3. Ahmed B (2016) Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. J Eng Sci Technol 12:737–753. https://doi.org/10.1016/j.jestch.2015.11.006
4. Ahmed B, Abdulsamad T (2015) Achievement of Minimized Combinatorial Test Suite for Configuration-Aware Software Functional Testing Using the Cuckoo Search Algorithm. Inf Softw Technol 66:13–29
5. Ansari SA, Devadkar KK, Gharpure P (2013) Optimization of test suite-test case in regression test. IEEE International Conference on Computational Intelligence and Computing Research 2013:1–4. https://doi.org/10.1109/ICCIC.2013.6724206
6. De Lucia A, Di Penta M, Oliveto R, Panichella A (2012) On the role of diversity measures for multi-objective test case selection. 2012 7th International Workshop on Automation of Software Test (AST) 145–151. https://doi.org/10.1109/IWAST.2012.6228983
7. Di Nucci D, Panichella A, Zaidman A, De Lucia A (2018) A test case prioritization genetic algorithm guided by the hypervolumee Indicator. IEEE Trans Software Eng 46:1–1. https://doi.org/10.1109/TSE.2018.2868082
8. Ding C, He X (2002) Cluster merging and splitting in hierarchical clustering algorithms 2002 IEEE International Conference on Data Mining, 2002. Proceedings 139–146. https://doi.org/10.1109/ICDM.2002.1183896
9. Epitropakis MG, Yoo S, Harman M, Burke EK (2015) Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. 234–245. https://doi.org/10.1145/2771783.2771788
10. Gupta N, Sharma A, Pachariya MK (2019) An Insight Into Test Case Optimization: Ideas and Trends With Future Perspectives. IEEE Access 7:22310–22327. https://doi.org/10.1109/ACCESS.2019.2899471
11. Kavitha RV, Kavitha VR, Kumar NS (2010) Requirement based test case prioritization. 2010 International Conference on Communication Control and Computing technologies 826–829. https://doi.org/10.1109/ICCCCT.2010.5670728
12. Liu F, Zhang J, Zhu EZ (2017) Test-suite reduction based on K-Medoids clustering algorithm. 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC) 186–192. https://doi.org/10.1109/CyberC.2017.38
13. Liu Y, Wang K, Wei W, Zhang B, Zhong H (2011) User-session-based test cases optimization method based on agglutinate hierarchy clustering. 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing. IEEE
14. Mala D J, Mohan V (2010) Quality improvement and optimization of test cases: A hybrid genetic algorithm-based approach. ACM SIGSOFT Softw Eng Notes 35:1–14
15. Marchetto A, Islam MM, Asghar W, Susi A, Scanniello G (2016) A Multi-objective technique to prioritize test cases. In IEEE Trans Softw Eng 42(10):918–940. https://doi.org/10.1109/TSE.2015.2510633
16. Mirarab S, Akhlaghi S, Tahvildari L (2012) Size-constrained regression test case selection using multicriteria optimization. In IEEE Trans Softw Eng 38(4):936–956. https://doi.org/10.1109/TSE.2011.56
17. Panichella A, Kifetew F M, Tonella P (2018) Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. IEEE Trans Softw Eng 44(2)122- 158
18. Shin Y, Harman M (2007) Pareto efficient multi-objective test case selection. In Proc Int Symp Softw Test Anal 140–150
19. Srivatsava PR, Mallikarjun B, Yang XS (2013) Optimal test sequence generation using firefly algorithm. Swarm Evol Comput 8:44–53
20. Tan TT, Wang BS, Tang Y, Zhou X (2019) An improved K-means algorithm for test case optimization. IEEE 4th International Conference on Computer and Communication Systems (ICCCS). https://doi.org/10.1109/CCOMS.2019.8821687
21. Verma AS, Choudhary A, Tiwari S (2020) Test Case Optimization using Butterfly Optimization Algorithm. 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence) 704–709. https://doi.org/10.1109/Confluence47617.2020.9058334
22. Wang Y, Zhao X, Ding X (2015) An effective test case prioritization method based on fault severity. 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS) 737–741. https://doi.org/10.1109/ICSESS.2015.7339162

23. Wu K, Fang C, Chen Z, Zhao Z (2012) Test case prioritization incorporating ordered sequence of program elements. 2012 7th International Workshop on Automation of Software Test (AST) 124–130. https://doi.org/10.1109/IWAST.2012.6228980
24. Xia C, Zhang Y, Hui Z (2021) Test suite reduction via evolutionary clustering. IEEE Access 9:28111–28121. https://doi.org/10.1109/ACCESS.2021.3058301
25. Yamada Y, Masuyama N, Amako N, Nojima Y, Loo CK, Ishibuchi H (2020) Divisive Hierarchical Clustering Based on Adaptive Resonance Theory. International Symposium on Community-centric Systems (CcS) 2020:1–6. https://doi.org/10.1109/CcS49175.2020.9231474
26. Yoo S, Harman M (2010) Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. J Syst Softw 83:689–701. https://doi.org/10.1016/j.jss.2009.11.706
27. Zheng W, Hierons R M, Li M, Liu X H, Vinciotti V (2016) Multiobjective optimisation for regression testing. Inf Sci 334:1–16

Janani Varun completed her undergraduate studies at Anna University, Chennai and her postgraduate studies at Anna University, Coimbatore. She completed a master's Programme in Artificial Intelligence that included Data Science, Machine Learning, and Deep Learning. She has been working in an IT company on Agile projects using Machine Learning and Artificial Intelligence. That's where she's run into a lot of roadblocks that made her take up a research problem related to Agile area and solve it with Machine learning algorithms. Her research presents a solution to a challenge that most projects face daily.

R. A. Karthika is currently working as an Associate Professor, in the Department of Computer Science and Engineering, Vels Institute of Science, Technology and Advanced Studies, Chennai, India. She obtained her Ph.D. degree from Noorul Islam University, Kanyakumari, India (2015). She has around 14 years of teaching experience and her areas of interest include networking, IOT, cloud computing and big data. She has published around 34 papers in national and international journals and she is member of various international accreditation bodies like CSI.