Journal of Theoretical and Applied Information Technology
15th February 2023. Vol.101. No 3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AUTOMATION OF TEST CASE PRIORITIZATION: A SYSTEMATIC LITERATURE REVIEW AND CURRENT TRENDS

JIJO JOSEPH C GEORGE[1], D. PETER AUGUSTINE[2]

[1]Department of Computer Science, CHRIST (Deemed to be University), India
[2]Department of Computer Science, CHRIST (Deemed to be University), India
E-mail: [1]jijo.joseph@res.christunversity.in, [2]peter.augustine@christuniversity.in

ABSTRACT

An Important stage in software testing is designing a test suite [18]. The test case repository consists of a large number of test cases. However, only a portion of these test cases would be relevant and can find bugs. Test case prioritization(TCP) is one such technique that can substantially increase the cost-effectiveness of the testing activity. Using test case prioritization, more relevant test cases can be captured and tested in the earlier stages of the testing phase. The objective of the study is to understand different techniques used and a systemic study on the effectiveness of these approaches. The Literature consists of a few relevant articles introducing novel techniques for test case prioritization between 2008 and 2022. Studies show that parameters that are considered for test case prioritization are important. Hence, the current article also focuses on the parameters considered in the literature. 40% of the articles used in the literature review use different test case information as parameters. A systemic review and analysis of data sets involved in the literature are evaluated in the study. The review also focuses on the different approaches used for comparing the newly introduced approach and reveals a novel approach for prioritization.

Keywords :– *Software Testing, Software Test Automation, Software Engineering, Test Case Prioritization, Regression Testing*

1. INTRODUCTION

Quality of the software is significant as software shoulders a critical role. The cost of a single defect could be enormous. Due to this reason, the quality of the software is significant. Testing is a method through which we can assure quality. Different types of testing like sanity testing, regression testing, unit testing, smoke testing, etc. are conducted depending on multiple factors like duration of testing, coverage, the phase at which testing is conducted, etc.

Regression testing is performed when a new version of an existing software is released. This includes bug fixes from previous versions and additional features in the software [15]. Regression testing is essential to ensure that the new version of the software has not caused any bugs. A large number of test cases covering different scenarios would be available in the regression test bucket. On the contrary, only a few of these test cases would be relevant [20]. Given the limited time and resources, it would be difficult to execute all the test cases available in the test case bucket. Hence, it is important to choose relevant test cases for execution which will reduce the execution cost significantly and improve testing efficiency [23]. One solution for this problem would be test case prioritization. Using effective prioritization techniques, the total number of tests case can be drastically reduced without sacrificing the quality of the software. Realizing the significance, prioritization technology has gained importance in software industries [21]. Understanding the relevance lot of research has already gone into different prioritization techniques. Using different techniques, parameters, comparison techniques, and datasets, there are different types of test case prioritization techniques available today. Due to this, uniqueness is visible in almost all studies, it is very difficult to determine which technique is most suitable for a given scenario. The current paper is a scientific study of different techniques that are currently available to understand the scenarios where these techniques can be used. For this purpose, we are looking at a few most relevant papers published in the recent past which are unique. Uniqueness could be observed in the software tested, the approach used, the algorithm, parameters selected, the data set available, strategies used for comparison, etc.

2. LITERATURE REVIEW

In a given test suite T, the objective of Test case prioritization is to reorder the test cases in such a way that the probability of hitting a bug in the initial stages would be higher [18]. This will help the engineers spot the bugs sooner and have sufficient time to fix them before the software is released. TCP can be combined with regression test selection (RTS) to generate a subset of T. This subset will contain only prioritized relevant test cases that are prioritized. This will help minimize the test duration required for testing without sacrificing the quality.

Several approaches exist to automate TCP. Over the years, various approaches have evolved with the common objective of increasing quality. For this study, the few most relevant articles from the recent past were selected. These articles include unique approaches to reach the common objective of prioritizing test cases. In this section, we are briefly analyzing these approaches.

Miairab et al. [1] (2008) have implemented TCP using a Bayesian network (BA). Where the parameters considered include the average percentage of faults detected, running duration of test cases, feedback mechanism using BA, source code differences using Unix diff, and source code similarities using constant pool difference. The Data set includes open-source programs namely ant, Jmeter, xml, nanoxml, and galileo. The system was compared using different BA algorithms like BA without feedback with Unix diff (D), BA with feedback with D, BA without feedback with Constant pool diff (CP), and BA with feedback with CP. Carlson et al. [2] (2011) used a clustering algorithm. Parameters used include Code coverage, Code complexity, and Fault history information. The data set comprises Microsoft Dynamics Ax 2009 including the initial release and SP1 release. Current TCP was compared against all combinations of parameters with and without clustering. Frang Wang et al. [3] (2011) use neural networks and slicing techniques to prioritize test cases. Parameters used include execution traces and source code changes. The efficiency was compared by testing all test cases, randomly picking test cases, and using neural networks with and without slicing techniques. Md. Junaid et al. [4] (2013) use k-means clustering to prioritize. The parameters include term extraction and requirement term-document matrix to create a requirement cluster, Lines of code, Nested block depth, and McCabe Cyclomatic Complexity. The data set include two college student project namely Capstone and iTrust. The results were compared against the original and random cluster order with and without code metric. Using hamming distance to analyze the changes in the code, and test case failure history as parameters, Pang et al. [5] (2013) uses k-means clustering to segregate test cases to execute. A comparison was made by running the solution on different versions of datasets. The dataset includes nanoxml, jtopas, JMeter, xml-security, and ant. Di Nardo et al. [6] (2013) using coverage-based strategies have implemented TCP. The parameter consists of code coverage. The dataset used for the study was an industrial software named Noisegen. A comparison was made using function entry, function return, block, basic block, and decision. Chaurasia,

Geetanjali et al. [7] (2015) use clustering to prioritize test cases. The parameters include Code coverage, fault history, Source code info, Execution time, and test-case execution history. The data set consists of two open-source software called Jmeter and Apache ANT. The results were compared using other TCP methods like Untreated, Random, function-total, function-additional, agglomerative hierarchical clustering, and K-means clustering. Saha et al. [8] (2015) use information retrieval (IR) for TCP. Parameters include different levels of code changes between versions. A comparison was made using different TCP strategies like Untreated test prioritization, Random test prioritization, Method coverage using the Additional strategy, Method coverage using the total strategy, Statement coverage using the additional strategy, Statement coverage using the total strategy, and JUPTA. The data set used for the study includes open-source programs namely Time and Money, Mime4J, Jaxen, XML-Security, XStream, Commons-Lang, Joda-Time, and Apache Ant. Noor et al. [9] (2015) used a similarity-based approach for a solution for TCP. The parameters used for this study include Test case history and code changes. A Comparison was made with other approaches like Hamming distance, the traditional history-based approach, Edit distance, basic counting, and improved basic counting. The data set used for this study includes 5 Java programs: Commons Lang, JodaTime, JFreeChart, Commons Math, and Closure Compiler.

Using the SVM rank algorithm, Busjaeger et al. [10] (2016) implemented TCP. Parameters used for this study include text content similarity, text path similarity, failure history, Java code coverage, test age, and test case list. Automation live software Salesforce was used as part of the data set constituting 45000 test cases. A comparison was made with other TCP approaches including random, coverage, text path, text content, History, and Test age. Lachmann et al. [11] (2016) used SVM rank

algorithm for TCP. Parameters include test case description, requirements coverage, failure count, failure age, failure priority, and execution costs. The data set used includes one internal project named Body comfort system and automotive industrial data. A comparison was made using SVM rank with and without test case description, and random. Chen et al. [12] (2016) used a clustering algorithm. Test case information is the only parameter used for this study. Data sets include programs like CCoinBox, WindShieldWiper, SATM, RabbitAndFox, and WaveletLibrary. A comparison was made using other TCP approaches including MoClustering_means Algorithm, MOClustering – medoid, and random testing with method sequence.

Using Linux utilities like cal, comm, look, spline, and unique and Siemens Suite from Software-artifact Infrastructure Repository (SIR) as data set Zhang et al. [13] (2016) prioritized test cases using Adaptive random sequence algorithm (ARS). The parameter includes test execution history. The results were compared using ARS-pass, ARS-all, and Ledru algorithms. Using differences found in 2 subsequent versions of software, Panda et.al [14] (2016) solve the TCP problem with the help of a slice-based approach. A comparison was made using a previous study by [19].

Lachmann et al. [15] (2018) used different machine-learning approaches to understand a suitable solution for TCP. Parameters include Test case description (natural language), Test case age, Number of linked requirements, Number of linked defects (history), Severity of linked defects, Test case execution cost (time), and Project-specific features (e.g., market). The data set includes a test suit from an automotive industry and an academic project. Spieker et al. [16] (2018) use reinforcement learning as a solution to the TCP problem. Parameters include test case history. The data set used for this study are paint control, IOF/ROL, and Google Shared dataset of Test Suite Results (GSDTSR). Results were compared using test case failure reward, test case count reward, and time reward.

Pradhan et al. [17] (2019) use REMAP, a combination of rule mining and multi-objective search algorithms to evaluate TCP. The parameter for the study includes test case history. The data set used for this study are VCS products from Cisco, open source case studies from ABB Robotics for Paint Control, IOF/ROL, and "Google Shared Dataset of Test Suite Results (GSDTSR)". Yu, Zhe, et al. formulated a novel TCP technique using the total recall algorithm. Parameters consist of execution history, test case description, and feedback from previous runs. With the help of the LexisNexis database, they have compared the solution with random, history-based, cost-based, test case description-based, and feedback-based. Using the tag-based IR algorithm Azizi, Maral proposed a TCP technique using source code changes and the details from the test case as parameters. The data set consists of open-source tools namely nopCommerce, Umbraco-CMS, Joda-Time, JFreeChart, Commons-Lang, and XStream. A comparison was made using untreated, random, total Statement, additional Statement, and text Retrieval Based. Chen, Jinfu, et al. used clustering-based random sequencing for TCP using the parameters Test case pool (TCP), Number of test cases to be selected from TCP, and Number of clusters to be generated. A comparison was made using MOClustering_mean, MOClustering_medoids, DMClustering, method coverage, and random sequencing. The Data set consisted of open-source software which includes CCoinBox, WindShieldWiper, SATM, RabbitsAndFoxes, WaveletLibrary, IceChat, and CSPspEmu.ss. Using test sequence and code coverage as parameters, Chi, Jianlei, et al. has demonstrated the impact of relation-based algorithms on TCP. The dataset consisted of open-source programs Commons.lang, Jodatime, Log4j-core, Commons.math, Jfreechart, Ant, Commons.math3, Google Closure Compiler, and Average. A comparison was made using different search-based, greedy, adaptive algorithms, and natural order. Di Nucci, Dario, et al. uses a hypervolume-based genetic algorithm for TCP. Parameters include execution cost, statement coverage, and past fault coverage. The dataset includes GNU utilities namely Bash, Flex, Grep, GZip, and Sed. A Comparison was made using GDE3 [24] and MOEA/D-DE [25]. Bagherzadeh, Mojtaba, et al. analyze different reinforcement learning algorithms in their study. The parameter includes test case details. The data set used includes Paint, IOFROL Simple, Codec Enriched, Compress Enriched, Imaging Enriched, IO Enriched, Lang Enriched, and Math. A Comparison was made using different existing reinforcement learning algorithms.

3. FINDINGS

In this article, we are focusing on some important and most common attributes used in each of the TCP articles. These attributes include the algorithm used in the article, the parameters considered, the type of data set used, similar algorithms used for comparison, and the research

question asked in each article. Though most articles have used the Average Percentage of Faults Detected (APFD) metric introduced by Rothermel et al [22]. As a standard metric, we haven't compared the results as most of the articles have formulated their problem and solution in favor of the attributes discussed above. Comparing these results could be biased. The frequency of articles discussing TCP has increased over the years. Also, considering relatively newer articles will give us the latest trends used to prioritize test cases. Bearing these points in mind, we have considered more articles from recent years. Figure 1 is a representation of the year these articles were published.

The literature shows that each article is based on a particular algorithm or model. Most of the articles in the study use more than one algorithm to define the system. However, those articles play a subsidiary role in the formulation of the approach. The following figure is a representation of the primary algorithms used in the articles discussed in the literature review. If we look closely at the literature review we can see the articles have different algorithms under a common domain. For simplicity, we have broadly classified the algorithms under four major domains.

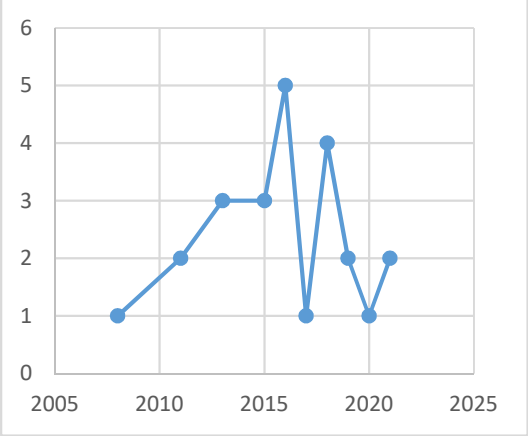

Figure 1: Published year

This includes IR, Machine learning (ML), clustering, and others. Close to 50% of articles involve some kind of ML algorithm. Over the years the popularity of ML has increased due to its accuracy in finding relevant test cases and simplicity. IR is the second most used algorithm with 27%, followed by clustering with 20%. Others include articles that do not include algorithms that are usually not used in solving TCP problems. In our case, these algorithms include rule mining, graph-based, genetic algorithm, relation-based, and multi-objective search algorithms.

Parameters are lists of dependent properties that impact the prioritization. Each article considers a set of variables that have the potential to define the prioritization list. Data has a significant role in defining the parameters used for the study. In some cases only the test suit is available. In such scenarios, it is difficult to find an optimal solution. In most of the articles in the literature review, Test case details like execution history, creation date, past test execution results, test case status, etc., are available. In these cases, ML algorithms perform well. On the other hand in a few articles, data including different versions of the program under test are available. In those scenarios using IR is preferable. These parameters play a key role in the result. Figure 3 is a broad classification of the parameters used in the literature. For simplicity, similar parameters are grouped under a single category. Table 1 is a list of parameters considered in each category. Data sets consist of different versions of the software under test. Industrial data sets, open-source data sets, and academic-related projects are three major categories of data sets used for testing the approaches. Each data set has its significance. Industrial data sets can capture real-world issues.

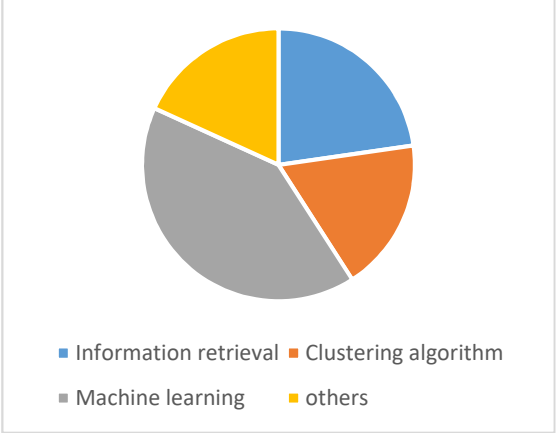

Figure 2: Algorithms

However, obtaining industrial data sets and testing is generally difficult. Academic and open-source data sets are easily accessible. One advantage of academic data set over an open-source data set is that; in academics, we can add any number of new parameters. Many articles consider a combination of two of the three categories.

Table 1: Categories

Others	Cost	Source code	Test case	Coverage
The average percentage of faults detected	Execution costs	Source code changes using Unix diff	Fault history information	Code coverage
Feedback mechanism using Bayesian network	Code complexity	Source code consistencies using Constant Pool diff	Execution traces	Requirements coverage
test case list	Execution time	Source code info	Number of linked defects	Number of linked requirements
Project-specific features (e.g., market)		text content similarity	The severity of linked defects	Test case coverage
		text path similarity	test age	statement coverage
		Java code coverage	failure count	past fault coverage
		Difference between the current version and the previous version	failure age failure priority	

61% of articles considered in the literature use open-source data set. Followed by industrial data set with 23%. The least used type is academic-related data set with just 14%. To verify the results obtained by a new approach, it is essential to compare results with other similar approaches. This serves as evidence for the claim that a particular TCP technique outperforms others. The two methods usually seen in the literature are compared with different algorithms and the same algorithm but different parameters. Generally, in most articles, APFD is used as the standard scale of measure. One approach most articles commonly used for comparison is random search. Other important comparison algorithms include; testing all the test cases, Logistic regression, neural, Ensemble, SVM, SVM rank, hamming distance, clustering, Distance equations, text path, History based, etc. Algorithms used for comparison help us understand alternative approaches that can be used in a particular scenario. Hence, is important to understand the comparison techniques used in the literature.

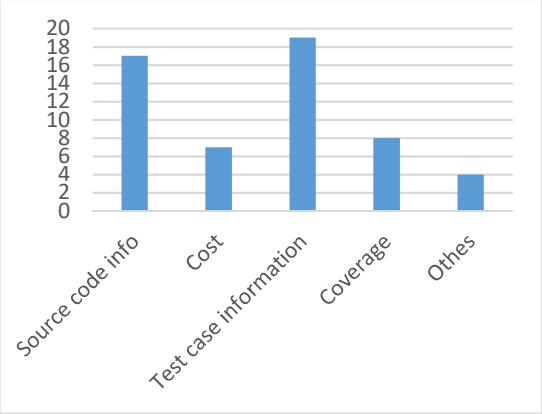

Figure 3: Parameters

4. SCOPE OF FUTURE RESEARCH

The articles used for literature were obtained by searching google scholar for "prioritization automation", and "prioritization techniques". Relevant articles which define the properties discussed in the current article were cherry-picked. Certainly, all alternative approaches, parameters, algorithms, and comparison techniques are discussed in the current article. However, this article helps us get an idea of the different properties we need to look at closely when we are defining a TCP approach. The findings clearly show that each article has an algorithm, data set used, method, parameters, and comparison in common. Further research must focus on defining all possible parameters and algorithms we can use for TCP. This defined list can be prepared with an extensive

literature review. The user can then define the weightage of each parameter according to the available data. Using this data we can decide on the algorithm or approach we can use for a particular problem. Table 2 is a representation demonstrating the system. The table is a sample parameter weightage list. Here all the parameters related to the source code are assigned 0. This is because the sample data set did not have source code information.

Table 2: Parameter weightage

Parameter	Weightage
text content similarity	0
text path similarity	0
failure history	3
Code coverage	0
test age	1
fault history	2
Source code info	0
Execution time	5
test-case execution history	5
Code differents between 2 versions	0
code complexity	0
Requirement information	0

5. CONCLUSION

The article discussed the different algorithms used, the importance of choosing the right parameter, different types of data sets, and algorithms used for comparison. We also discussed the current trends in algorithms. Currently, we do not see prioritization automation in the mainstream because of the complexities, overheads, and inaccurate results. Though research has proven the effectiveness of TCP, most industries are still not willing to adopt TCP. This is due to the risk and time involved to create a TCP solution. However, we have found significant growth in research dealing with different TCP approaches. As discussed above, a repository that defines all possible parameters and choosing the algorithms according to the parameters and available data set could simplify the problem significantly. With further research and fine-tuning, prioritization automation has scope in the future. The current study has thrown light on different aspects which need further research. We understood the data that must be considered before prioritization, and the multiple drawbacks of the available prioritization methodologies. In future work, with the help of further literature review, we are planning to develop a system similar to the proposed solution and test with industrial data set with the aim to simplify the TCP problem.

REFERENCES

[1] Mirarab, Siavash, and Ladan Tahvildari. "An empirical study on bayesian network-based approach for test case prioritization." *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 2008.

[2] Carlson, Ryan, Hyunsook Do, and Anne Denton. "A clustering approach to improving test case prioritization: An industrial case study." *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011.

[3] Wang, Farn, Shun-Ching Yang, and Ya-Lan Yang. "Regression testing based on neural networks and program slicing techniques." *Practical Applications of Intelligent Systems*. Springer, Berlin, Heidelberg, 2011. 409-418.

[4] Arafeen, Md Junaid, and Hyunsook Do. "Test case prioritization using requirements-based clustering." *2013 IEEE sixth international conference on software testing, verification and validation*. IEEE, 2013.

[5] Pang, Yulei, Xiaozhen Xue, and Akbar Siami Namin. "Identifying effective test cases through k-means clustering for enhancing regression testing." *2013 12th International Conference on Machine Learning and Applications*. Vol. 2. IEEE, 2013.

[6] Di Nardo, Daniel, et al. "Coverage-based test case prioritisation: An industrial case study." *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, 2013.

[7] Chaurasia, Geetanjali, Sonali Agarwal, and Swarnima Singh Gautam. "Clustering based novel test case prioritization technique." *2015 IEEE Students Conference on Engineering and Systems (SCES)*. IEEE, 2015.

[8] Saha, Ripon K., et al. "An information retrieval approach for regression test prioritization based on program changes." *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE, 2015.

[9] Noor, Tanzeem Bin, and Hadi Hemmati. "A similarity-based approach for test case prioritization using historical failure data." *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2015.

[10] Busjaeger, Benjamin, and Tao Xie. "Learning for test prioritization: an industrial case study." *Proceedings of the 2016 24th ACM SIGSOFT International symposium on foundations of software engineering*. 2016.

[11] Lachmann, Remo, et al. "System-level test case prioritization using machine learning." *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2016.

[12] Chen, Jinfu, et al. "An adaptive sequence approach for OOS test case prioritization." *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2016.

[13] Zhang, Xiaofang, Xiaoyuan Xie, and Tsong Yueh Chen. "Test case prioritization using adaptive random sequence with category-partition-based distance." *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2016.

[14] Panda, Subhrakanta, Dishant Munjal, and Durga Prasad Mohapatra. "A slice-based change impact analysis for regression test case prioritization of object-oriented programs." Advances in Software Engineering 2016 (2016).

[15] Lachmann, Remo. "Machine learning-driven test case prioritization approaches for black-box software testing." *The European test and telemetry conference, Nuremberg, Germany*. 2018.

[16] Spieker, Helge, et al. "Reinforcement learning for automatic test case prioritization and selection in continuous integration." *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2017.

[17] Pradhan, Dipesh, et al. "Employing rule mining and multi-objective search for dynamic test case prioritization." *Journal of Systems and Software* 153 (2019): 86-104.

[18] de Castro-Cabrera, M. del Carmen, Antonio García-Dominguez, and Inmaculada Medina-Bulo. "Trends in prioritization of test cases: 2017-2019." *Proceedings of the 35th annual acm symposium on applied computing*. 2020.

[19] Panigrahi, Chhabi Rani, and Rajib Mall. "A heuristic-based regression test case prioritization approach for object-oriented programs." *Innovations in Systems and Software Engineering* 10.3 (2014): 155-163.

[20] Broekman, Bart, and Edwin Notenboom. *Testing embedded software*. Pearson Education, 2003.

[21] Zhang, Weixiang, Bo Wei, and Huisen Du. "Test case prioritization based on genetic algorithm and s-points coverage." *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, Cham, 2014.

[22] Rothermel, Gregg, et al. "Prioritizing test cases for regression testing." *IEEE Transactions on software engineering* 27.10 (2001): 929-948.

[23] Lenz, Alexandre Rafael, Aurora Pozo, and Silvia Regina Vergilio. "Linking software testing results with a machine learning approach." *Engineering Applications of Artificial Intelligence* 26.5-6 (2013): 1631-1640.

[24] Kukkonen, Saku, and Jouni Lampinen. "GDE3: The third evolution step of generalized differential evolution." 2005 IEEE congress on evolutionary computation. Vol. 1. IEEE, 2005.

[25] Li, Hui, and Qingfu Zhang. "Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II." IEEE transactions on evolutionary computation 13.2 (2008): 284-302.

[26] Chen, Jinfu, et al. "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering." Journal of Systems and Software 135 (2018): 107-125.

[27] Azizi, Maral, and Hyunsook Do. "Graphite: A greedy graph-based technique for regression test case prioritization." 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2018.

[28] Di Nucci, Dario, et al. "A test case prioritization genetic algorithm guided by the hypervolume indicator." IEEE Transactions on Software Engineering 46.6 (2018): 674-696.

[29] Yu, Zhe, et al. "Terminator: Better automated ui test case prioritization." Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019.

[30] Chi, Jianlei, et al. "Relation-based test case prioritization for regression testing." Journal of Systems and Software 163 (2020): 110539.

[31] Azizi, Maral. "A tag-based recommender system for regression test case prioritization." 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2021.

[32] Bagherzadeh, Mojtaba, Nafiseh Kahani, and Lionel Briand. "Reinforcement learning for test case prioritization." IEEE Transactions on Software Engineering (2021).