2024, 21(2 Special Issue): 0609-0621
https://doi.org/10.21123/bsj.2024.9604
P-ISSN: 2078-8665 - E-ISSN: 2411-7986

Baghdad Science Journal

An Exploratory Study of History-based Test Case Prioritization Techniques on Different Datasets

S. M. Junaid H.[1,2], ***Dayang N. A. Jawawi**[1], ***Johanna Ahmad**[1]

[1]Faculty of Computing, Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia.
[2]Department of Information Technology, Faculty of Information & Communication Technology (FICT), Balochistan University of Information Technology, Engineering, and Management Sciences (BUITEMS), Quetta, Pakistan.
*Corresponding Author.
PARS2023: Postgraduate Annual Research Seminars 2023.

Received 22/09/2023, Revised 10/02/2024, Accepted 12/02/2024, Published 25/02/2024

Abstract

In regression testing, Test case prioritization (TCP) is a technique to arrange all the available test cases. TCP techniques can improve fault detection performance which is measured by the average percentage of fault detection (APFD). History-based TCP is one of the TCP techniques that consider the history of past data to prioritize test cases. The issue of equal priority allocation to test cases is a common problem for most TCP techniques. However, this problem has not been explored in history-based TCP techniques. To solve this problem in regression testing, most of the researchers resort to random sorting of test cases. This study aims to investigate equal priority in history-based TCP techniques. The first objective is to implement different history-based TCP techniques. The second objective is to explore the problem of equal priority in history-based TCP techniques. The third objective is to explore random sorting as a solution to the problem of equal priority in history-based TCP techniques. Datasets of historical records of test cases from conventional and modern sources were collected. History-based TCP techniques were applied to different datasets. The History-based TCP techniques were investigated for the problem of equal priority. Then random sorting was explored as a solution to the problem of equal priority. Finally, the results were elaborated in terms of APFD and execution time. The results indicate that history-based techniques also suffer from the problem of equal priority like other types of TCP techniques. Secondly, random sorting does not produce optimal results while trying to solve the problem of equal priority in history-based TCP. Furthermore, random sorting deteriorates the results of history-based TCP techniques when employed to solve the problem of equal priority. One should resort to random sorting if no other solution exists. The decision to choose the best solution requires a cost-benefit analysis of context and solutions at hand.

Keywords: Average Percentage of Fault Detected, Equal Priority, History Based, Random, Regression Testing, Test Case Prioritization.

Introduction

The software development life cycle consists of many stages but testing is one of the important phases of the software development lifecycle. Software testing comprises different activities but test case generation and optimization are at the forefront[1]. Test case optimization can be further divided into 3 activities TCP, test case selection (TCS), and test case minimization (TCM). TCP amongst its other counterparts has proven to be more useful as it does not compromise the test cases, while TCS selects some test cases from among the available set of test cases according to some objective, which might miss some important test cases and in TCM it reduces the test suit to a minimal number of test cases required which may compromise the ability of test cases to detect faults in future as the test cases are permanently removed[2-4].

There are different techniques used to prioritize test cases. These techniques use different data, procedures, and factors such as coverage data, requirements of the software, development history of software, search algorithms, and the similarity between test cases[2]. Coverage-based approaches work on the principle of how much code is covered by a test case, but it does not always guarantee good performance[5,6]. Requirements-based approaches can be employed in the initial phases because they use user requirements that are available in the beginning[7]. Similarly, historical approaches use past data from software development which is analogous to stock market prediction where the future can be predicted with the help of past data. It can also be used in the early phases. In Search-based techniques, algorithms can be used to search for the best solutions according to the provided constraints. Similarity-based approaches use **c**ommonness among test cases and code to prioritize test cases[2]. TCP techniques suffer from different problems such as equal priority, not achieving high APFD, running identical test cases recurrently, the enormous size of the test suite, resource scarcity, and incomplete coverage of code[8,9].

TCP techniques using history-based approaches are gaining popularity among researchers over time and the availability of historical data has also increased[10]. Previously it was difficult to find historical data because it was not recorded but now with the efforts of researchers and industry, few platforms are available from where historical data can be acquired[2]. Open source datasets are more commonly available as compared to industrial datasets[10]. Secondly, historical data can be generated by executing and analyzing the source code available in version control repositories but with the introduction of newer versions of software tools, support for previous ones usually ends, so while executing old projects to acquire data researchers might run into errors. Efforts are required to provide up-to-date datasets to the research community so that different techniques can be put to the test.

This research has three objectives; the primary objective is to implement history-based TCP techniques dispersed in the literature under one roof, the secondary objective is to investigate the problem of equal priority in history-based TCP techniques and the tertiary objective is to explore random sorting as a solution to the problem of equal priority in history-based TCP techniques. Similar work was found in the literature but with different objectives[10], the study has an objective to establish which techniques work well for open and closed-source projects and whether there is a technique that works effectively for both types of projects. However, the study at hand focuses on highlighting the problems of history-based TCP techniques.

Literature Review

Researchers have presented history-based techniques with only one history-based factor[11-13]. Secondly, different studies have been found that have utilized more than one history-based factor[14,15]. Some researchers have incorporated one or more than one type of TCP technique such as coverage based, or similarity-based with history-based technique to improve the performance of TCP in regression testing[16,17]. The drawback of one type of TCP technique may be reduced by combining it with another type of TCP technique.

The most recent failure (MRF) has been derived from the approach that makes use of history-based and similarity-based TCP techniques[16]. An approach that combined factors such as failure rate (FR) with test case age to form an indicator to prioritize test cases[15]. A history-based approach Exponential Decay (ED) that makes use of a statistical technique known as exponential smoothing to calculate the priority of test cases based on their test case execution results[11]. The approach can also be considered in terms of coverage and test case age-based TCP approaches. However, the coverage base is not under the scope of this study as this study focuses on history-based TCP approaches only. The history-based approach is widely used by researchers. An industrial weightage scheme named ROCKET (R) was introduced to prioritize test cases using historical records and the execution time of test cases[14]. The ROCKET weightage scheme gives the maximum importance to failure in the last run, then it gives medium level importance to failure in the second last run, and then least importance to failures in the third last run and all previous runs.

The approach known as a co-failure-based approach assigned failure probability to test cases according to failing test cases and then rearranged them accordingly[12]. The co-failure-based (CoF) approach is dynamic because reprioritizes the test cases after every run. The flipping history-based (FH) approach uses the ROCKET to identify the first failure and then statistically assigns priority to the test cases based on the result of flipping[13]. Flipping is when two test cases switch states simultaneously that is pass or fail together. Terminator (T) has been proposed in[17]. It has 3 different versions that are similarity and feedback-based, feedback and history-based, and similarity, history and feedback-based. However, this study considers only history and feedback-based approaches as similarity-based is not in the scope of this study. Furthermore, this technique was proposed for UI datasets that are different from normal test cases. There are other history-based techniques also which cannot be discussed here due to space limitations.

TCP techniques use different factors to prioritize test cases such as coverage-based factors, similarity-based factors, and requirements-based factors. Factors used in history-based techniques include test case execution time[14], test case age[11,15], and failure count[15]. However, the most fundamental factor used in history-based approaches is test case execution results[11-17]. Table 1 shows the factors used by history-based techniques.

Table 1. Factors used by history-based techniques

Technique	Test Case Execution Results	Test Case Execution Time	Test Case Age	Failure Count
Most Recent Failure (MRF)[16]	✓			
Failure Rate (FR)[15]	✓		✓	✓
Exponential Decay (ED)[11]	✓		✓	
ROCKET (R)[14]	✓	✓		
Co-failure (CoF)[12]	✓			
Flipping History (FH)[13]	✓			
Terminator (T)[17]	✓			

The most readily available data is related to test case executions. However, the earlier datasets of historical data of execution of test cases encounter the issue of imbalance that is there are more pass instances and fewer fail instances[2]. The imbalance can affect the techniques depending on this type of historical data. But with time the imbalance has decreased. This may be due to the modern way of software development by using version control tools such as GitHub and the like, where developers from around the world can work together and where the platform records most of the data. Another factor is the introduction of bug-tracking systems such as Bugzilla and Jira, where bugs found in the software are maintained. The nature and particulars of data used may also change over time, earlier the researchers used mutation-based testing, so the bugs generated were mutation-based similarly this resulted in the generation of mutation-based historical data afterward the trend shifted to using historical data of real bugs.

Materials and Methods

The history-based TCP techniques considered in this study are MRF, FR, R, ED, CoF, FH, and T. The most recent failure with random (MRFR), failure rate with random (FRR), ROCKET metric with random (RR), Exponential Decay with random (EDR), co-failure with random (CoFR), and AFSAC flipping history with random (FHR). The history-based techniques, history-based techniques with random and random sorting will be applied to the dataset. Then factors will be calculated by techniques and based on factors test cases will be arranged. In the next step test cases are run against historical results to calculate the APFD metric and execution time of each technique. Finally, APFD results will be plotted in a graph to provide results in a visual form to make comparison easier, and execution time will be displayed in tabular form. The whole process is presented in Fig. 1.

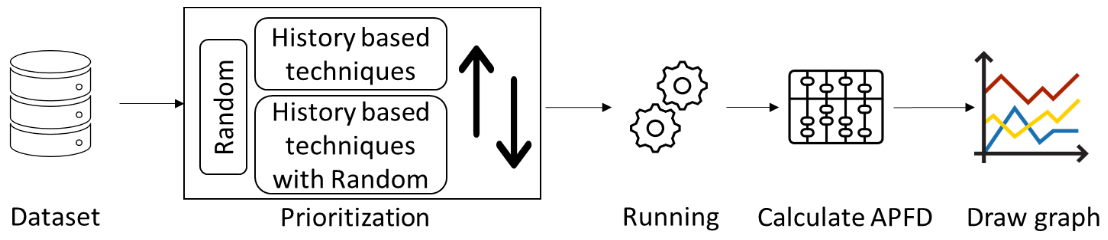


Figure 1. Flow of activities.

The dataset which contains execution results of test cases from a recent study[10] will be used. It provides data from 30 projects collected from GitHub and Travis CI, but only 12 projects will be included in this study due to time and space constraints. Some of the projects have an enormous amount of data. The execution time of history-based techniques on these projects may approach 18 hours or more[10] so projects with the appropriate amount of data were utilized in this study. These datasets are mostly based on Java and ruby programming languages, while few of them use Python and C++. The projects selected from the dataset[10] are deeplearning4j, structr, diaspora, okhttp, puppet, rspec, loomia, parsl, wicket bootstrap, radical, titan, and jetty project.

Table 2. Description of datasets selected from GitHub.

Project Name	Total Test Cases	Maximum Number of Failed Test Cases in one build	Maximum Number of times a single test case failed	Total Builds
Deeplearning4j	87	13	290	309
Okhttp	65	25	554	558
Puppet	54	12	361	386
Jetty project	159	56	156	156
Structr	188	44	830	830
Diaspora	104	62	251	1122
Titan	98	56	117	118
Rspec core	61	44	127	259
Radical	74	51	235	322
Parsl	77	6	15	193
Loomio	74	65	65	174
Wicket bootstrap	149	103	337	337

The dataset selected has some properties to ensure that suitable test cases are selected such as the number of test cases, number of builds, maximum number of times a single test case fails, maximum number of test cases failed in one build, number of failed test cases, number of developers, duration of project. The properties of selected projects from the GitHub repository are mentioned in Table 2. Those projects were selected which had enough failed test cases. The total number of test cases ranged from 54 – 188. The projects have more than 5 developers. The duration of all the projects is more than 1 year. The number of total builds ranges from 118 to 1122. Failed test cases range from 6 to 830.

Secondly, 5 datasets from the Software-artifact Infrastructure Repository (SIR) which is one of the oldest repositories were selected based on the properties of the projects[18]. SIR is a well-known repository and has been used extensively by researchers working on regression testing. It holds different datasets comprised of projects developed in the C programming language. In terms of test cases, the GitHub datasets have a smaller number of test cases while SIR datasets have a large number of test cases. GitHub projects were tested more extensively than SIR projects so the GitHub datasets are enriched with extensive build data with real faults which is a feature of modern workflow systems while the SIR dataset lacks this feature because, at the time of its inception, these modern tools were not available at the disposal of software.

The projects selected from the SIR repository are tcas, space, printtokens2, replace, and schedule2. Tcas has been used in regression testing studies[18-20]. Space has been used by researchers in software testing[20-23]. Replace was used by scientists[20-23]. Printtokens2 has been used by researhers[24,25-28]. Schedule2 was utilized by studies[29-31]. Table 3 contains the properties of datasets collected from SIR. It states the number of test cases, number of versions, maximum number of times a single test case fails, maximum number of test cases that failed in one version, and number of failed test cases.

Table 3. Description of datasets selected from SIR.

Project Name	Total Test Cases	Maximum Number of Failed Test Cases in one version	Maximum Number of times a single test case failed	Total Versions
Tcas	1608	8	133	41
Space	13585	13555	16	38
Printtokens2	4115	8	518	10
Replace	5542	10	309	32
Schedule2	2710	3	68	10

As this study is focused on history-based TCP the similarity-based approach was omitted and only the historical approach was considered[16]. Similarly, the dataset used does not come with the execution time of each test case, so it was not considered while implementing[14]. Only failure rate was considered in this study as test case age is not explicitly provided in datasets[15]. Secondly, it would be difficult to calculate it without sufficient information at hand. Thirdly, if all test cases were run then it would not make any significant difference.

Independent variables during the experiment were history-based techniques (MRF, FR, R, ED, CoF, FH, and T), history-based techniques with random ordering (MRFR, FRR, RR, EDR, CoFR, and FHR), and Random sorting. The dependent variable will be APFD and execution time. The experimental setup for conducting experiments consists of a desktop personal computer with a core i7 processor having 4 cores besides 8 threads and 16 GB of RAM.

The selected projects are briefly described here. Deeplearning4j is a collection of tools for implementing and training deep learning models using the JVM. Diaspora is a social network that keeps privacy at the forefront. Jetty project is an expandable web server written in Java language. It is insubstantial in terms of resource usage. Puppet is a server management system that can perform administrative tasks automatically. Structr is a GUI-based environment where users can create applications with minimal coding skills. Okhttp is an HTTP client which uses fewer resources and loads content faster. Rspec core is a tool for code maintainability. Loomio is a tool that supports organizations working together in decision-making. Parsl can be used to run Python on multiple systems to provide parallelism. Titan is a project with which large graphs can be processed and stored. Radical is a collection of tools that offer distributed computing facilities for different tasks. Wicket bootstrap can be used for the development of web applications.

The performance of TCP techniques is measured in terms of fault detection and it is calculated with the help of APFD[32]. APFD will be used to quantify the performance of TCP techniques. Execution time[10] will be used to measure the performance of the technique in terms of overhead incurred. APFD has been used in studies[33-36] to measure the effectiveness of TCP techniques. APFD can be calculated by the formula given in Eq. 1.

$$\mathrm{APFD} = 1 - \frac{\mathrm{TF1} + \mathrm{TF2} + \cdots + \mathrm{TFn}}{\mathrm{n} \times \mathrm{m}} + \frac{1}{2\mathrm{n}} \quad \text{............1}$$

In Eq.1, TF_1, TF_2, and TF_n represent the position of the first fault detected in each of the n test cases in the test set. The smaller the position, the better, as it indicates that the fault was found earlier in the test. The total number of test cases in your test set is represented by n. The total number of faults in the system being tested is represented by m. Time can be calculated by counting the number of seconds elapsed during the execution of a certain technique.

Results and Discussion

In this section, the performance and execution time of history-based TCP techniques will be presented and discussed when these techniques are applied to different data sets. It was found that most of the history-based techniques suffer from the problem of equal priority and to solve this problem if random sorting is employed it does not give optimal results. Fig. 2(a-i) shows the box plot of 12 selected projects and demonstrates the results of random, history base with random sorting, and history base techniques without random sorting. APFD is plotted on the y-axis. TCP techniques used in this study are plotted on the x-axis.

It can be observed that for all 12 projects, random sorting is the worst-performing technique as it does not arrange test cases according to some heuristic it just randomly arranges them. After the random sorting terminator is the second worst-performing technique, this finding is in line with the findings of[10]. MRF and ED are among the best-performing techniques as shown in Fig. 2(a-i). FR, R, CoF, and F perform similarly. Their performance lies in a mediocre range when compared to best and worst-performing techniques. However, it can be noticed that using random techniques with history-based techniques to solve equal priority does not yield better results, in fact, it slightly deteriorates the original performance of history-based techniques. Moreover, including random adds some extra overhead in terms of execution time. It is better to merge two techniques when their synergy offers better results than using the techniques separately.

Fig. 3(a-e) shows a boxplot of APFD of history-based approaches on the SIR dataset. For tcas dataset CoFR and CoF perform best, FHR and FH are the second best performing techniques, R and FR are the third best performing techniques and random is the worst performing technique when APFD is considered. The results of the SIR data set are contradictory to the GitHub dataset results in terms of APFD. For the Printtokens2 dataset except T, the other techniques produce similar results. Interestingly if look at the results of Schedule2 and Printokens2, T is the best performing technique and for the Replace dataset, it is the second best-performing technique in terms of APFD. But if execution time is considered while deciding to select the best-performing technique, the T technique is considered as the worst performing technique. So a decision regarding the best-performing technique can only be considered wise if both execution time and APFD are taken into consideration. For the space dataset, the execution time exceeded several hours so the execution was stopped and the APFD and execution time for Random, MRFR, MRF, FRR, FR, EDR, ED, RR, and R were collected.

It can be examined that there are outlier values in all 12 projects which means still the performance of prioritization techniques can be improved. Similarly, it can be noted in the SIR dataset results that still there is room for improvement. There is minimal number of outliers in SIR dataset results because the available data is limited. Upon careful inspection of the dataset, it was noticed that one reason behind the equal priority problem can be test cases that produce similar results, that is when one test passes the other passes, and when one test case fails the other fails.

MRF = Most recent failure, MRFR = most recent failure with random, FR = failure rate, FRR failure rate with random, R = ROCKET, RR = ROCKET with random, ED = Exponential Decay, EDR Exponential Decay with random, CoF co-failure, CoFR co-failure with random, FHR = AFSAC flipping history with random, FH = AFSAC flipping history, and T = terminator.

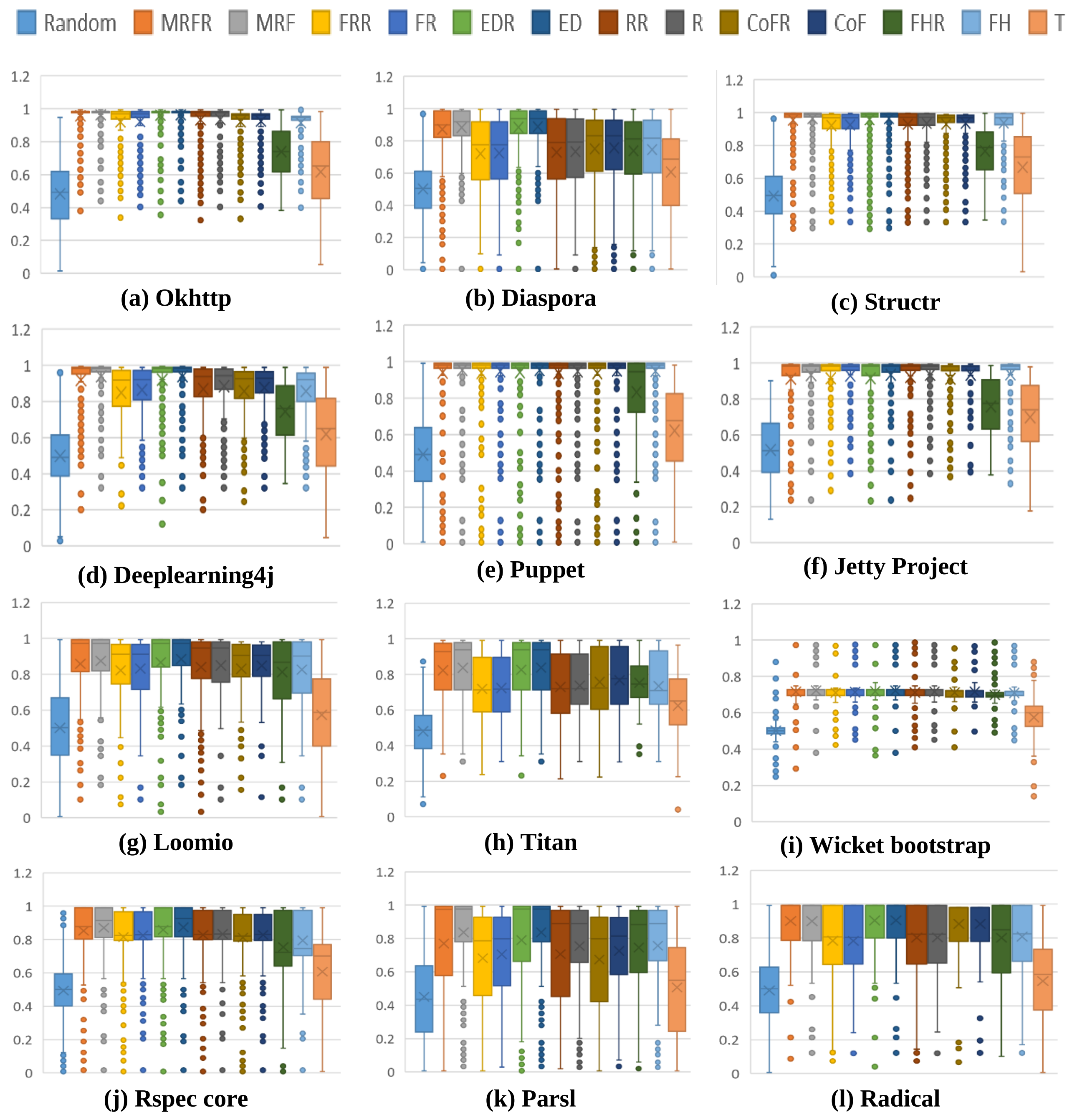
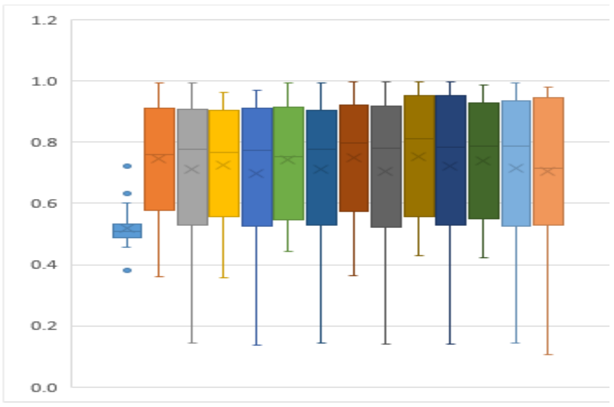


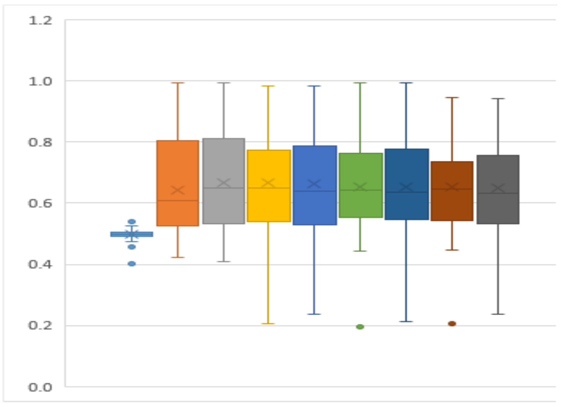
Figure 2(a-i). APFD of history-based approaches on GitHub datasets.

MRF = Most recent failure, MRFR = most recent failure with random, FR = failure rate, FRR failure rate with random, R = ROCKET, RR = ROCKET with random, ED = Exponential Decay, EDR Exponential Decay with random, CoF co-failure, CoFR co-failure with random, FHR = AFSAC flipping history with random, FH = AFSAC flipping history, and T = terminator.

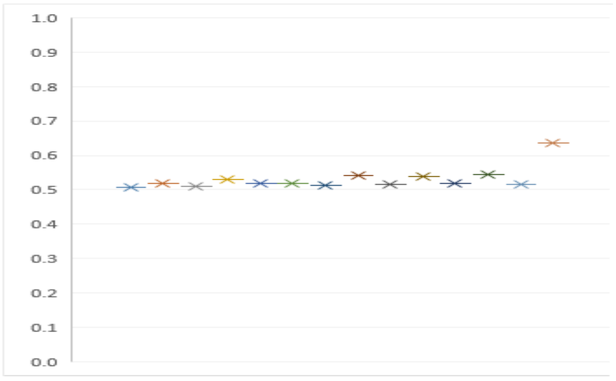
🟦 Random 🟧 MRFR ⬜ MRF 🟨 FRR 🟦 FR 🟩 EDR 🟦 ED 🟫 RR ⬛ R 🟫 CoFR 🟦 CoF 🟩 FHR 🟦 FH 🟧 T



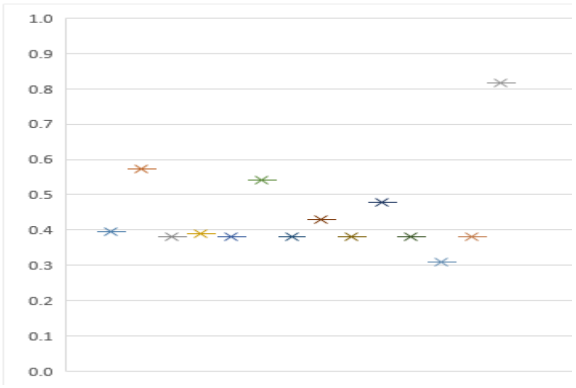
(a) Replace



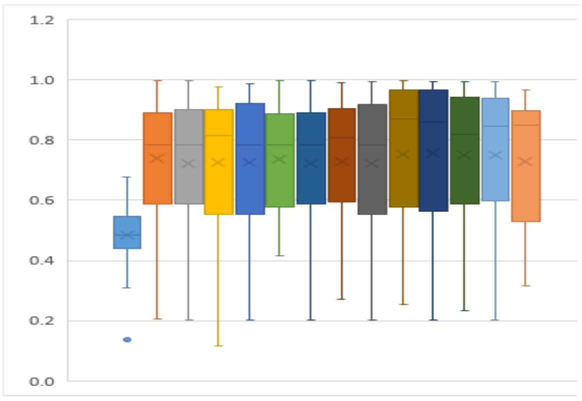
(b) Space



(c) Printtokens2



(d) Schedule2



(e) tcas

Figure 3(a-e). APFD of history-based approaches on SIR datasets.

A technique's performance may improve in terms of APFD if the number of versions and number of test cases are increased but at the same time, it becomes costly in terms of time to execute a technique on larger data sets. On smaller datasets such as printokens2 and schedule which have only

10 versions of software and a smaller number of test cases, the results are not dispersed due to the smaller number of data points available and hence no outliers can be tracked in Fig. 3(c) and Fig. 3(d). While Fig. 3(e) tcas, Fig. 3(a) replace and Fig. 3(b) space have a varied representation of the performance of test cases when history-based techniques are applied.

The execution time of history-based techniques when applied to GitHub and SIR datasets is shown in Table 4 and Table 5. In datasets from GitHub, the terminator is the most expensive technique in terms of execution time, Co-failure-based approach is the second most expensive in terms of execution time as it assigns failure probability and rearranges cases every time a test case fails. Flipping history-based approach is the third most expensive. Rocket is the fourth most expensive technique. While random is the least expensive technique for most of the data sets it is approximately equal to zero which can be viewed in Table 4

Table 4. Execution time of techniques on GitHub datasets.

	Rand.	MRFR	MRF	FRR	FR	EDR	ED	RR	R	CoFR	CoF	FHR	FH	T
Deeplearning4j	0.04	0.7	0.7	0.9	0.9	1.1	1.5	1.5	1.4	73	76	5	5	167
Diaspora	0.14	6.6	6.5	15.1	16.2	18.2	18	24	24	1408	1343	201	206	3218
Jetty Project	0.03	0.3	0.3	0.4	0.4	0.5	0.5	0.7	0.6	61	59	3	3	136
Puppet	0.03	0.9	0.9	0.8	0.8	1.1	1.1	1.4	1.4	44	45	4	5	104
Okhttp	0.06	2.1	2.0	2.3	2.5	2.8	2.8	3.6	3.6	139	139	10	12	311
Structr	0.18	12.5	11.6	14.7	17.5	24.3	23	35	34	3414	3381	135	147	7193
Wicket bootstrap	0.06	0.8	0.7	2.0	1.9	2.6	2.4	2.9	2.9	259	254	182	186	904
titan	0.01	0.1	0.1	0.3	0.2	0.4	0.3	0.4	0.3	18	17	3.2	2.64	45
parsl	0.01	0.2	0.3	0.3	0.3	0.4	0.4	0.5	0.5	23	23	0.9	0.97	51
radical	0.03	0.6	0.6	0.8	0.8	1.1	1.0	1.3	1.4	57	53	7.7	7.55	133
rspec score	0.01	0.3	0.3	0.5	0.4	0.6	0.6	0.8	0.7	26	25	6.2	6.30	68
loomio	0.01	0.1	0.1	0.3	0.3	0.3	0.3	0.4	0.4	17	17	2.2	2.28	41

MRF = Most recent failure, MRFR = most recent failure with random, FR = failure rate, FRR failure rate with random, R = ROCKET, RR = ROCKET with random, ED = Exponential Decay, EDR Exponential Decay with random, CoF co-failure, CoFR co-failure with random, FHR = AFSAC flipping history with random, FH = AFSAC flipping history, and T = terminator.

For SIR-based datasets, Table 5 shows that random is the best-performing technique according to execution time and terminator is the worst-performing technique which matches the GitHub dataset results. For the Space dataset, the execution time for CoFR, CoF, FHR, FH, and T is not available because the execution time went into the exponential domain so the execution was stopped. The reason behind this is the large number of test cases as compared to the other datasets from SIR.

Table 5. Execution time of techniques on SIR datasets.

	Rand.	MRFR	MRF	FRR	FR	EDR	ED	RR	R	CoFR	CoF	FHR	FH	T
Printtokens2	0.09	0.07	0.05	0.07	0.11	0.08	0.08	0.10	0.08	202	186	19	14	540
Replace	0.32	0.83	0.78	0.81	0.69	1.21	0.95	1.25	1.06	3600	2986	116	104	13896
Schedule2	0.02	0.05	0.03	0.04	0.03	0.04	0.03	0.04	0.03	82	61	2	2	401
TCAS	0.15	0.49	0.42	0.60	0.43	0.56	0.55	0.59	0.70	566	500	23	21	1463
Space	0.84	1.92	1.74	2.95	2.67	4.15	3.61	4.41	4.20	NA	NA	NA	NA	NA

MRF = Most recent failure, MRFR = most recent failure with random, FR = failure rate, FRR failure rate with random, R = ROCKET, RR = ROCKET with random, ED = Exponential Decay, EDR Exponential Decay with random, CoF co-failure, CoFR co-failure with random, FHR = AFSAC flipping history with random, FH = AFSAC flipping history, and T = terminator.

The variation in the performance of techniques over different datasets can be attributed to differences in software development methodology which has evolved. However, the execution time sets a clear message about the selection of technique for the desired task. The decision of which technique works best for a dataset depends on APFD and execution time. When time and APFD are considered most recent failure and exponential decay are the most suitable techniques for selected datasets. Combining random with history-based TCP techniques also incurs a cost that may be insignificant for smaller datasets but becomes significant when the size of the dataset increases. Secondly, it can be noticed that for best-performing techniques in terms of execution time, adding random to the technique slightly increases execution time but for techniques that are worst performers regarding time adding random would worsen the performance.

It was found that most of the history-based TCP techniques face the problem of equal priority while prioritizing test cases like other TCP techniques. However, when random sorting is used to solve this problem of equal priority, favorable results are not achieved. The performance of history-based TCP approaches deteriorates in terms of APFD, and more time is incurred. Besides the problem of equal priority, history-based TCP also faces the problem of unavailability of data historical data, small amount of available historical data, proper formation of data, and imbalance in historical data.

Conclusion

History-based TCP techniques are encountered with the problem of equal priority as many other techniques of TCP do. Secondly using random ordering is not the best solution to the problem of equal priority in regression testing. To get to the bottom of why equal priority issues are encountered by history-based techniques the researchers examined the dataset closely, it was found that the test cases are acting alike as they pass and fail simultaneously. Secondly, the properties inherited in the datasets due to development processes employed also play a major role in the ways certain techniques react to these datasets. Individual techniques respond differently because of the features of datasets. So, to solve this problem existing techniques are not sufficiently capable enough as demonstrated with the help of experiments. Code inspection-based approaches, coverage-based, and change-based approaches can be explored discretely and in combination in the future to solve the problem of equal priority in history-based TCP techniques.

Acknowledgment

The author would like to acknowledge the lab associates and technical support staff of the lab at UTM, Malaysia. The author wishes to convey appreciation to the Higher Education Commission (HEC), Pakistan for providing the opportunity to study abroad and for supporting the author financially for higher studies. The author additionally recognizes BUITEMS, Pakistan, the organization that offered opportunities and assistance for further education.

Authors’ Declaration

- Conflicts of Interest: None.
- We hereby confirm that all the Figures and Tables in the manuscript are ours. Furthermore, any Figures and images, that are not ours, have been included with the necessary permission for re-publication, which is attached to the manuscript.
- Ethical Clearance: The project was approved by the local ethical committee in Universiti Teknologi Malaysia.

Authors' Contribution Statement

S.M.J. H. discussed the idea with D.N.A J. and J. A. S.M.J. H., D.N.A J., and, J. A. designed the study, J. A. helped S.M.J. H. with data collection. S.M.J. H. performed the experiments. S.M.J. H. drafted the MS. D.N.A J. and J. A. proofread the MS. S.M.J. H. revised the manuscript according to the comments of D.N.A J. and J. A.

References

1. Younis MI, Alsewari AR, Khang NY, Zamli KZ. CTJ: Input-output based relation combinatorial testing strategy using jaya algorithm. Baghdad Sci. J. . 2020 Sep 8;17(3 (Suppl.)):1002-1009. https://dx.doi.org/10.21123/bsj.2020.17.3(Suppl.).1002
2. Khatibsyarbini M, Isa MA, Jawawi DN, Tumeng R. Test case prioritization approaches in regression testing: A systematic literature review. Inf. Softw. Technol.. 2018 Jan 1; 93:74-93. https://doi.org/10.1016/j.infsof.2017.08.014
3. Gupta A, Mahapatra RP. Multifactor Algorithm for Test Case Selection and Ordering. Baghdad Sci. J.. 2021 Jun 20;18(2 (Suppl.)):1056-1075. http://dx.doi.org/10.21123/bsj.2021.18.2(Suppl.).1056
4. Bajaj A, Sangwan OP. A systematic literature review of test case prioritization using genetic algorithms. IEEE Access. 2019; 7:126355–75. https://doi.org/10.1109/access.2019.2938260
5. Hao D, Zhang L, Zhang L, Rothermel G, Mei H. A unified test case prioritization approach. ACM Trans. Softw. Eng. Methodol.. 2014;24(2):1–31. https://dx.doi.org/10.1145/2685614
6. Hao D, Zhang L, Zang L, Wang Y, Wu X, Xie T. To be optimal or not in test-case prioritization. IEEE Trans. Softw. Eng.. 2016;42(5):490–505. https://dx.doi.org/10.1109/tse.2015.2496939
7. Srikanth H, Hettiarachchi C, Do H. Requirements based test prioritization using risk factors: An industrial study. Inf. Softw. Technol.. 2016; 69:71–83. https://dx.doi.org/10.1016/j.infsof.2015.09.002
8. Lou Y, Chen J, Zhang L, Hao D. Chapter one-a survey on regression test-case prioritization. vol. 113 of Adv. Comput. 2019;Volume 113:Pages 1-46 https://doi.org/10.1016/bs.adcom.2018.10.001
9. Dalal S, Assistant Professor, Maharshi Dayanand University, Rohtak, India. Challenges of regression testing: A pragmatic perspective. Int. J. Adv. Res. Comput. Sci. 2018;9(1):499–503. https://dx.doi.org/10.26483/ijarcs.v9i1.5424
10. Ling X, Agrawal R, Menzies T. How different is test case prioritization for open and closed source projects? IEEE Trans. Softw Eng. 2022;48(7):2526–40. https://doi.org/10.1109/tse.2021.3063220
11. Kim JM, Porter A. A history-based test prioritization technique for regression testing in resource constrained environments. In Proc. - 24th Int. Conf. Softw. Eng.. 2002 May 19 (pp. 119-129). https://doi.org/10.1145/581339.581357
12. Marijan D, Gotlieb A, Sen S. Test case prioritization for continuous regression testing: An industrial case study. In: 2013 IEEE Int. Conf. softw. maint. .IEEE. 2013. https://doi.org/10.1109/icsm.2013.91
13. Cho Y, Kim J, Lee E. History-based test case prioritization for failure information. In 2016 Proc. - 23rd Asia-Pac. Softw. Eng. Conf. (APSEC). IEEE; 2016. https://doi.org/10.1109/apsec.2016.066
14. Zhu Y, Shihab E, Rigby PC. Test re-prioritization in continuous testing environments. In 2018 IEEE Int. Conf. Softw. Maint. and Evol. (ICSME) 2018 ;Sep 23 :pp. 69-79. IEEE. https://doi.org/10.1109/icsme.2018.00016
15. Fazlalizadeh Y, Khalilian A, Azgomi MA, Parsa S. Prioritizing test cases for resource constraint environments using historical test case performance data. In: 2009 2nd IEEE Int. Conf. Comput. Sci. Inf. Tech.. IEEE; 2009. https://doi.org/10.1109/ICCSIT.2009.5234968
16. Hemmati H, Fang Z, Mantyla MV. Prioritizing manual test cases in traditional and rapid release environments. In: 2015 IEEE 8th Int. Conf. Softw. Test., Verif. and Valid.. (ICST). IEEE. 2015. https://doi.org/10.1109/ICST.2015.7102602
17. Yu Z, Fahid F, Menzies T, Rothermel G, Patrick K, Cherian S. TERMINATOR: better automated UI test case prioritization. In 2019: Proc. - 27th ACM Joint Meeting on Eur. Softw. Eng. Conf. and Symp. on Found. Softw. Eng.. New York, NY, USA: ACM; 2019. https://doi.org/10.1145/3338906.3340448
18. Qasim M, Bibi A, Hussain SJ, Jhanjhi NZ, Humayun M, Sama NU. Test case prioritization techniques in software regression testing: An overview. Int. j. adv. appl. sci. .2021 May;8(5):107-21. https://doi.org/10.21833/ijaas.2021.05.012
19. Khatibsyarbini M, Isa MA, Jawawi DN, Hamed HN, Suffian MD. Test case prioritization using firefly

algorithm for software testing. IEEE access. 2019 Sep 10; 7:132360-73. https://doi.org/10.1109/ACCESS.2019.2940620
20. Chen J, Gu Y, Cai S, Chen H, Chen J. A novel test case prioritization approach for black-box testing based on K-medoids clustering. J. Softw.: Evol. Process. 2023 Mar 31; e2565. https://doi.org/10.1002/smr.2565
21. Mukherjee R, Patnaik KS. A survey on different approaches for software test case prioritization. J. King Saud Univ. - Comput. Inf. Sci. .2021 Nov 1;33(9):1041-54. https://doi.org/10.1016/j.jksuci.2018.09.005
22. Zhou ZQ, Liu C, Chen TY, Tse TH, Susilo W. Beating random test case prioritization. IEEE Trans. Reliab. . 2020 Jun 16;70(2):654-75. https://doi.org/10.1109/TR.2020.2979815
23. Lu C, Zhong J, Xue Y, Feng L, Zhang J. Ant colony system with sorting-based local search for coverage-based test case prioritization. IEEE Trans. Reliab.. 2019 Aug 9;69(3):1004-20. https://doi.org/10.1109/TR.2019.2930358
24. Mahdieh M, Mirian-Hosseinabadi SH, Mahdieh M. Test case prioritization using test case diversification and fault-proneness estimations. Autom. Softw. Eng. .2022 Nov;29(2):50. https://doi.org/10.1007/s10515-022-00344-y
25. Ganjkhani E, Afsharchi M. An effective test case prioritization by combination of strategies. SN Appl. Sci. .2019 Sep; 1:1-4. https://doi.org/10.1007/s42452-019-1076-1
26. Chiang CL, Huang CY, Chiu CY, Chen KW, Lee CH. Analysis and assessment of weighted combinatorial criterion for test suite reduction. Qual. Reliab. Eng. Int.. 2022 Feb;38(1):358-88. https://doi.org/10.1002/qre.2984
27. Mondal S, Nasre R. Mahtab: Phase-wise acceleration of regression testing for C. J. Syst. Softw.. 2019 Dec 1; 158:110403. https://doi.org/10.1016/j.jss.2019.110403
28. Nithya TM, Chitra S. Soft computing-based semi-automated test case selection using gradient-based techniques. Soft Comput.. 2020 Sep;24(17):12981-7. https://doi.org/10.1007/s00500-020-04719-9
29. Jatana N, Suri B. Particle swarm and genetic algorithm applied to mutation testing for test data generation: a comparative evaluation. J. King Saud Univ. - Comput. Inf. Sci.. 2020 May 1;32(4):514-21. https://doi.org/10.1016/j.jksuci.2019.05.004
30. Qureshi N, Mukhija MK, Kumar S. RAFI: Parallel Dynamic Test-suite Reduction for Software. New Front. Commun. Intell. Syst, SCRS, India. 2021:165-76. https://doi.org/10.52458/978-81-95502-00-4-20
31. Wang R, Li Z, Jiang S, Tao C. Regression test case prioritization based on fixed size candidate set ART algorithm. J. Softw. Eng. Knowl. Eng.. 2020 Mar;30(03):291-320. https://doi.org/10.1142/s0218194020500138
32. Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Trans. Softw. Eng.. 2001 Oct;27(10):929-48. https://doi.org/10.1109/32.962562
33. Khatibsyarbini M, Isa MA, Jawawi DN, Shafie ML, Wan-Kadir WM, Hamed HN, Suffian MD. Trend application of machine learning in test case prioritization: A review on techniques. IEEE Access. 2021 Dec 14;9:166262-82. https://doi.org/10.1109/ACCESS.2021.3135508
34. Yan R, Chen Y, Gao H, Yan J. Test case prioritization with neuron valuation based pattern. Sci. Comput. Program.. 2022 Mar 1; 215:102761. https://doi.org/10.1016/j.scico.2021.102761
35. Huang Y, Shu T, Ding Z. A learn-to-rank method for model-based regression test case prioritization. IEEE Access. 2021 Jan 20; 9:16365-82. https://doi.org/10.1109/ACCESS.2021.3053163
36. Laaber C, Gall HC, Leitner P. Applying test case prioritization to software microbenchmarks. Empir. Softw. Eng.. 2021 Nov;26(6):133. https://doi.org/10.1007/s10664-021-10037-x

2024, 21(2 Special Issue): 0609-0621
https://doi.org/10.21123/bsj.2024.9604
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

دراسة استكشافية لتقنيات تحديد أولويات حالة الاختبار القائمة على التاريخ على مجموعات بيانات مختلفة

سيد محمد جنيد حسن[1،2]، دايانغ ن.أ. جواوي[1]، جوانا أحمد[1]

[1]كلية الحاسبات، الجامعة التكنولوجية الماليزية، جوهور باهرو، ماليزيا.
[2]قسم تكنولوجيا المعلومات، FICT، BUITEMS، كويتا، باكستان.

الخلاصة

في اختبار الانحدار، يعد تحديد أولويات حالة الاختبار TCP أسلوبًا لترتيب جميع حالات الاختبار المتاحة. يمكن لتقنيات TCP تحسين أداء الكشف عن الأخطاء والذي يتم قياسه بمتوسط النسبة المئوية لاكتشاف الأخطاء APFD .
يعد TCP المستند إلى التاريخ أحد تقنيات TCP التي تأخذ في الاعتبار تاريخ البيانات السابقة لتحديد أولويات حالات الاختبار. تعتبر مسألة تخصيص الأولوية المتساوية لحالات الاختبار مشكلة شائعة بالنسبة لمعظم تقنيات .TCPومع ذلك، لم يتم استكشاف هذه المشكلة في تقنيات TCP المستندة إلى التاريخ. لحل هذه المشكلة في اختبار الانحدار، يلجأ معظم الباحثين إلى الفرز العشوائي لحالات الاختبار. تهدف هذه الدراسة إلى تنفيذ تقنيات برنامج التعاون الفني القائمة على التاريخ والمنتشرة في الأدبيات تحت سقف واحد. الهدف الثاني هو دراسة مشكلة الأولوية المتساوية في تقنيات برنامج التعاون الفني المستندة إلى التاريخ. الهدف الثالث هو استكشاف الفرز العشوائي كحل لمشكلة الأولوية المتساوية في تقنيات TCP القائمة على التاريخ. تم جمع مجموعات البيانات من السجلات التاريخية لحالات الاختبار من المصادر التقليدية والحديثة. تم تطبيق تقنيات TCP المستندة إلى التاريخ على مجموعات بيانات مختلفة. تم فحص تقنيات TCP المستندة إلى التاريخ بحثًا عن مشكلة الأولوية المتساوية. ومن ثم تم استخدام الفرز العشوائي كحل لمشكلة تساوي الأولوية. وأخيرا، تم تفصيل النتائج من حيث APFD ووقت التنفيذ. تشير النتائج إلى أن التقنيات المبنية على التاريخ تعاني أيضًا من مشكلة الأولوية المتساوية مثل الأنواع الأخرى من تقنيات TCP .
ثانيًا، لا يؤدي الفرز العشوائي إلى نتائج مثالية أثناء محاولة حل مشكلة الأولوية المتساوية في برنامج التعاون الفني المستند إلى التاريخ. علاوة على ذلك، يؤدي الفرز العشوائي إلى تدهور نتائج تقنيات TCP القائمة على التاريخ عند استخدامها لحل مشكلة ذات أولوية متساوية. ينبغي للمرء اللجوء إلى الفرز العشوائي في حالة عدم وجود حل آخر. يتطلب قرار اختيار الحل الأفضل تحليل التكلفة والعائد مع مراعاة السياق والحل قيد النظر.

الكلمات المفتاحية: متوسط النسبة المئوية للخطأ المكتشف، الأولوية المتساوية، بناءً على التاريخ، عشوائي، اختبار الانحدار، تحديد أولويات حالة الاختبار.