International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Machine Learning Approach for Regression Testing: A Case Study in Markov Chain Model

Neelam Rawat[1], Vikas Somani[2], Arun Kr. Tripathi[3]

Submitted: 15/01/2024 **Revised**: 23/02/2024 **Accepted**: 01/03/2024

Abstract: In the wake of digitization of business processes, Software has turned into an approach to carrying out the businesses while considering software testing as an investment not as an expense. Now-adays, businesses focus on prioritizing scalability so that they may allow testers to make con-tinuous integration and testing without affecting end user's experience towards the application. Regression Testing can be performed multiple times (if required) to either to whole application under test or to only part of the application to ensure the validity of the changes made. Because of the repetitive nature of Regression Testing, software testers need to automate the process to minimize the consumption of time and effort. This paper mainly focuses on one of the machine learning techniques i.e., Markov Chain Model that randomly selecting a path to generate test cases that may further identify those test cases which are most prospective to uncover the new bugs. This can be done by observing the transition from a state in which no bugs have been found to a state in which a bug has been found.

Keywords*: digitization, scalability, Regression Testing, consumption, bug*

1. Introduction

Software Testing [1][2] is an evaluation procedure that attempts to verify and validate the software and its functionality. Expanded utilization of software applications and the complexity of its components, as well as abbreviated period of evaluation of software quality, have expanded the significance and need for automation of test case prioritization [3] in software testing. Automated software testing [4][5] tool by utilizing Machine Learning techniques [6] limits and minimizes errors in testing process [7], yet in addition permits a speedier assessment. In consideration to these facts, a Machine Leaning framework is proposed in to prioritize the software test cases for automation of regression testing process [8].

The framework will provide a suitable test case as well as learning models for software testing life cycle [9] to automate the process by analyzing and categorizing ML algorithms applicable to distinct testing phases. A software program change must be checked for accuracy [10] before being implemented. Testing [11] and repairing any regressions in the current features is also crucial for the success of a software product. Any faults or bugs that may have snuck into the earlier functionalities following the relevant change are referred to as regression issues. Regression testing [12] must be done effectively to maintain an application operating flawlessly after each small or significant update. Regression test cases [13] must be carefully chosen for execution in order for regression testing to be efficient.

The sequencing of the testing of software modules [14] that are significant from the standpoint of the customer ultimately results in the selection of these regression test cases. This leads to the requirement for test case prioritization [15].

The business requirements, prior test cycle experience on the functionality of the existing features, and delivery schedules are taken into consideration when prioritizing test cases for regression testing. The goal is to propose one of the test case prioritization techniques [16] that will also incoperate risk-based testing [17]. The proposed technique will use test case execution history as the primary criterion of assessment. There are three key goals for this study. First, examine and summarize techniques for prioritizing test cases [18] based on their (Eloberate word their) history. Next, note the variations among the test case prioritization techniques [19] and lastly, evaluvate the proposed strategies to evaluate the efficacy.

The subsequent sections of the paper are structured as follows. *Section 1.1* provides an outline of the research background that lays the foundation for our proposed machine learning approach to test case prioritization. In *Section 1.2*, review and discuss the existing literature and research efforts that are relevant to our proposed machine learning approach for regression testing. *Section 2* presents our machine learning approach designed for regression testing, tailored to the intricacies of Markov Chain models, and describe the materials and methods that will be used to investigate and proper responces of the research questions, including the specific techniques for test case prioritization. In *Section 3*, deals with experimental results. *Section 4* discusses the implications of the outcones and their potential impact. Finally, the conclusions of the study will be presented, along with suggestions for future research in section 5.

[1,2] Department of Computer Science and Engineering, Sangam University, Atoon, Rajasthan 311001, India.
Email: neemarawat11@gmail.com
https://orcid.org/0000-0003-0759-6583

Department of Computer Science and Engineering, Sangam University, Atoon, Rajasthan 311001, India.
Email: vikas.somani@sangamuniversity.ac.in
http://orcid.org/0000-0002-2562-2581

[3]Department of Computer Applications, KIET Group of Institutions, Delhi-NCR,Ghaziabad, 201206 India.
Email: mailtoaruntripathi@gmail.com
http://orcid.org/ 0000-0001-5138-2190

1.1 Research Background

Test Case Prioritization

This section deals with concise overview of significant pieces of concerned work in the field of test case prioritization and its historical significance [20]. Regression testing, which constitutes 80 percent of the testing process [21], plays a vital role in evaluating software against updated requirements and swiftly fixing bugs. However, it presents challenges due to its time-consuming nature, limited resources, and frequent occurrence [22]. The real-time embedded system regression testing is particularly rigorous, given the strict timing constraints in the simulation environment and the oversight of various initiatives like Retest-All, Regression Test Selection (RTS), Test Case Prioritization (TCP) [23], and Test Suite Minimization (TSM). TCP is a technique employed in regression testing to optimize the order of test case execution.

The primary objective of TCP is to identify and execute the most significant test cases first, allowing the early detection of potential issues in the software [24]. [64] To achieve safe regression test selection, test cases that reveal at least one flaw are chosen using a safe selection technique [24], though this does not guarantee safety in all cases due to varying conditions. Conversely, unsafe test case selection techniques lead to the discarding of several test cases. TCP emphasizes that test cases having higher priority should be executed more swiftly during testing. Offline TCP is a cost and time-effective approach that is not considered an expense [25].

There are two primary forms of test case prioritization: first one is version-specific prioritization [26] and another one is generic prioritization. While prioritization ordering is beneficial for later versions of software. General test case prioritization [27] techniques and version-specific prioritization [28] techniques focus solely on optimizing test case order for a particular software version.

1.2 Related Work

Following are the highlights of different research studies related to prioritization techniques in associatation with machine learning techniques with reference to software testing. Each study applies various approaches and performance metrics to analyze the effectiveness of various proposed techniques. Datasets used in these studies vary from open-source projects to specific domains like automotive and financial institutions.

Ref. Id	Authors and Year	Prioritization Technique	Machine Learning Technique	Performance Metric	Data Set
[1]	Haghighatkhah et al. (2018) [1]	Diversity-Based Test Prioritization (DBTP) and History-Based Test Prioritization (HBTP)	Not mentioned	Time and number/percentage of Faults Detected.	Six open-source projects
[4]	Hajri et al. (2011)	Fault tendency of Requirements	Classification Framework	Not mentioned	Industrial product on automotive domain.
[6]	Vu Nguyen et. al. (2021)	Weighted Coverage Graph	Reinforcement Learning	Number of test faults detected	Nine data sets from two web applications
[8]	Lousada et al.	Network Approximator and Test Case Failure Reward	Reinforcement Learning	Normalized percentage of Fault Detection (NAPFD)	Financial Institution dataset
[10]	Bagherzadeh et al.	Ranking Models	Reinforcement Learning	Normalized Rank Percentile Average (NRPA) & Average Percentage of Faults Detected (APFD)	Not specified
[13]	Lachmann et al.	Failure Count, Failure Priority, Execution Cost, Requirement Coverage	Supervised machine learning - SVM Rank algorithm	APFD	Body Comfort System & Automotive Industry Data
[14]	Zhang et al.	Requirement Coverage	Metaheuristic optimization algorithm - Ant Colony Optimization Algorithm	Enhanced Average Percentage of Win-cost Coverage (eAPWC)	Triangle classification program
[15]	Zhao et al.	Code Coverage	Bayesian Networks	APFD	Java projects with mutants
[16]	Mirarab et al.	Change in source code, test coverage	Bayesian Networks	APFD	8 consecutive versions of a software
[18]	Lousada et al.	Random selection	Neural Network	APFD	NA
[20]	Carlson et al.	Code coverage & Complexity	Clustering approach	APFD	Microsoft Dynamics Ax (industrial software product)
[28]	Jahan et al.	Test Cases Complexity	ANN	Fault detection rate, accuracy, precision, and recall	Two applications - Registration-Verifier

					Application & Credit Card Approval System
[30]	Kashyap et al.	Model-based approach	Markov Chain & HMM	NA	Two case studies on a web application usage
[34]	Spieker et al.	Random selection, failed test cases with higher priority	Reinforcement Learning	NAPFD	Industrial data sets from ABB Robotics
[36]	Konsaard et al.	Coverage-Based apprach	Genetic Algorithm	Average Percentage of Code Coverage (APCC) metric.	NA
[39]	Sharif, A et.al.	History-Based Test Prioritization (HBTP)	Deep Neural Network	APFD NAPFD	Industrial and Synthetic Datasets
[46]	Gökçe, N. et.al.	Event sequence, pair-wise coverage	Unsupervised Neural Network Clustering	MSE	Case study on RealJukebox (RJB)
[47]	Gökçe, N. et.al.	Model-based approach	Un-supervised neural network	Number of Events Capturing Unique Faults, Repetitive Faults, Nonrepetitive Faults & All Faults	Web-based tourist services system
[51]	Gökçe, N. et.al.	Model-based apprach	Multilayer perceptron (MLP) - Feed Forword Neural Network	MSE	
[56]	Mirarab, S. et.al.	History-Based Test Prioritization (HBTP)	Bayesian Networks	APFD	Five open-source Java objects

While regression metrics are commonly used for evaluating machine learning techniques in [15][16][18][20][34][36][39][46][47] [51][56], TCP is a specific task where classification and clustering metrics can also be highly relevant and valuable for evaluation. TCP involves organizing test cases based on their likelihood of detecting faults or their importance in terms of covering critical functionalities. Therefore, using classification and clustering metrics can provide deeper insights into the performance of machine learning techniques for this task[65].

2. Proposed Method

The initial step of this study is to identify the meaning of the Research Queries as in *Figure 1* that helps to identify the objective of this paper. As the interest of applying ML techniques are evolving in TCP [29], through this paper we expect to appreciate and audit a portion of the parts of the software/ application. To accomplish this objective, we enunciated four Research Queries.

Through these inquiries, we plan to survey all the methods of ML utilized to show what the most proper strategies are to help Developers and Testers for additional investigation so that they may choose the suitable method for Test Case Prioritization [30]. We will likewise explore the most utilized metrics that evaluate the effectiveness [31] of the proposed techniques in TCP. In this regard, following research queries (RQs) are set that aims to answer based on the existing literature in the field. These questions are designed to guide the review process and provide a framework for test case prioritization using an effective machine learning technique.

Research Queries (RQs) defined as follows:

RQ1. Which machine learning techniques can be practical and efficient at various stages of the software testing life cycle to rank test cases?

RQ2. What are the qualities and shortcomings of each learning technique with regards to testing?

RQ3. How would we be able to decide the best possible type of learning technique for different stages of software testing process?

RQ4. What are the focused points where Machine Learning & Deep Learning contributes to software testing process?

2.1 Data Collection for each Research Queries

In our survey paper, 60 papers were comprehended for our detail review to identify ML techniques used for TCP. We then, at that point, extricated related information from the papers as indicated by defined research queries. Table 1 displays the essential data corresponding to each research inquiry. The kind of information contemplated the objective that each research query addresses.

Research Queries	**Extracted Data**
R**Q1**	ML methods employed for TSP and its noted advantages and disadvantages
R**Q2**	
R**Q3**	Feature sets utilized in data collecting and prediction processes
	Subjects, experiment ideas and evaluation metrics
R**Q4**	Results of Machine Learning-Based TSP Methods in Terms of Accuracy
	Accessibility of Datasets and Comprehensive Results

Table 1: Data Required for Each RQs

After reviewing 60 research papers, it was concluded that HMM model needs to be explored more. As the exhibited modelling approaches can be implemented in test plan generation, concept of test case selection, formulation of requirements [32], design and verification of systems. The whole review process insight us to capture how any ML model is important to generate test cases and prioritize [33] them using a likelihood-based method. The likelihood-based approach makes use of the probabilistic data pertaining to the system states, their interactions with system users, and their surroundings.

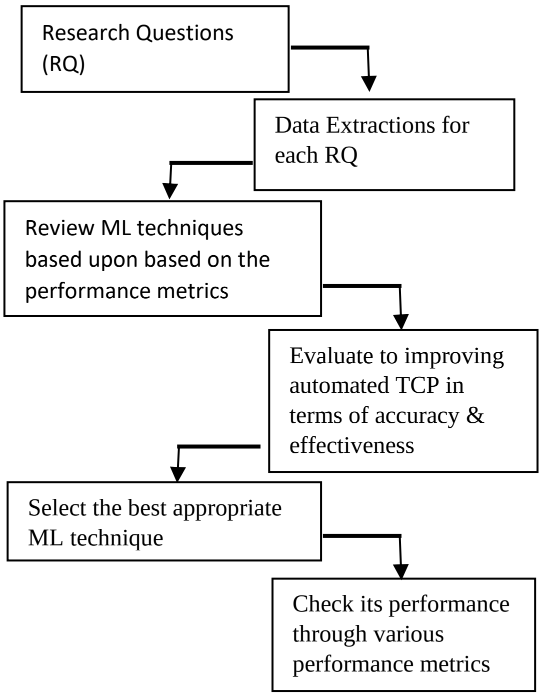

Research Questions (RQ)
Data Extractions for each RQ
Review ML techniques based upon based on the performance metrics
Evaluate to improving automated TCP in terms of accuracy & effectiveness
Select the best appropriate ML technique
Check its performance through various performance metrics

Fig. 1: RQs addressed in Research Papers

Furthermore, within this research, both a simulated case study and a real-world web application dataset are employed to analyse the efficiency of modelling and test case prioritization strategies. For investigation of system's time domain progression, we used Markov chains. Only software-based systems have been validated by the exhibited approach, and more study is required to determine its application to hardware spheres. The applicability of the method described in this study is also constrained by the fact that not all software systems have usage data sets that include time domain data. The strategy can be expanded in situations where time domain data are not accessible by modelling the system as discrete Markov chains and employing hidden Markov models (HMM). The exhibited methodology's ability to discover faults has not been tested, which is another study disadvantage. It is advised to conduct additional research, such as the creation of a software-based system and the seeding of it with recognized problems, to analyse the fault detection performance.

2.2 Test Cases based on Hidden Markov Model

A test case is a series of states or transitions that move through the Markov test chain from the starting state to the ending state when creating test cases for embedded software using the Markov use model. The following is how the Markov test chain is created from the usage model: The test chain is first constructed using the usage chain's structure, and each edge's transition probability is changed to a counter with an initial value of 0. The number of the edge then goes up by 1 as more test cases pass it as the testing process [34] progresses. Finally, depending on the statistical findings, the counter for each edge is transformed to transition probability. In this article, we offer a method for creating test cases [35] for software reliability tests based on the Markov consumption model.

Input: A Markov chain M, a positive integer n.

Output: A sequence of n test cases.

Step 1. Let A be an nXn matrix whose i, j entry is $p_{i,j}$ for $1 \leq i, j \leq n$.

Step 2. From initial distribution of M, generate a random vector x0 and let $x0 = (x_{0,1}, x_{0,2}, ..., x_{0,n})$, where $x_{0,j}$ is the component of x_0 corresponding to state j.

Step 3. For k = 1, 2, ..., n

Step 3.1 Generate a random number u from the uniform distribution on [0, 1].

Step 3.2 Compute $j = \min\{l \mid 1 \leq l \leq n \text{ and } u \leq A(x_k-1,l)\}$.

Step 3.3 Set $x_k = (x_{k1}, x_{k2}, ..., x_{kn})$, where x_{kj} is the component of x_k corresponding to state j.

Step 3.4 Output the test case x_k.

In the aforementioned algorithm, test instances are created using uniform random distribution that are compared to the Markov model's transition probability. This algorithm takes into account the statistical (transition) probability of code coverage [36] in addition to satisfying the randomness of the test case generation process and TCP [37] process. The prioritization of a test case Xk as in equation (1) computed as:

$$W(X_k) = W1(X_k) + W2(X_k) \quad (1)$$

where:

- $W1(X_k)$ is the weight of the test case X_k with respect to the test coverage.
- $W2(X_k)$ is the weight of the test case X_k with respect to the test case difficulty.

$W1(X_k)$ is computed as:

$$W1(X_k) = (C(X_k) / C_{max}) \quad (2)$$

where $C(X_k)$ is the number of covered requirements by the test case X_k and C_{max} is the total number of requirements.

$W2(X_k)$ is computed as:

$$W2(X_k) = (E(X_k) / E_{max}) \quad (3)$$

where $E(X_k)$ is the number of executed requirements by the test case X_k and E_{max} is the total number of requirements.

The importance weight of the test case X_k is computed as the sum of the weights $W1(X_k)$ and $W2(X_k)$ from equation (2) and (3).

2.3 Performance Measure

Following steps are taken to calculate the performance of HMM for prioritization of software test cases through the accuracy, precision, and recall values:

Step 1: Train the HMM model

- Define the observation sequences for each test case.
- Define the ground truth labels for each test case.
- Create an HMM model with the desired number of states.
- Fit the HMM model to the observation sequences.

Step 2: Predict the test case priorities.

- Calculate the probability of each observation sequence belonging to each state using the trained HMM model.
- Assign the most probable state to each observation sequence.
- Map the states to the corresponding ground truth labels to generate the predicted test case priorities.

Step 3: Calculate Accuracy, Precision, and Recall values.

- Compare the predicted test case priorities with the ground truth labels to calculate the fol-lowing:
- True Positives (TP): The number of test cases correctly classified as high priority.
- False Positives (FP): The number of test cases incorrectly classified as high priority.

- False Negatives (FN): The number of test cases incorrectly classified as low or medium priority.
- Calculate the Accuracy, Precision, and Recall values using the following formulas:

Accuracy = (TP + TN) / (TP + TN + FP + FN)
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)

3. Results

Out of 60 research papers reviewed, 34 papers including surveys were used Machine Learning techniques [38], 6 papers used Deep Learning techniques [39]-[44] and remaining ones are non-ML techniques. From figure 2, research papers that address research questions are typically structured in a way that includes 56.7% covers RQ1, 16.7% RQ2 & RQ3, 10% RQ1

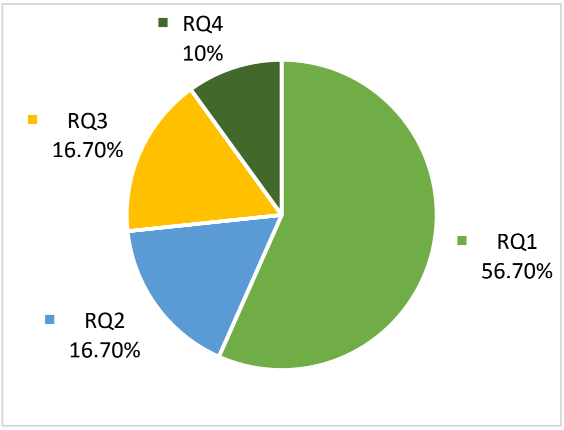


Fig 2: RQs addressed in Research Papers

A comprehensive and rigorous research design, and a detailed analysis of the data that provides insights into the topic being studied. In order to find the solutions, we have outlined the research question to summarize the utilization of ML techniques for test case prioritization [45][46] process with performance metrics mentioned in Figure 3. Our methodical survey aims to give priority values to choose defined ML techniques for TCP [47]. In this way, we intend to evaluate ML methods to ensure capability of improvising automated [48] TCP in terms of accuracy and effectiveness.

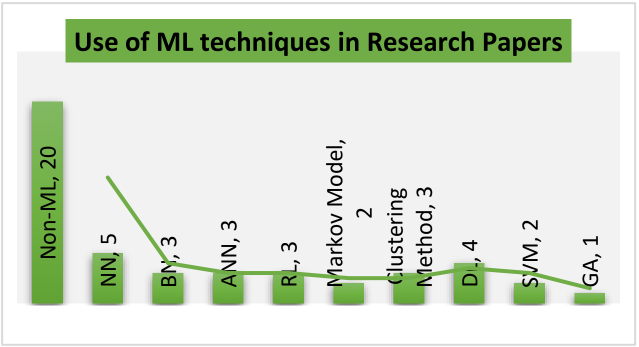
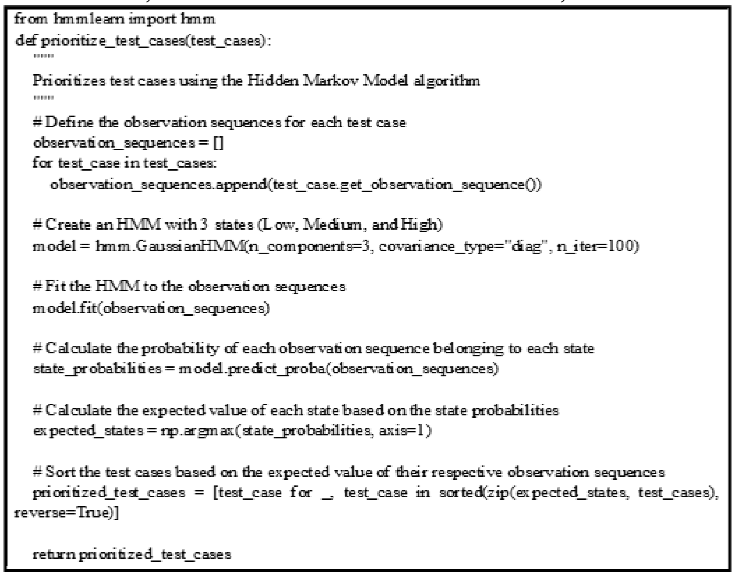


Fig.3: ML Technique used for TCP

Now, we may need to choose only those test cases/ suites [49][50] that provide the maximum code coverage in the quickest amount of time in subsequent testing rounds where time could be one of the limited constraints. To improve testing efficiency, Test Impact Analysis [51]-[56] is another optimization technique that is used to identify which tests execute a particular code and what time it has taken while executing that code. The snippet of a Hidden Markov Model approach in the following code figure 3 with three states (Low, Medium, and High) that can be used for test case prioritization of modelling [57].

In figure 4 code, the test_cases parameter is a list of test case objects. Each test case object has a get_observation_sequence() method that returns a sequence of observations from the execution of the test case. Observation sequence can be any measurable aspect of the test case, such as time taken to execute the test case, or



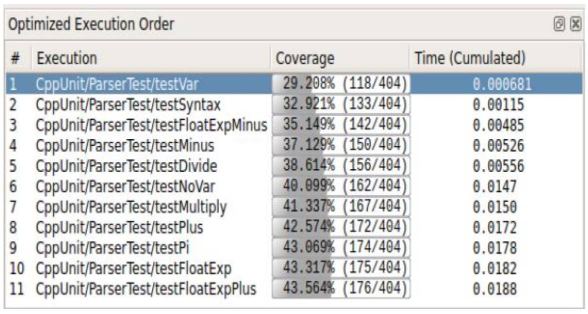
```
from hmmlearn import hmm
def prioritize_test_cases(test_cases):
    """
    Prioritizes test cases using the Hidden Markov Model algorithm
    """
    # Define the observation sequences for each test case
    observation_sequences = []
    for test_case in test_cases:
        observation_sequences.append(test_case.get_observation_sequence())

    # Create an HMM with 3 states (Low, Medium, and High)
    model = hmm.GaussianHMM(n_components=3, covariance_type="diag", n_iter=100)

    # Fit the HMM to the observation sequences
    model.fit(observation_sequences)

    # Calculate the probability of each observation sequence belonging to each state
    state_probabilities = model.predict_proba(observation_sequences)

    # Calculate the expected value of each state based on the state probabilities
    expected_states = np.argmax(state_probabilities, axis=1)

    # Sort the test cases based on the expected value of their respective observation sequences
    prioritized_test_cases = [test_case for _, test_case in sorted(zip(expected_states, test_cases),
reverse=True)]

    return prioritized_test_cases
```

Fig. 4: Snippet of HMM for three priorities states

number of defects found during execution. The code creates an HMM with three states (Low, Medium, and High) and fits the model to the observation sequences. It then calculates the probability of each observation sequence belonging to each state and calculates the expected value of each state based on the state probabilities. The test cases are then sorted based on the expected value of their respective observation sequences [58], with the regression test cases [59] that are expected to have the highest impact on the software application [60] appearing first in the sorted list.

After all this comparison, Markov Model would be able to generate test cases and prioritize test cases as per execution order and execution time mentioned in figure 5 with the help of evaluating aggregate code coverage and time taken while executing the test suites. For example, we have taken a sample of only for 200 test suites. Test suite 1 gives 29.208% code coverage with computation test time of 0.000681 seconds. Test suite 2, will be having both code coverage of test suite 1 and test suite 2 i.e., 32.921 with computation time of 0.00115 and so on. Generally, test cases are ordered based on the following priority levels:

- High priority: Test cases that cover functionality that is critical to the application.
- Medium priority: Test cases that cover functionality that is important, but not critical.
- Low priority: Test cases that cover functionality that is not important.



#	Execution	Coverage	Time (Cumulated)
1	CppUnit/ParserTest/testVar	29.208% (118/404)	0.000681
2	CppUnit/ParserTest/testSyntax	32.921% (133/404)	0.00115
3	CppUnit/ParserTest/testFloatExpMinus	35.149% (142/404)	0.00485
4	CppUnit/ParserTest/testMinus	37.129% (150/404)	0.00526
5	CppUnit/ParserTest/testDivide	38.614% (156/404)	0.00556
6	CppUnit/ParserTest/testNoVar	40.099% (162/404)	0.0147
7	CppUnit/ParserTest/testMultiply	41.337% (167/404)	0.0150
8	CppUnit/ParserTest/testPlus	42.574% (172/404)	0.0172
9	CppUnit/ParserTest/testPi	43.069% (174/404)	0.0178
10	CppUnit/ParserTest/testFloatExp	43.317% (175/404)	0.0182
11	CppUnit/ParserTest/testFloatExpPlus	43.564% (176/404)	0.0188

Fig. 5: Prioritization of Unit Test Cases

From 200 test cases, the ground truth priorities are as follows:

- 50 test cases are high priority.
- 100 test cases are medium priority.
- 50 test cases are low priority.

The prioritization model based on Hidden Markov predicts the following priorities:

- 40 high priority test cases
- 100 medium priority test cases
- 60 low priority test cases

Hence, Out of 200 test cases, total 40 test cases are classified as high priority test cases.

4. Discussions and Conclusions

Based on the comprehensive analysis, the paper suggests that Markov Chain Model is a useful machine learning technique employed for generating test cases that are most likely to uncover new bugs. By examining the transition probabilities between states in the model, it is possible to identify the test cases that are most likely to move from a state where no bugs have been found to a state where a bug has been discovered. This strategy has the potential to enhance the efficiency and efficacy of the testing procedure, enabling developers to promptly and precisely detect and rectify bugs. Although machine learning has shown great potential in improving the effectiveness and effi-ciency of test case prioritization. By training classifiers on large and diverse datasets, we can identify patterns and relationships between various factors that can influence the priority of test cases. This allows us to develop accurate and robust models that can prioritize test cases based on their relative importance. In addition to, Hidden Markov Models (HMM) have shown potential for test case prioritization by modeling the temporal behavior of software systems. By using HMM to capture the sequence of states that a system transitions through during execution, we can prioritize test cases based on their ability to trigger transitions between high-priority states.

One of the limitations is that HMM assumes a stationary and ergodic system, which may not hold for all software systems. Additionally, HMM may is not suitable for capturing complex or non-linear relationships between states in the system.

A delimitation of using HMM is the availability of sufficient data to train the model. HMM requires a large amount of data to accurately model the behavior of the software system, which may be a challenge for smaller or less mature systems. Additionally, the performance of HMM may be sensitive to changes in the system or the underlying data, which can affect the accuracy and reli-ability of the prioritization results.

Future work in this area can focus on addressing these limitations and delimitations and exploring new approaches to improve the effectiveness and efficiency of HMM-based TCP. Some potential areas for future research include:

- Developing methods for identifying and mitigating non-stationary and non-ergodic behavior in software systems.
- Exploring the application of alternative probabilistic frameworks, like Markov Decision Processes (MDP), for capturing intricate and non-linear interdependencies among system states.
- Investigating the use of domain-specific features and metrics that can capture the unique charac-teristics and requirements of different types of software systems and applications.
- Integrating HMM-based prioritization with other testing techniques such as regression testing, mutation testing, and fuzz testing to create more comprehensive and effective testing strategies.

Overall, the use of HMM for test case prioritization holds great promise for improving the quality and efficiency of software testing, particularly for systems that exhibit stationary and ergodic behavior. With continued research and development, we can expect to see more sophisticated and effective HMM-based prioritization models that can be applied to a wide range of software systems and applications.

Acknowledgements

We extend our heartfelt appreciation to the KIET Group of Institutions and Sangam University for their generous support, which made this work possible. Additionally, we are immensely grateful to the reviewers, colleagues, and guides who dedicated their time and effort to provide valuable and constructive feedback during the review process. Their insightful input has significantly contributed to enhancing the quality of this manuscript.

References

[1] Haghighatkhah, A., Mäntylä, M., Oivo, M., & Kuvaja, P. (2018). Test Prioritization in Continuous Integration, *Environments. J. Syst. Softw.*, 146, 80-98, https://doi.org/ 10.1016/ j.jss.2018.08.061.

[2] Noorian, M. Bagheri, E., & Du, W. (2011). Machine Learning-Based Software Testing: Towards a Classification Framework. *In SEKE*, pp. 225–229.

[3] Lou, Y., Chen, J., Zhang, L., & Hao, D. (2019). A Survey on Regression Test-Case Prioritization. *In Advances in Computers*; Elsevier, pp 1–46, https://doi.org/ 10.1016/zbs.adcom.2018.10.001

[4] Hajri, I., Goknil, A., Pastore, F., & Briand, L. C. (2020). Automating System Test Case Classification and Prioritization for Use Case-Driven Testing in Product Lines, *Empir. Softw. Eng.*, 25(5), 3711–3769. https://doi.org/ 10.1007/s10664-020-09853-4.

[5] Kim, J., Ryu, J. W., Shin, H. J., & Song, J. H. (2017). Machine Learning Frameworks for Automated Software Testing Tools: A Study. *International Journal of Contents*, 13(1), 38–44, https://doi.org/10.5392/IJoC.2017.13.1.038

[6] Nguyen, V & Le, B. RLTCP: A Reinforcement Learning Approach to Prioritizing Automated User Interface Tests. *Inf. Softw. Technol.* 2021, 136 (106574), 106574. https://doi.org/10.1016/j.infsof.2021.106574.

[7] Durelli, V. H. S., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R. C. & Guimaraes, M. P. (2019) Machine Learning Applied to Software Testing: A Systematic Mapping Study. *IEEE Trans. Reliab.*, 68(3), 1189–1212. https://doi.org/10.1109/tr.2019.2892517.

[8] Lousada, J. & Ribeiro, M. (2020) Reinforcement Learning for Test Case Prioritization. arXiv [cs.SE],. http://arxiv.org/abs/2012.11364.

[9] Li, Q. & Pham, H. Modeling Software Fault-Detection and Fault-Correction Processes by Considering the Dependencies between Fault Amounts. *Appl. Sci. (Basel)* 2021, 11(15), 6998. https://doi.org/10.3390/app11156998.

[10] Bagherzadeh, M., Kahani, N. & Briand, L. Reinforcement Learning for Test Case Prioritization. *IEEE Trans. Softw. Eng.* 2022, 48(8), 2836–2856. https://doi.org/10.1109/tse. 2021.3070549.

[11] Agarwal, D. Tamir, D. E. Last, M. Kandel, A. (2012). A Comparative Study of Artificial Neural Networks and Info-Fuzzy Networks as Automated Oracles in Software Testing, *IEEE Trans.*

Syst. Man Cybern. A Syst. Hum., 42(5), 1183–1193. https://doi.org/10.1109/tsmca.2012. 2183590.

[12] Path, O. C. (2019). Heuristic based Regression Test Case Prioritization Algorithm with Analysis for Test Cost Reduction and Optimized Cost Path Generation. *Journal of Theoretical and Applied Information Technology*, No. 17.

[13] Lachmann, R., Schulze, S., Nieke, M., Seidl, C., & Schaefer, I. (2016). System-Level Test Case Prioritization Using Machine Learning. In 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE., https://doi.org/10.1109/ICMLA.2016. 0065

[14] Zhang, W., Qi, Y., Zhang, X., Wei, B., Zhang, M., & Dou, Z. On Test Case Prioritization Using Ant Colony Optimization Algorithm. In 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th **International** Conference on Data Science and Systems (HPCC/SmartCity/DSS); IEEE, 2019. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00388

[15] Zhao, X., Wang, Z., Fan, X., & Wang, Z. (2015). A Clustering-Bayesian Network Based Approach for Test Case Prioritization. In 2015 IEEE 39th Annual Computer Software and Applications Conference; IEEE,. https:// doi.org/10.1109/COMPSAC.2015.154

[16] Mirarab, S., & Tahvildari, L. (2007). A Prioritization Approach for Software Test Cases Based on Bayesian Networks. *In Fundamental Approaches to Software Engineering; Springer Berlin Heidelberg: Berlin, Heidelberg*, pp. 276-290. https://doi.org/10.1007/978-3-540-71289-3_22

[17] Stallbaum, H., Metzger, A. & Pohl, K. (2008) An Automated Technique for Risk-Based Test Case Generation and Prioritization. In Proceedings of the 3rd international workshop on Automation of software test; ACM: New York, NY, USA, https://doi.org/10.1145/1370042.1370057

[18] Lousada, J., & Ribeiro, M. (2020). Neural Network Embeddings for Test Case Prioritization. arXiv [cs.SE], http://arxiv.org/abs/2012.10154.

[19] Meçe, E. K., Paci, H. & Binjaku, K. (2020). The Application of Machine Learning in Test Case Prioritization-a Review. *European Journal of Electrical Engineering and Computer Science*, 4(1), https://doi.org/10.24018/ejece.2020. 4.1.128

[20] Carlson, R., Do, H., & Denton, A. (2011). A Clustering Approach to Improving Test Case Prioritization: An Industrial Case Study. In 2011 27th IEEE International Conference on Software Maintenance (ICSM); IEEE., https://doi.org/10.1109/ICSM.2011.6080805

[21] Gupta, N., Sharma, A. & Pachariya, M. K. (2022). Multi-Objective Test Suite Optimization for Detection and Localization of Software Faults. *J. King Saud Univ. - Comput. Inf. Sci.*, 34(6), 2897–2909. https://doi.org/ 10.1016/j.jksuci. 2020.01.009.

[22] Cheruiyot, V. (2021). Machine Learning Driven Software Test Case Selection. University of Alberta Libraries. https://doi.org/10.7939/R3-M9D9-9861.

[23] Xiao, H., Cao, M., & Peng, R. (2020). Artificial Neural Network-Based Software Fault Detection and Correction Prediction Models Considering Testing Effort. *Applied Soft Computing*, 94, https://doi.org/10.1016/j.asoc.2020.106491

[24] Höstklint, N., & Larsson, J. (2021). Dynamic Test Case Selection Using Machine Learning.

[25] Lima, J. A. P.; Vergilio, S. R. (2022). A Multi-Armed Bandit Approach for Test Case Prioritization in Continuous Integration Environments. *IEEE Trans. Softw. Eng*., 48 (2), 453–465. https://doi.org/10.1109/tse.2020.2992428.

[26] Lachmann, R. (2017). Black-Box Test Case Selection and Prioritization for Software Variants and Versions, Universitätsbibliothek Braunschweig, https://doi.org/ 10.24355/dbbs.084-201711021237

[27] Busjaeger, B. & Xie, T. (2016). Learning for Test Prioritization: An Industrial Case Study. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering; ACM: New York, NY, USA, https://doi.org/10.1145/2950290.2983954

[28] Jahan, H., Feng, Z., Mahmud, S. M. H. & Dong, P. (2019). Version Specific Test Case Prioritization Approach Based on Artificial Neural Network. *J. Intell. Fuzzy Syst*. 2019, 36(6), 6181–6194, https://doi.org/10.3233/jifs-181998.

[29] Chen, J., Bai, Y., Hao, D., Xiong, Y., Zhang, H., Zhang, L., & Xie, B. Test Case Prioritization for Compilers: A Text-Vector Based Approach. In 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST); IEEE, 2016. https://doi.org/10.1109/ ICST.2016.19

[30] Kashyap, A. (2013). A Markov Chain and Likelihood-Based Model Approach for Automated Test Case Generation, Validation and Prioritization: Theory and Application (Doctoral Dissertation).

[31] Chen, J., Hu, W., Hao, D., Xiong, Y., Zhang, H., Zhang, L., & Xie, B. (2016). An Empirical Comparison of Compiler Testing Techniques. In Proceedings of the 38th International Conference on Software Engineering; ACM: New York, NY, USA, https://doi.org/10.1145/2884781.2884878

[32] Kalyani, R., Mounika, P.S., Naveen, R., Maridu, G., & Ramya, P. (2018). Test Case Prioritization Using Requirements Clustering. *International Journal of Applied Engineering Research*, 13(15), 11776–11780.

[33] Chen, J., Bai, Y., Hao, D., Xiong, Y., Zhang, H. & Xie, B. (2017). Learning to Prioritize Test Programs for Compiler Testing. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE); IEEE, https://doi.org/ 10.1109/ICSE.2017.70

[34] Spieker, H. Gotlieb, A. Marijan, D & Mossige, M. (2017). Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis; ACM: New York, NY, USA, https://doi.org/10.1145/3092703.3092709

[35] Tonella, P., Avesani, P., & Susi, A. Using the Case-Based Ranking Methodology for Test Case Prioritization. In 2006 22nd IEEE International Conference on Software Maintenance; IEEE, 2006. https://doi.org/10.1109/ICSM. 2006.74

[36] Konsaard, P., & Ramingwong, L. (2015). Total Coverage Based Regression Test Case Prioritization Using Genetic Algorithm. In 2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON); IEEE, https://doi.org/10.1109/ECTICon.2015.7207103

[37] Improving Proceeding Test Case Prioritization with Learning Software Agents. (2014). In Proceedings of the 6th International Conference on Agents and Artificial Intelligence; SCITEPRESS - Science and and Technology Publications, https://doi.org/10.5220/0004920002930298

[38] Lachmann, R. (2018). 12.4 - Machine Learning-Driven Test Case Prioritization Approaches for Black-Box Software Testing. In Proceeding - ettc2018; AMA Service GmbH, Von-Münchhausen-Str. 49, 31515 Wunstorf, Germany, https://doi.org/10.5162/ettc2018/12.4

[39] Sharif, A., Marijan, D., & Liaaen, M. (2021). DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing. In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME); IEEE, https://doi.org/10.1109/ICSME52107.2021.00053

[40] Matcha, W., Touré, F., Badri, M., & Badri, L. (2020). Using Deep Learning Classifiers to Identify Candidate Classes for Unit Testing in Object-Oriented Systems. *In SEKE*, pp 353–358.

[41] Medhat, N., Moussa, S. M., Badr, N. L., Tolba, M. F. A. (2020). Framework for Continuous Regression and Integration Testing in IoT Systems Based on Deep Learning and Search-Based Techniques. *IEEE Access*, 8, 215716–215726. https://doi.org/10.1109/access.2020.3039 931.

[42] Landin, C., Hatvani, L., Tahvili, S., Haggren, H., Längkvist, M., Loutfi, A., & Håkansson, A. (2020). Performance Comparison of Two Deep Learning Algorithms in Detecting Similarities between Manual Integration Test Cases. In The Fifteenth International Conference on Software Engineering Advances (ICSEA 2020); Porto, Portugal, pp. 90–97.

[43] Chen, Y., Wang, Z., Wang, D., Yao, Y., & Chen, Z. (2019). Behavior Pattern-Driven Test Case Selection for Deep Neural Networks. In 2019 IEEE International Conference On Artificial Intelligence Testing (AITest); IEEE, https://doi. org/10.1109/AITest.2019.000-2

[44] Omri, S., & Sinz, C. (2021). Machine Learning Techniques for Software Quality Assurance: A Survey. arXiv [cs.SE],. http://arxiv.org/abs/2104.14056.

[45] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (1999). Test Case Prioritization: An Empirical Study. In Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). "Software Maintenance for Business Change" (Cat. No.99CB36360); IEEE, https://doi.org /10.1109/ICSM.1999.792604

[46] Gökçe, N., Eminov, M., & Belli, F. Coverage-Based, Prioritized Testing Using Neural Network Clustering. *In Computer and Information Sciences – ISCIS 2006; Springer Berlin Heidelberg: Berlin*, Heidelberg, 2006, pp. 1060–1071. https://doi.org/10.1007/11902140_110

[47] Gökçe, N., Belli, F., Eminli, M., & Dinçer, B. T. (2015). Model-Based Test Case Prioritization Using Cluster Analysis: A Soft-Computing Approach. *TURK. J. OF ELECTR. ENG. COMPUT. SCI*. 23, 623–640. https://doi.org/10.3906/elk-1209-109.

[48] Koochakzadeh, N. & Garousi, V. A (2010). Tester-Assisted Methodology for Test Redundancy Detection. A*dv. Softw. Eng.*, pp. 1–13. https://doi.org/10.1155/2010/ 932686.

[49] Singh, Y., Kaur, A., & Suri, B. (2010). Test Case Prioritization Using Ant Colony Optimization. *Softw. Eng. Notes,* 35(4), 1–7. https://doi.org/10.1145/1811226. 1811238.

[50] Paramshetti, P., & Phalke, D. A. (2014). Survey on Software Defect Prediction Using Machine Learning Techniques. *International Journal of Science and Research,* 3(12), 1394–1397.

[51] Gokce, N. & Eminli, M. (2014). Model-Based Test Case Prioritization Using Neural Network Classification. *Comput. Sci. Eng. Int. J*., 4 (1), 15–25. https://doi.org/ 10.5121/cseij.2014.4102.

[52] Ribeiro, M., Grolinger, K., Capretz, M. A. M. (2015). MLaaS: Machine Learning as a Service. In 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA); IEEE, https://doi.org/10.1109/ ICMLA.2015.152

[53] Jiang, B., & Chan, W. K. (2015). Input-Based Adaptive Randomized Test Case Prioritization: A Local Beam Search Approach. *J. Syst. Softw*., 105, 91–106. https://doi.org/ 10.1016/ j.jss.2015.03.066.

[54] Yan, R., Chen, Y., Gao, H., & Yan, J. (2022). Test Case Prioritization with Neuron Valuation-Based Pattern. Science of Computer Programming, 215. https://doi.org/10.1016/ j.scico.2021.102761

[55] Korel, B., & Koutsogiannakis, G. (2009). Experimental Comparison of Code-Based and Model-Based Test Prioritization. In 2009 International Conference on Software Testing, Verification, and Validation Workshops; IEEE, https://doi.org/10.1109/ICSTW.2009.45

[56] Mirarab, S. & Tahvildari, L. (2008). An Empirical Study on Bayesian Network-Based Approach for Test Case Prioritization. In 2008 International Conference on Software Testing, Verification, and Validation; IEEE,. https:// doi.org/10.1109/ICST.2008.57

[57] 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.* 30(02), 263–290. https://doi.org/10.1142/s021819402050 0126.

[58] Chen, J., Zhu, L., Chen, T. Y., Huang, R., Towey, D., Kuo, F.-C., & Guo, Y. An Adaptive Sequence Approach for OOS Test Case Prioritization. In 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW); IEEE, 2016. https://doi.org/10 .1109/ ISSREW. 2016.29

[59] Muthusamy, T. K. S. (2014). Effectiveness of Test Case Prioritization Techniques Based on Regression Testing. *Int. J. Softw. Eng. Appl.,* 5 (6), 113–123. https://doi.org/ 10.5121/ijsea.2014.5608.

[60] King, T. M., Arbon, J., Santiago, D., Adamo, D., Chin, W., & Shanmugam, R. (2019). AI for Testing Today and Tomorrow: Industry Perspectives. In 2019 IEEE International Conference on Artificial Intelligence Testing (AITest); IEEE,. https://doi.org/10.1109/AITest.2019.000-3

[61] Rajora , K., & abdulhussein , N. salih. (2023). Reviews research on applying machine learning techniques to reduce false positives for network intrusion detection systems. Babylonian Journal of Machine Learning, 2023, 26–30. https://doi.org/10.58496/BJML/2023/005

[62] Alsajri, A., & Steiti, A. (2023). Intrusion Detection System Based on Machine Learning Algorithms:(SVM and Genetic Algorithm). Babylonian Journal of Machine Learning, 2024, 15–29. https://doi.org/10.58496/BJML/2024/002

[63] Alsajri, A. (2023). A Review on Machine Learning Strategies for Real-World Engineering Applications. Babylonian Journal of Machine Learning, 2023, 1–6. https://doi.org/10.58496/BJML/2023/001

[64] T. Bhaskar, S. A. Shiney, S. B. Rani, K. Maheswari, S. Ray and V. Mohanavel, "Usage of Ensemble Regression Technique for Product Price Prediction," 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2022, pp. 1439-1445, doi: 10.1109/ICIRCA54612.2022.9985521.

[65] Saravanan, P., Aparna Pandey, Kapil Joshi, Ruchika Rondon, Jonnadula Narasimharao, and Afsha Akkalkot Imran. "Using machine learning principles, the classification method for face spoof detection in artificial neural networks." In *2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pp. 2784-2788. IEEE, 2023.