An Empirical Study of Mutation-Based Test Case Clustering Prioritization and Reduction Technique

Longbo Li*, Yanhui Zhou*, Yong Yu*, Feiyan Zhao*, Shenghua Wu† and Zhe Yang†
*Department of Computer and Information Science
Southwest University
Chongqing, China
E-mail: {lilongbo iyuyong zfy201809}@email.swu.edu.cn, xiaohui@swu.edu.cn
†Meiyun Zhi Number Technology Co., Ltd.
Guangdong,China
E-mail: {wush18 yangzhe1}@meicloud.com.

Abstract—**Regression testing is an important activity to ensure software quality throughout the software life-cycle. However, due to the expansion of the software scale, a large number of test cases are generated in the regression test. In the actual regression test process, it is impossible for us to execute all the test cases. In order to save time and improve efficiency, we need to prioritize and reduce the test cases. In this paper, we propose a new concept mutation program unit priority that works well in the prioritization and reduction of test cases. To evaluate our approach, we designed the experiment and validated it using the Defects4J data set, which contains the real fault programs. We experimented with 350 real faults and 550254 developer-written test cases for Defects4J. The average reduction rate for test cases is 40%, and the fault detection capability is only reduced by 1.38%. The results show that the mutation-based test case prioritization and reduction method improves the effectiveness of test case prioritization and reduction technique.**

Keywords–test case prioritization; regression test; clustering algorithms.

I. Introduction

A large number of test cases need to be executed during the regression test, which makes the regression test process take a long time. For example, in the era when the application software version is updated frequently, we urgently hope that the regression testing can be executed quickly. A recent study shows that for Apache Geode Test takes 14 hours [1]. Many test case prioritization techniques have been empirically studied [2][3]. Researchers have created a variety of regression testing techniques. These include test case selection [4], test suite minimization [5] and test case prioritization [3].

Most of the existing test cases prioritization technology mainly depends on the code coverage information, code complexity metrics and expert knowledge. In a lot of empirical research, based on the code coverage information measurement was proved to be effective [6]. Using branch, statement, and mutation programs to study the effectiveness of test case prioritization techniques, they found that test case prioritization techniques based on mutation program have the best results [3][6]. We know that the use of hierarchical clustering algorithm (HCA) to cluster test cases based on code coverage information has achieved great results in test case prioritization [7]. Thus, we apply HCA to mutation-based test case prioritization and reduction.

In this paper, we use HCA to prioritize and reduce test cases, focusing on mutation program unit priorities and real fault programs. We believe that the priority of the mutation program unit in the test case prioritization is different. We propose the novel concept of the priority of the mutation program unit and give the calculation method. Based on the result of the mutation test, we generate the mutation program unit kill & priority matrix.

To verify the validity of our method, we use the Defects4J benchmark data set [8], which contains a large number of test cases written by developers, and Defects4J also contains real fault programs. The Defects4J data set [8] is widely used by many researchers in mutation testing, so we use it for our experiments. Our results show that the mutation-based test case prioritization and reduction, and use HCA can improve the effectiveness of test case prioritization and reduction technique.

The empirical research contributions of this paper are as follows:

- We proposed the novel concept of mutation program unit priority. For a large number of mutation programs that has many diverse attributes, and each corresponding mutation program has different priorities.
- Combining prioritization and reduction techniques, we experimented with 350 real fault programs and 550254 developer-written test cases in Defects4J. The test cases after prioritization and reduction are 330166. The average reduction rate for test cases is 40%, and the fault detection capability only lost by 1.38%.

The rest of the paper is organized as follows. Section II provides background for mutation-based test case prioritization and reduction technique. Section III presents a novel definition of the mutation program unit priority and test case prioritization evaluation method. Section IV describes the design of our empirical evaluation. Section V introduces the results of which are presented and analysed. Section VI discusses conclusion and future work.

II. Background

In this section, we will introduce the test case prioritization and test case reduction, and use formally language to describe the problem of research.

A. Test Case Prioritization

The goal of test case prioritization is to find an ideal test case execution order that exposes faults as early as possible. The test case prioritization approach was first mentioned by [9]. Subsequently, many researchers have carried out related

ISBN: 978-1-61208-752-8

research [2][3][10][11]. The test case prioritization problem was formally defined by [3].

Definition 1: Test case prioritization problem

Given:

a test suite,TS

the set of permutations of TS, PTS

a function that gives a numerical score for $T' \in PTS$,f

Problem:

find $T' \in PTS$ such that

$(\forall T'')(T'' \in PTS)(T'' \neq T')[f(T') \geq f(T'')]$

In Definition 1, PTS represents all possible orderings of the given test case in TS, and f represents an evaluation function that calculates an award value for an ordering $T' \in PTS$. Since we can't get the fault message during the regression testing progress, we usually need to consider a surrogate for fault detection based on the historical information of the test case. Hoping that early maximization of a certain chosen surrogate property will result in maximization of earlier fault detection. In a controlled-regression-testing environment, the result of prioritization can be evaluated by executing test cases according to the fault-detection rate [6].

The code coverage information, such as statement coverage and branch coverage are one of used surrogates in test case prioritization [3][12]. For example, a test case covering more statements has higher priority. The mutants are also used as another surrogate for test case prioritization [13][14]. For instance, the work in [3] consider the fault expose potential (FEP)-total approach that prioritize test cases according to the number of mutants killed by individual test cases, they find mutation-based test case prioritization techniques work better. In the process of mutation testing, a large number of mutants were generated. Donghwan et al. consider that these mutation programs have diverse attributes [15]. Based on this research, we propose a novel concept mutation program unit priority. For more details, we will discuss in Section III.

B. Test Case Reduction

The test case reduction is primarily to remove redundant test cases from the test case set, which usually do not change the mutation score in mutation-based test case reduction. For test case reduction problem, we can't reduce a test case set to a minimum set of test cases. The test case set minimization problem is an NP-complete problem. We try to reduce the test case set as much as possible without affecting the mutation score. Mike et al. introduces the relationship between the mutation score and real faults [16]. They believe that achieving higher mutation scores improves significantly the fault detection. Since the redundant mutation program problem is another area beyond the scope of this paper, we will not explain it in follow sections.

More formally, we consider the test suite reduction problem is defined as follows [6]:

Definition 1: Test suite reduction problem

Given:

a test suite, T, a set of test requirements $r_1, \ldots, r_n$, which must be satisfied to provide the desired adequate testing of the program, and subsets of $T, T_1, \ldots, T_n$, one associated with each of the $r_i s$ such that any one of the test case t_j belong to T_i can be used to achieve requirement r_i.

Problem:

Find a representative set, T', of test cases from T that satisfies all $r_i s$.

A number of test suite reduction approaches have been proposed in the literature [17]-[19]. Many researchers let statement coverage be the kind of test requirement considered, a reduced test suite that covers the same statements as the original test suite. Recently, a clustering test case reduction approach was proposed that reduced test suites only partially preserve the test requirement of the original test suites [20]. They empirically evaluate this methods that define guidelines for these to get trade-offs between reductions of in test suite size and losses of fault-detection capability. They mainly are concerned with the level of code coverage. In this paper, we use HCA to prioritize and reduce the test cases. We mainly focus on the level of program mutation.

III. Empirical Evaluation and Mutation Program Unit Priority

In this section, we introduce the novel concept of mutation program unit priority, mutation program unit kill & priority matrix, test case prioritization evaluation index and HCA.

A. Mutation Program Unit Kill Matrix

In the study of existing mutation-based test case prioritization problems, all mutation programs are considered to be equally important in test case prioritization process and no analysis of the priority of mutants. The minimum mutation program unit is usually called a mutant. For the mutants generated by the same mutation operator, we call a large mutation program unit. Due to the limited space of the paper, we only discuss the minimum mutation program unit in this paper. In the future work, we analyze and explain the whole mutation program unit theory in detail. Recently, The diversity-aware mutation-based techniques was proposed by [15][21]. They believe that many mutation programs are diverse and require more test cases to distinguish mutation programs, which aims to distinguish one mutant's behaviour from another mutation programs. Obviously, mutation programs have diverse attributes, and each mutation program has different priorities. When perform mutation analysis, the test case is executed into the mutation program unit, it is marked as 1 if the test case kills the mutation program, otherwise it is marked as 0. For example, Table I show mutation program unit kill matrix. $m_0 \cdots m_7$ show mutation program unit name and $T_0 \cdots T_4$ show test case name.

TABLE I. MUTATION PROGRAM UNIT KILL MATRIX

Test Case Name	Mutation Program Unit (MPU)							
	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
T_0	1	0	**1**	0	0	1	0	0
T_1	1	1	**1**	0	1	0	0	0
T_2	0	1	**1**	1	1	1	0	0
T_3	0	0	**1**	0	0	0	1	1
T_4	0	0	0	0	0	0	0	0

In general, in mutation analysis, we estimate the fault detection capability of test cases as measured by how many mutation programs are killed. In test case prioritization process, we might find such a set of sequences using greedy algorithm, for instance $T_2 - T_1 - T_0 - T_3$. Because T_2 kills the most mutation programs, it is chosen first. T_4 did not kill any of the mutants, we removed them in the reduction of the test case. However, we find that m_2 is killed by all test case. No matter how we prioritize it, m_2 will be killed when the first test case is executed. So, only the T_3 is in the first executed in the prioritization, the m_6, m_7 mutation program units are killed. In the prioritization process, mutation program units have different priorities. We should pay attention to mutation programs that are killed by very few test case. This mutation program is more difficult to kill.

TABLE II. MUTATION PROGRAM UNIT PRIORITY MATRIX

Test Case Name	**Mutation Program Unit(MPU)**							
	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
T_0	0.5	0	**0**	0	0	0.5	0	0
T_1	0.5	0.5	**0**	0	0.5	0	0	0
T_2	0	0.5	**0**	0.75	0.5	0.5	0	0
T_3	0	0	**0**	0	0	0	0.75	0.75

B. Mutation Program Unit Priority

Given a program under test (PUT), after mutation testing, generated m mutation program units, n test cases, we define an $m \times n$ mutation program unit kill matrix H that represents the kill information of the mutation program unit. H_{ij} indicates whether the test case j killed the mutation program unit i. We formally define this mutation program unit priority weight $W_i (0 < W_i < 1)$.

$$W_i = -\frac{\sum_{j=1}^{n} H_{ij}}{n} + 1 \tag{1}$$

As for (1) the value of W_i cannot be taken as 0 and 1 because in the prioritization and reduction we removed the mutation program unit with a value of 0 and 1. We updates mutation program unit kill matrix, Table II show mutation program unit priority matrix and m_2 is a redundant mutants by optimizing the weight calculation. Because m_2 will must be killed no matter how we prioritization. For example, we might find such a set of sequences using greedy algorithm in test case prioritization, $T_2 - T_3 - T_1 - T_0$. Similarly, in Table I, we remove T_4 from the mutation program unit kill matrix because it has no ability to kill all mutants.

C. Average Percentage of Fault-Detection

We usually use the Average Percentage of Fault-Detection (APFD) metric results in test case prioritization [22]. Higher APFD values confirm faster fault-detection rates. It is simply and accurately formalized as follows:

$$APFD = 1 - \frac{TF_1 + \cdots + TF_n}{nm} + \frac{1}{2n} \tag{2}$$

where TF_i is the first test case position in test case prioritization among n test cases which detects the ith fault among m faults. According to mutation program unit priority weight, we formally define the Average Percentage of Weight Fault-Detection (APWFD) as follows:

$$APWFD = 1 - \frac{W_1 \times TF_1 + \cdots + W_n \times TF_n}{\sum_{j=1}^{n} W_i \times n} + \frac{1}{2n} \tag{3}$$

where W_i is a weight as mutation program unit m_i. In order to better help readers understand, we using Table I and II results to clearly and accurately calculate APFD and APWFD. We use greedy algorithm to prioritize test case and based on killed the most mutation program. For example, $T_2 - T_1 - T_0 - T_3$ as for Table I and APFD is 0.5625. $T_2 - T_3 - T_1 - T_0$ as for Table II and APWFD is 0.7279. Only from the value of the result we have at least improved about 16.54%. Because we do not only consider the most mutants that is killed, but also consider the mutants that is hard to kill.

D. Hierarchical Clustering Algorithm

The clustering algorithm is an unsupervised learning algorithm that reveals the intrinsic properties and law of data. We obtain mutation analysis kill information by mutation analysis tool Major [23] and use Algorithm 1 (shown in Figure 2) generate mutation program unit kill & priority matrix. Cluster analysis was performed on the generated mutation program unit kill matrix and priority matrix, and the test case prioritization and reduction results were evaluated use APFD and APWFD. After configure data set Defects4J [8], we firstly checkout our every project program and compile fixed program. Thus, we use test execution framework and mutation analysis our fixed program. In Algorithm 1 (shown in Figure 2), T is contain developer-written test case set. The algorithm takes a developer-written test case set T, a mutation analysis result set of $mutantlog, testMap, killMap, trigger_bug$ as input, and returns a mutation program unit kill & priority matrix. In Algorithm 1 (shown in Figure 2), $killMap$ represents the mutation analysis tool kill information. The value of $testMap$ contain test case ID and name. Matrix information can be generated by use both information, and test cases that trigger faults can also be statistically analyzed. But we not conducted relevant research in this paper.

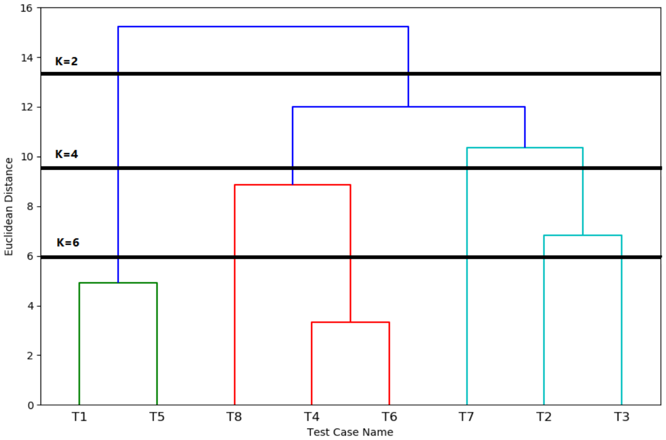
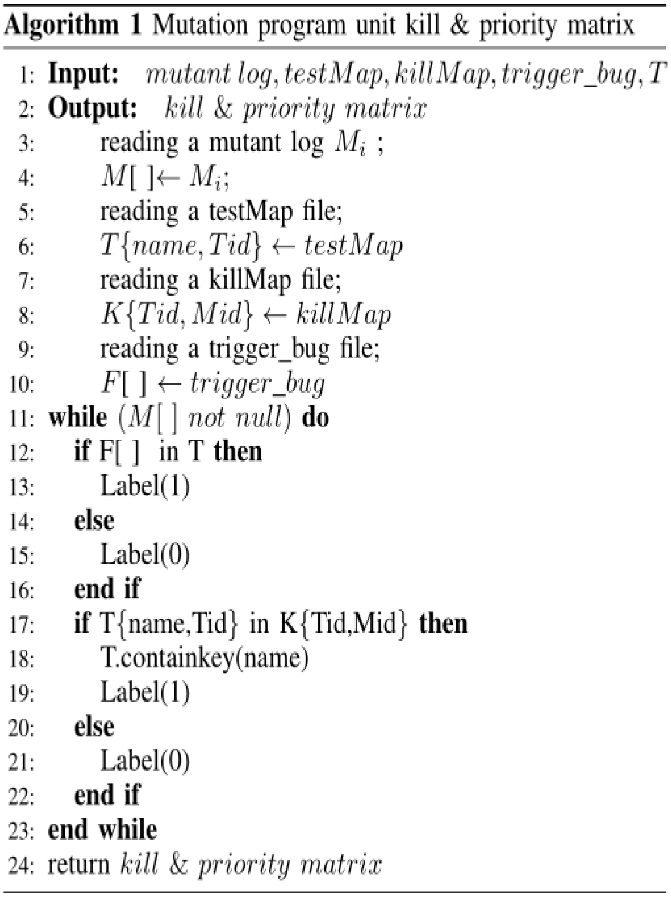


Figure 1. Hierarchical clustering tree.

Similarly, we choose a HCA, which can well control the size of clusters after clustering. HCA has achieved great test case prioritization results in terms of code coverage and expert knowledge [24][25]. Figure.1 shows the hierarchical clustering tree generated by HCA. The x coordinate represents the name of the test case, and the y coordinate represents the average Euclidean distance between test cases. We can easily control the size of the cluster generated by the HCA, according to the size of the test case set. Each vertical represents a cluster. The k is the size of the clusters. We use fast cluster function and improve prioritization and reduction strategy [26]. Recently, in the clustering prioritization and reduction of test cases based on code coverage, Carmen et al.[20] studied the effects of different clustering calculation methods on the test case prioritization and reduction, and found that clustering results are obviously different. Based on their research, our first research question in the experiment explores the impact of different computational methods on mutation-based test case prioritization and reduction.



```
Algorithm 1 Mutation program unit kill & priority matrix
Input: $mutant\ log, testMap, killMap, trigger\_bug, T$
Output: $kill\ \&\ priority\ matrix$
    reading a mutant log $M_i$ ;
    $M[\ ] \leftarrow M_i$;
    reading a testMap file;
    $T\{name, Tid\} \leftarrow testMap$
    reading a killMap file;
    $K\{Tid, Mid\} \leftarrow killMap$
    reading a trigger_bug file;
    $F[\ ] \leftarrow trigger\_bug$
while $(M[\ ]\ not\ null)$ do
  if F[ ]  in T then
    Label(1)
  else
    Label(0)
  end if
  if T{name,Tid} in K{Tid,Mid} then
    T.containkey(name)
    Label(1)
  else
    Label(0)
  end if
end while
return $kill\ \&\ priority\ matrix$
```

Figure 2. Algorithm for mutation program unit kill & priority matrix.

IV. Experimental design

In this section, we conduct empirical evaluation and design experiment that use developer-fixed program, developer-written tests, and real faults. As for empirical evaluation, we investigate the following two main research problems:

- RQ1: Is the different distance calculation methods have different effects on test case prioritization and reduction?
- RQ2: Is the test case prioritization and reduction techniques that outperforms in terms of trade-off between reductions in test suite size and losses in fault-detection capability?

We use benchmark data sets Defects4J [8] that include mutation analysis tool Major [23], developer-fixed program, manually-verified real fault and developer-written test case. Figure 3 shows the overall flow of our experiment. We will introduce our experiment in detail.

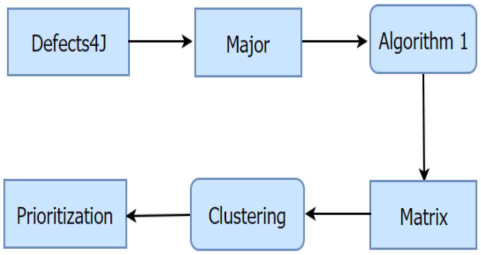


Figure 3. Experiment setup.

A. Experimental Tool

a) Defects4J database: We mainly use test execution framework in Defects4J that contains 438 bugs from the open-source projects. We consider five open-source projects (JFreeChart, Closure compiler, Apache commons-lang, Joda-Time and Apache commons-math) and 350 fixed programs. Because some fixed programs includes seldom developer-written test cases. Therefore we delete some fixed program. For example, fixed program Chart-23 only include five developer-written test cases. One of our goals is to provide a method for prioritization and reduction for a large number of test cases, helping to improve the efficiency of test engineers in regression testing.

b) Major: As for mutation kill information, we use Major mutation analysis tool for generating and executing all mutants to the developer-written test case for each fixed-program. Major includes a set of commonly used mutation operator [27]. For example, Binary operator replacement (BOR), Unary operator replacement (UOR), Constant value replacement (CVR), Branch condition manipulation (BCM), Logical operator replacement, and Statement deletion (STD). Because different mutation operators produce different mutation programs. Therefore, we applied all the mutation operators in mutation analysis.

B. Design Review

We configure Defects4J in Ubuntu18.04 LTS (intel i7-7700 cpu, RAM 8G). For more detailed configuration information, please refer to Defects4J official webpage at [28]. To configure Defacts4J on the Ubuntu18.04 system, we firstly perform “checkout” on all the programs, and then compile. If the test command is executed at this time, the test case written by the tester included in the test case Defects4J is used. Secondly, we use Major to perform mutation analysis on all “checkout” programs. The time of mutation analysis is controlled within one hour, and the program that exceeds the time is deleted. Finally, we use algorithm to generate mutation program unit kill and priority matrix for mutation analysis results, cluster the mutation program unit kill matrix and priority matrix, and use APFD and APWFD to evaluate test case prioritization and

TABLE III. CALCULATION STRATEGY STATISTICAL ANALYSIS.

Program	Calculation method			
	Jaccard	Hamming	Euclidean	Cosine
Chart	20.4816	21.0925	21.0862	20.0210
Closure	127.2672	128.9538	128.9563	123.9185
Lang	52.9445	56.8870	56.8507	47.6511
Math	90.8519	93.7491	93.3974	91.4900
Time	26.4618	26.4767	26.4825	25.5815
Average	0.9085	0.9347	0.9336	0.8819

TABLE IV. TEST CASE PRIORITIZATION AND REDUCTION STATISTICAL ANALYSIS

Program	Analysis index			Analysis index		
	APFD	Cluster time(s)	Test case	APWFD	Cluster time(s)	Test case
Chart	20.0210	2.0764	5693	18.7334	0.1189	3577
Closure	123.9185	1649.2548	440296	123.1317	758.1702	182155
Lang	47.6511	10.0078	11338	48.4177	1.5700	6480
Math	91.4900	165.7274	22688	88.2791	63.9586	9941
Time	25.5815	122.3370	70239	25.2988	71.4638	17935
Average	0.8819	5.5697	-	0.8681	2.5579	-

reduction results. Similarly, we can also use the automatic generation tools that generate a large number of test cases. Exploring the impact of different types of test case sets on test case prioritization and reduction is beyond the scope of this paper. We can delve into this issue in future research work.

V. RESULT AND ANALYSIS

In this section, we discuss the experiment obtained results and answer Section IV two questions.

A. RQ1: Is the different distance calculation methods have different effects on test case prioritization and reduction?

RQ1 explores the impact of different computational strategies on clustering results. Because of different clustering results have an impact on code coverage-based test case prioritization. Carmen et al. used code coverage information to study different computational strategies for hierarchical clustering [20]. They point out cluster test case, the use of the cosine and Jaccard-based dissimilarities seems to be more promising than the use of the Euclidean and Hamming.

We use mutation-based information to study different computational strategies for hierarchical clustering. We use four calculation methods(Jaccard, Hamming, Euclidean, Cosine) to conduct experiments. In order to control the experiment, only the calculation method is different, and the other experimental factors are all the same. Figure 4 shows that the x coordinate is the number of each program, and the y coordinate is the value of APFD, and four calculation strategies curve trend results are almost the same on the 350 real fault programs.

Different computing strategies maybe have different clustering effects on mutation-based test case prioritization. We can conclude that in the mutation-based test case prioritization in Figure 4, this strategy has no significant different. We also counted the average of four calculation strategies. Table III shows that cosine similarity method is lower than the other three calculation methods. However the difference is not obvious, the highest to the lowest is only 5.28%. Each column of data is the sum of real fault programs' APFD. For example, the 20.4816 of the $Chart$ program is the sum of the APFD values of all $Chart$ programs. Our experimental results show that there is no significant difference between test case prioritization based on mutation analysis using different calculation strategies for HCA.

B. RQ2: Is the test case prioritization and reduction techniques that outperforms in terms of trade-off between reductions in test suite size and losses of fault-detection capability?

RQ2 mainly explores test case reduction and loss of fault detection capabilities. In order to control the experiment, we use the Cosine dissimilarity distance calculation method. We use the prioritization and reduction strategy to calculate the value of APWFD in this paper. We used the value of APFD in one of the evaluation indicators in the control experiment.

From Table IV, it results that we used the proposed method to reduce and prioritize 550254 test cases. The number of test cases after prioritization and reduction is 330166. The average reduction rate for test cases is 40%. We also analyze the average reduction rate of each program. For example, the average fault reduction rate for the $Chart$, $Closure$, $Lang$,$Math$, and $Time$ programs are 62.82%,41.37%,57.15%,43.81% and 25.53% respectively. We use test case prioritization and reduction methods to make a large number of reductions to test cases. However the fault detection capability is only reduced by 1.38%. The average clustering time for each program is 2.5579s. Clustering time after prioritization and reduction reduced by 45.92%. This is a very interesting discovery, then software test engineers can use the test case prioritization and reduction techniques of clustering method to better manage and optimize regression testing activity. For example, a corporation does not have enough time to run all the test cases, and they still have a better chance of capturing faults.

Our experimental results show that the proposed prioritization and reduction strategy has good consequents in terms of reduction in number, time reduction and fault detection. Test case prioritization and reduction techniques ideal goal mainly is a higher test case reduction rate and a lower fault loss rate.

VI. CONCLUSION AND FUTURE WORK

This paper proposes a new concept mutation program unit priority. We use mutation-based test case prioritization and reduction strategies to prioritize and reduce test cases combine with mutation program unit priority. In the empirical evaluation, we used four different clustering calculation strategies to study the effects of different computing strategies on the results of prioritization test cases. The results show that the computational strategy has little effect on the results after clustering. We also present an empirical study comparing test case prioritization and reduction methods in terms of test case reduction in number and fault detection capability. Our method can reduce the number of test cases by 40%, and the loss of fault detection capability is only 1.38%.

In the future, we will continue to study the impact of different clustering numbers in test case prioritization and reduction. Similarly, we also noticed that there is no empirical analysis of the test cases that triggered the faults in the prioritization and reduction methods. We will analyze and explain the whole mutation program unit theory in detail.

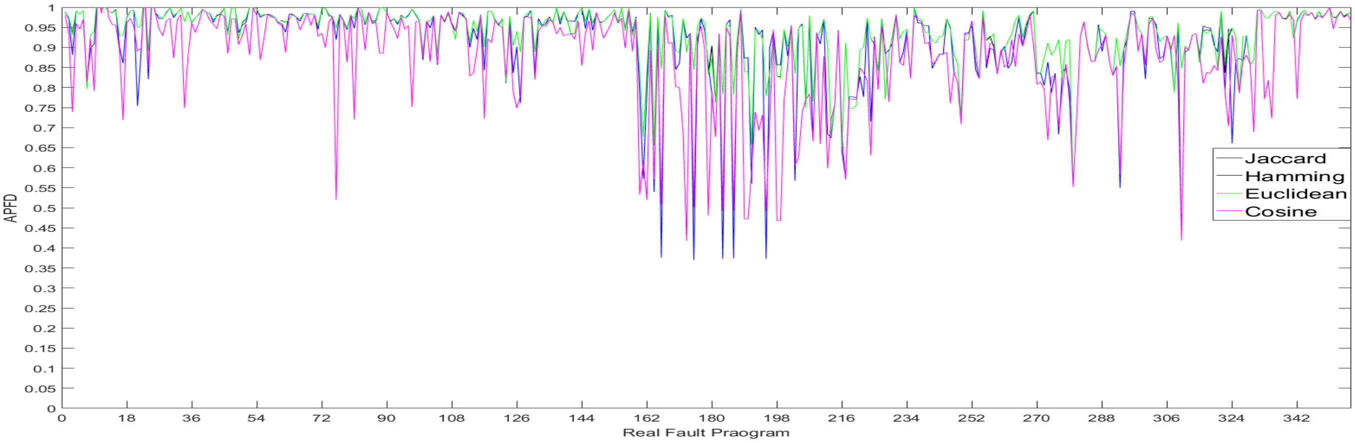


Figure 4. Calculation strategy.

REFERENCES

[1] Apache Geode Nightly Test Report.(2018).https://builds.apache.org/view/E-G/view/Geode/job/Geode-release/lastCompletedBuild/testReport/

[2] H. Do, G. Rothermel and A. Kinneer, "Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis," Empirical software engineering, vol. 11, no. 1, 2006, pp. 33-70.

[3] S. Elbaum, A. Malishevsky and G. Rothermel, "Prioritizing Test Cases for Regression Testing," IEEE Transactions on Software Engineering, vol. 27, no. 10, 2001, pp. 924-948.

[4] M. J. Harrold, D. Rosenblum, G. Rothermel and E. Weyuker, "Empirical studies of a prediction model for regression test selection," IEEE Transactions on Software Engineering, vol. 27, no. 3, 2001, pp. 248-263.

[5] J. Jones and M. Harrold, "Test suite reduction and prioritization for modified condition/decision coverage," IEEE Transactions on Software Engingeering, vol. 29, no. 3, 2003, pp. 193-209.

[6] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," Software Testing,Verification and Reliability, vol. 22, (2), 2012, pp. 67-120.

[7] R. Carlson, H. Do and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," IEEE International Conference on Software Maintenance (ICSM), 2011, pp. 382-391.

[8] R. Just, D. Jalali and M.D. Ernst , "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), 2014, pp. 437-440.

[9] W. Wong, J. Horgan, S. London and A. Mathur, "Effect of test set minimization on fault detection effectiveness," Software Practice and Experience , 28(4), 1998, pp. 347-369.

[10] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), 2002, pp. 97-106.

[11] H. Do, G. Rothermel and A. Kinneer, "Empirical studies of test case prioritization in a junit testing environment," Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE), 2004, pp. 113-124.

[12] L. Zhang, D. Hao, L. Zhang, G. Rothermel and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," In Proceedings of the 2013 International Conference on Software Engineering, 2013, pp. 192-201.

[13] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," IEEE Transactions on Software Engineering 2006, 32(9), pp. 733-752.

[14] Y. Lou, D. Hao and L. Zhang, "Mutation-based test-case prioritization in software evolution," In Proceedings of the 26th International Symposium on Software Reliability Engineering (ISSRE), 2015, pp. 46-57.

[15] D. Shin, S. Yoo and D.H. Bae, "A Theoretical and Empirical Study of Diversity-Aware Mutation Adequacy Criterion," IEEE Transactions on Software Engineering, vol. 44, (10), 2018, pp. 914-931.

[16] M. Papadakis, D. Shin, S. Yoo and D.H. Bae, "Are mutation scores correlated with real fault detection? A large scale empirical study on the relationship between mutants and real faults," International Conference on Software Engineering (ICSE), 2018, pp. 537-548.

[17] M. Jean Harrold, R. Gupta and M. Lou Soffa, "A Methodology for Controlling the Size of a Test Suite," ACM Transactions on Software Engineering, 2, 1993, pp. 270-285.

[18] Z. Li, M. Harman and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization," IEEE Transactions on Software Engineering. 33, 2007, pp. 225-237.

[19] L. Zhang, D. Marinov, L. Zhang and S. Khurshid, "An Empirical Study of JUnit Test-Suite Reduction," In Proceedings of International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2011 pp. 170-179.

[20] C. Coviello, S. Romano, G. Scanniello, A. Marchetto, G. Antoniol and A. Corazza, "Clustering support for inadequate test suite reduction," In Proceedings of International Conference on Software Analysis,Volution and Reengineering, Vol. 00, 2018, pp. 95-105.

[21] D. Shin, S. Yoo, M. Papadakis and D.H. Bae, "Empirical evaluation of mutation-based test case prioritization techniques," Software:testing verification and reliability, Vol. 29, (1-2), 2019, e1695.

[22] S. Elbaum, A. Malishevsky and G. Rothermel, "Test case prioritization: A family of empirical studies," IEEE Transactions on Software Engineering, 28(2), 2002, pp. 159-182.

[23] R. Just, "The Major mutation framework: Efficient and scalable mutation analysis for Java," In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), ACM, 2014, pp. 433-436.

[24] R. Carlson, H. Do and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," IEEE International Conference on Software Maintenance (ICSM), 2011, pp. 382-391.

[25] S. Yoo, M. Harman, P. Tonella and A. Susi, "Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge," Proceedings of the International Symposium on Software Testing and Analysis(ISSTA), 2009, pp. 201-211.

[26] D. Mllner, "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python," Journal of Statistical Software, 53, no. 9, 2013, pp. 1-18.

[27] A. S. Namin, J. H. Andrews and D. J. Murdoch, "Sufficient mutation operators for measuring test effectiveness," in Proceedings of the 30th International Conference on Software Engineering (ICSE), 2008, pp. 351-360.

[28] Defects4J. https://github.com/rjust/defects4j. last accessed on 10/04/19.