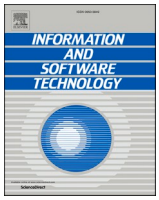
Information and Software Technology 165 (2024) 107339

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Cluster-based adaptive test case prioritization

Xiaolin Wang [a,b,c,d,*], Sulan Zhang [a]

[a] College of Information Science and Engineering, Jiaxing University, Zhejiang Jiaxing 314001, China
[b] Jiaxing Key Laboratory of Smart Transportations, Zhejiang Jiaxing 314001, China
[c] Business School, University of Shanghai for Science and Technology, Shanghai 200093, China
[d] Jujiang Construction Group Co., Ltd., Zhejiang Jiaxing 314500, China

ARTICLE INFO

Keywords:
Test case prioritization
Clustering analysis
Requirement
Regression testing

ABSTRACT

In order to enhance the efficiency of regression testing, test case prioritization (TCP) has been widely implemented, wherein a higher priority test case is executed earlier. Traditional TCP methods focus on improving the prioritization algorithm's efficacy. However, the majority of TCP approaches are characterized by a predetermined sequence of test cases prior to execution. Once established, this sequence remains consistent throughout the entire test execution process. As a result, any execution information generated during current test execution (such as fault-detected information) is unavailable for use in current round of test case prioritization and can only be utilized in subsequent regression testing. To address the issue of lagging utilization of fault-detected information, a cluster-based adaptive test case prioritization approach is proposed, which adds the new adaptive adjustment content in pre-prioritization. First, a new clustering criterion is defined and designed, by which produces test-case clusters in advance. Second, an adaptive TCP algorithm is proposed, which utilizes fault-detected information to adaptively adjust the order of test cases during the execution process based on the test-case clusters. Finally, one open-source Java program and three industrial-grade Java programs were selected for empirical evaluation. The experimental results demonstrate that the proposed technique not only serves as an enhanced version of pre-prioritization to improve the performance of the corresponding pre-prioritization technique, but also functions as an independent approach that outperforms other TCP techniques, including cluster-based TCPs, and another adaptive TCP. Specifically, when *step=2* is applied using our cluster-based adaptive TCP approach, the results are significantly better than those obtained with *step=1*. For instance, in *CT-14*, the median APFD improvement rate for *step=2* reaches 17.08 %, which is substantially higher than that achieved with *step=1* (5.48 %).

1. Introduction

Regression testing reuses test case sets to test modified programs, aiming to ensure the correctness of software behavior and that modifications will not adversely affect the unmodified parts of the program [1]. How to improve the effectiveness of regression testing is particularly important. Test Case Prioritization (TCP) is one of the important methods to improve the optimization efficiency of regression testing. It sorts test cases before test case execution according to certain rules, aiming to maximize the rate of fault detection of regression testing. And, most TCP techniques [2] can be considered as a preprocessing-testing approach, i.e., ordering test cases is preceded by test execution. The disadvantage of this preprocessing is that the test case prioritizing in current regression testing can only be operating based on the experience and historical data, and the information generated by the current test execution (e.g., fault detection information) cannot be used, which will have an impact on the rate of faults detection of TCP. The lag utilization of fault-detected information of current test execution is called the *hysteretic* problem in this paper.

In order to solve the *hysteretic* problem, this paper combines clustering analysis and adaptive algorithm to implement an adaptive prioritization and adjustment technique in the test-case execution process. The proposed technique expands the scope of test prioritization to encompass the test execution phase, integrating adaptive adjustments into pre-prioritization. Firstly, a clustering algorithm is designed to cluster test cases. Different from other cluster-based TCP methods, the clusters of test cases obtained in this paper are used to adjust test cases ordering during the test execution (most cluster-based TCP uses the

* Corresponding author at: College of Information Science and Engineering, Jiaxing University, Zhejiang Jiaxing 314001, China.
E-mail address: wangxiaolin@zjxu.edu.cn (X. Wang).

https://doi.org/10.1016/j.infsof.2023.107339
Received 21 March 2023; Received in revised form 11 September 2023; Accepted 30 September 2023
Available online 4 October 2023

clustering method in the preprocessing ordering before test execution). Secondly, an adaptive TCP technique is proposed to adjust the test sequence adaptively in the test execution process. If a test case detects faults during the test execution, the priority of the remaining unexecuted test cases will adjust their order adaptively. Different from the adaptive TCP method in literature [3], the adaptive TCP proposed in this paper uses the clusters of test cases to adjust the pre-ordered test case sequence.

In summary, the contributions of this work include:

- Fusion clustering techniques and adaptive algorithms solve the problem of the lag utilization of the current fault-detected information in test case prioritization.
- A clustering approach of test cases is designed, which will pave the way for the test case prioritization and adjustment in the test-execution process. At the same time, the acquisition methods of the three preconditions are described in detail.
- An adaptive test case prioritization technique is designed. The breakthrough underlying this approach is that it can extend the application stage of test prioritization to test execution and adaptively adjust the order of test cases by utilizing current fault-occurrence information to reduce the *hysteresis* problem. The feasibility of this technology is illustrated by a simulated example.
- A detailed experimental evaluation is presented by using one open source software system and three industrial projects. The experimental results not only validate the proposed technique as an enhanced version of the pre-prioritization, which improves the performance of the corresponding pre-prioritization technique, but also demonstrate its superiority over other TCP techniques as an independent approach. The conclusions are as follows: 1) Our proposed cluster-based adaptive TCP outperforms traditional TCP and cluster-based TCP in terms of fault detection efficiency. 2). Compared to the adaptive TCP method discussed in literature [3], our proposed method achieves a similar fault detection rate but incurs less test execution-time overhead. 3). When comparing different "*dissimilarity*" metrics, our proposed method demonstrates superior performance under the "correlation" measure. Additionally, if the "total" strategy is selected for priority processing in our proposed method, the role of "correlation" will be further enhanced; 4). Through comparing different adjustment *step*, the proposed method exhibits superior sorting efficacy under *step=2*.

The rest of this paper is organized as follows. Section 2 introduces the background of test case prioritization, clustering analysis, and existing related techniques. Section 3 focuses on describing our proposed cluster-based adaptive test prioritization technique, including the clustering criterion and the prioritization algorithm. Section 4 describes our empirical evaluation and analyzes the results. Section 5 discusses related work on TCP, cluster-based testing, and adaptive testing. Finally, the conclusions and future work are given in Section 6.

2. Background

2.1. Test case prioritization

2.1.1. Prioritization methodology

Test case prioritization is defined as follows [4,5]:

Given: T, a test suite already selected, PT, the set of all possible prioritizations (orderings) of T, and *f*, an objective function from PT to the real numbers, yields an award value for that ordering.
Problem: Find $(\forall T')(T' \in PT)(T' \neq T)[f(T') \geq f(T'')]$.

Elbaum and Rothermel et al. [2,4,5,6] pioneered test case prioritization techniques of the fine-grained entity, such as coverage prioritization (statement or branch coverage, etc.). This type of method is a classic method for TCP problems, with significant effects but high overhead. At the same time, they proposed the Total strategy and the Additional strategy for test prioritization based on the greedy algorithm. In our previous research [7], a history-based TCP technique was proposed based on historical data and the requirement-test case relationship. This technique includes two TCP algorithms based on requirement adjustment, which has a great advantage in improving the efficiency of regression test prioritization.

2.1.2. Traditional TCP techniques

The five TCP techniques are briefly introduced as follows.

Random prioritization [6]. Random prioritization orders test cases randomly. It is simple and convenient to operate, but unstable.

Total coverage prioritization [6]. Total coverage prioritization orders test cases based on the descendent number of units covered by these test cases. When multiple test cases cover the same number of units, the order is determined randomly.

Additional coverage prioritization [6]. Additional coverage prioritization orders test cases to achieve maximized coverage as early as possible. It first picks the test case with the greatest coverage, and then, successively adds those test cases that cover the most yet uncovered parts.

Total adjustment of requirements prioritization (TAR) [7]. This approach orders test cases according to the Requirement-based Priority (RP) values. First, it adjusts Importance Value (IV) of all requirements based on the difference of the numbers of detected faults in adjacent two regression testings. Then, it recalculates the RP value of each test case by the T-R relationship matrix and reorders test cases based on the descending order of RP values.

Additional adjustment of requirements prioritization (AAR) [7]. This approach combines the total adjustment of requirements prioritization and the additional coverage prioritization. It takes into account maximizing requirements covered by next selected test case. This approach and total adjustment of requirements prioritization are both from history-based dynamic test case prioritization technique proposed in our previous work [7].

2.1.3. Adaptive TCP techniques

In recent years, the adaptive technique, as a main method of dynamic programming, have been applied to test case prioritization to solve the *hysteretic* problem of traditional TCP techniques [3]. In reference [3], an adaptive test case prioritization technique based on output inspection is proposed. It schedules the execution order of test cases according to their fault detection ability during the execution of test cases. The process is as follows.

First, the initial fault-detection capability (*Priority(t)*) of each test case *t* is calculated based on the previously output execution information, and a test case t_s with the largest *Priority* is selected. Second, t_s is executed on the modified program and the output of t_s is recorded. Third, the *Priority* values of the remaining unselected test cases are modified based on the output of t_s, and the modified test case with the largest *Priority* is selected. Repeat the above steps until all test cases are sorted and executed.

2.2. Clustering analysis

Clustering analysis is one of the multivariate statistical analysis methods and an important branch of unsupervised pattern recognition. It divides the unmarked population into several subsets according to certain rules, and similar objects are classified into the same cluster as much as possible. The degree of dissimilarity between two objects can be measured by a "*dissimilarity*" metric. There are usually two kinds of metrics: one is "distance", which describes the degree to which two objects are close; The other is "correlation," which describes the degree to which two objects are related.

Fig. 1 illustrates the two kinds of metrics in terms of *dissimilarity*. In Fig. 1, there are four objects: *m, n, p*, and *q*. Among them, *m* and *n* are

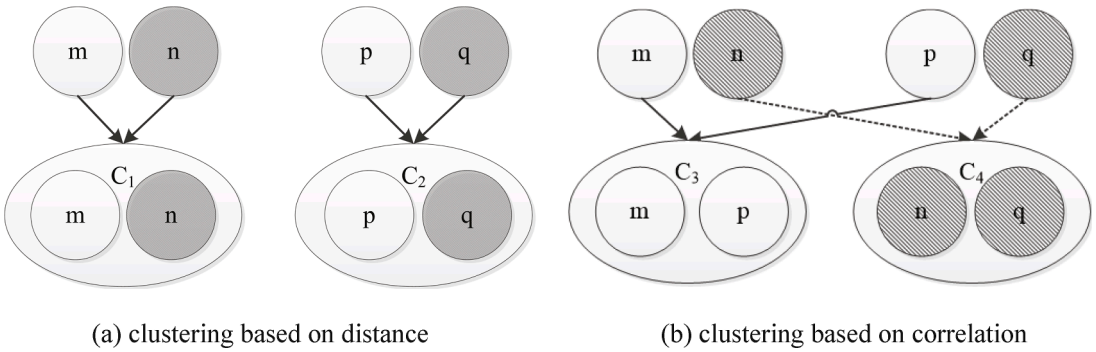


Fig. 1. An illustration of the two kinds of metrics.

located closer, and p and q are located closer. However, m and p have a similar feature (light color), while n and q have another similar feature (diagonal lines). Therefore, Fig. 1(a), which is based on "distance" metric, puts m and n into a cluster C_1, and p and q into another cluster C_2; Fig. 1(b), which is based on "correlation", puts m and p into a cluster C_3, and n and q into another cluster C_4.

2.2.1. Clustering methodology

The two common types of clustering analysis methods are partition clustering and hierarchical clustering [8]. Partition clustering initially divides the population into n clusters randomly. By reallocating individuals to different clusters, the quality of the clusters can be iteratively improved. This method can obtain high-quality clusters, but the computational process is expensive. Hierarchical clustering is a process of gradually producing clusters, which can be divided into: agglomerative method and divisive method. Although hierarchical clustering is faster than partition clustering and can produce different number of cluster partitions in a single run, it may not produce as good quality clusters as partition clustering, because the merge or split operation is irreversible once executed.

2.2.2. Cluster-based TCP techniques

In test optimization, cluster analysis is mainly used for test selection, that is, filtering and sampling effective test cases with the idea of clustering. Among them, the sampling strategy is the focus of test selection. In recent years, some researchers use the idea of cluster selection to conduct TCP research.

Cluster filtering-based TCP [9–11]. It selects and prioritizes test cases through the following process: first, test cases are divided based on clustering analysis. Second, test cases are selected according to the sampling strategy (*Random sampling, One-per-cluster sampling, Failure pursuit sampling, etc.*). Finally, the selected test cases are prioritized.

Requirement-based clustering prioritization [12]. First, it uses textual similarity and k-means clustering for requirement clustering. Second, the test cases are divided into clusters according to the requirement-test case traceability matrix. For the priority of test cases in the clusters, the code complexity metric is used for test sorting; for the priority of test clusters, the priority of the requirement clusters is used for prioritizing the test clusters.

Interleaved clusters prioritization [8]. This technique sorts test cases by interleaving clusters of test cases. First, the intra-cluster prioritization is performed. Second, a representative is picked from each cluster, and the clusters are prioritized by ranking the representatives. Finally, a cross-selection process is performed using the results of both intra- and inter-cluster prioritization.

3. Cluster-based adaptive test case prioritization

This section designs the cluster-based adaptive test case prioritization (CAP for short). Fig. 2 is the design framework of CAP. The yellow module constitutes the core research content of this paper, encompassing two key aspects: test cases clustering prior to execution and adaptive TCP during execution. In terms of test case clustering, the requirements clustering criteria has been designed to include three preparatory conditions: requirements, test cases and requirement-test case relationship matrix. During the execution of test cases, an adaptive sorting algorithm has been developed to adjust the pre-prioritized test sequence for greater efficiency.

3.1. Precondition

Definition 1 requirement. Given a software version, $\boldsymbol{Re}=\{r_1, r_2, \ldots \boldsymbol{r_n}\}$ is the set of requirements, which is the set of functions that needs to be implemented when coding the software. Where r is a single requirement that has at least one test case covering it.

Table 1 shows an example of requirements for a login page. The first row indicates the ID number of the requirement, and the second row indicates the content of the requirement. For instance, the content of r_2 is login, which means that this requirement is a login function which contains some actions such as inputting username, checking password and so on. In this study, only using the requirement ID to represent the specific requirement.

Definition 2 atomic test case. The test actions, designed by testers based on the requirements, are called atomic test cases (represented by ***at***) in this paper.

Definition 3 test case. A test sequence (or test subset) is composed of several atomic test cases for certain requirement, which is called a test case. It is represented by t, that is, $t = \{at_1, at_2, \cdots at_n\}$. In this granular definition, each requirement can be covered by a test case, and each test case can cover at least one requirement. A test case covering part of a requirement will not occur.

Table 2 illustrates the relationship between test cases and atomic test cases. The first row shows part of atomic test cases. The first column shows two test cases. For example, from the second line, test case t_1 represents testing the login page, which contains five atomic test cases: at_1-at_5, i.e., checking the login page display, checking user name, checking password, checking login button, and checking the forgot password button.

Definition 4 test case-requirement relationship matrix. The corresponding coverage relationship between test cases and requirements is represented by a relationship matrix, which is called the test case-requirements relationship matrix, or T-R matrix for short.

Table 3 shows a T-R matrix. The first row indicates requirements and the first column indicates test cases. The row/column intersections indicate whether the test case covers the specified requirement. If so, it is indicated by $\surd$. For instance, test case t_1 covers requirement r_1 and r_2.

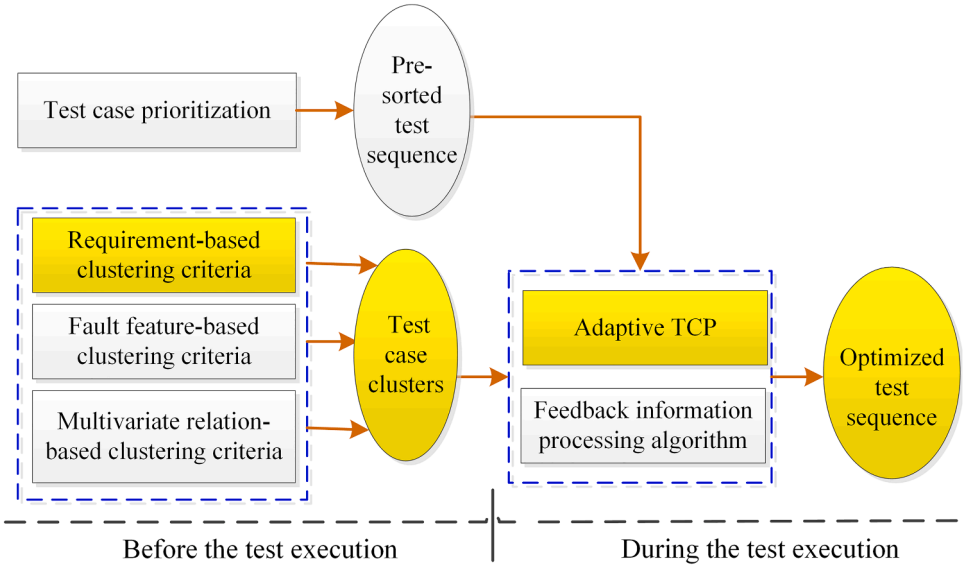


Fig 2. CAP Framework.

Table 1
Requirements for a login page.

ID	r_1	r_2	r_3	r_4	…
Req	homepage	login	register	find back the password	…

Table 2
Test case and atomic test case.

	at_1	at_2	at_3	at_4	at_5	at_6	at_7	…
t_1	√	√	√	√	√			
t_2	√				√	√	√	

Note: at_1 = checking the login page display, at_2 = checking user name, at_3 = checking password, at_4 =checking login button, at_5 =checking the forgot password button, at_6 = check the display of the password retrieval page, at_7 =checking reset password button; t_1 =testing the login page, t_2 = testing the password retrieval function.

3.2. Test case clustering

Definition 5 requirement coverage vector. Given a set of requirements $Re=\{r_1, r_2, \ldots r_n\}$ and a test case t, $\boldsymbol{R}(t) = \langle s_1, s_2, \ldots s_n\rangle$ is the requirement coverage vector with regard to t where

$$s_i = \begin{cases} 1 \ if \ t \ covers \ r_i; \\ 0 \ if \ t \ do \ not \ cover \ r_i. \end{cases} \quad (1)$$

Definition 6 test case similarity. Given two test cases t_i, t_j, and their requirement coverage vectors $\boldsymbol{R}(t_i) = \langle s_{i1}, s_{i2}, \ldots s_{in}\rangle$, $\boldsymbol{R}(t_j) = \langle s_{j1}, s_{j2}, \ldots s_{jn}\rangle$, their Similarity is defined as:

$$\boldsymbol{SR}(t_i, t_j) = \frac{\sum_{k=1}^{n} s_{ik} \cdot s_{jk}}{\sqrt{\sum_{k=1}^{n} (s_{ik})^2} \cdot \sqrt{\sum_{k=1}^{n} (s_{jk})^2}}. \quad (2)$$

Definition 7 cluster similarity. Given two clusters C_i, C_j, their Similarity is defined as:

$$\boldsymbol{SR}(C_i, C_j) = \mathrm{MIN}\{\mathrm{SR}(t_m, t_n) | t_m \in C_i, t_n \in C_j\}, \quad (3)$$

Where, t_m and t_n come from two clusters C_i and C_j respectively, and MIN refers to taking the minimum value of all $\mathrm{SR}(t_m, t_n)$.

Definition 8 clustering criterion. Given a test suite $T = \{t_1, t_2, \cdots t_n\}$, a threshold α, $\forall$ i,j,i ≠ j, $1 \le i.j \le n$, t_i and t_j are put in the same cluster if and only if $SR(t_i, t_j) > \alpha$.The greater α is, the similarity of t_i and t_j is.

Fig. 3 shows the two processes of clustering. The test cases on the left side represent the original test suite without clustering. From definition 6, it is assumed that threshold α is fixed at 50 %. In the first clustering process, $SR(t_1, t_2)$=60 % is calculated according to Eq. (2), so and t_1 and t_2 can be clustered into a cluster named C_1, seeing the process above the dotted line in Fig. 3. Similarly, cluster C_2 contains t_1 and t_3. Then, below the dotted line in Fig. 3, in the second clustering process, the similarity of C_1 and C_2 is calculated according to Eq. (3), so and $SR(C_1, C_2)$=60 % is obtained. C_1 and C_2 are merged into a new cluster *Cnew1*.

Combined with the hierarchical and partition methods, an algorithm of test cases clustering called "Clustering" is proposed, shown as Algorithm 1 for details.

It is found in the experiment that if the number of individuals in different clusters has a significant difference, the adaptive adjustment

Table 3
T-R relationship matrix.

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}
t_1	√	√										
t_2	√	√	√									
t_3	√	√		√								
t_4					√	√						
t_5					√	√	√					
t_6					√	√		√				
t_7					√	√	√	√	√			
t_8					√	√	√	√		√		
t_9					√	√	√	√			√	
t_{10}					√	√	√	√				√

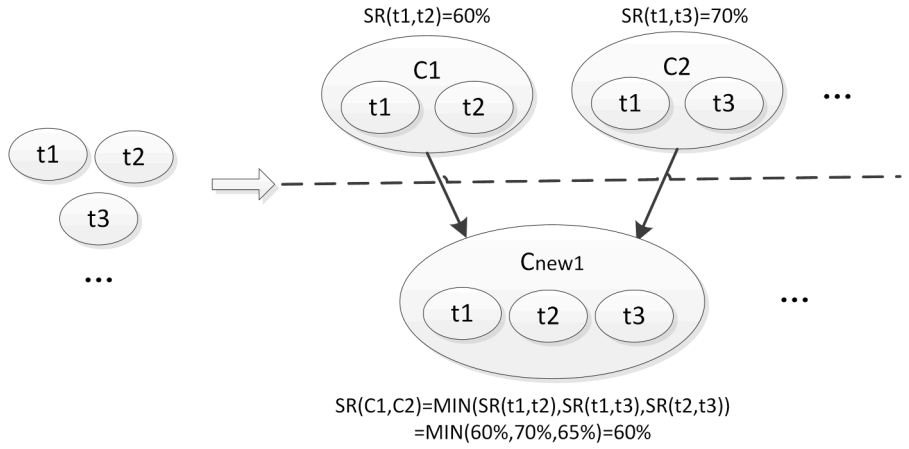


Fig 3. A clustering example.

Algorithm 1

Algorithm 1 Clustering

Input:
T // a test suite
α // The threshold of clustering
Output:
C //a set of clusters

$C = \emptyset$;
For each $t_i, t_j \in T$ //Initialization: get a binary-cluster set C
Computing $\boldsymbol{SR}(t_i, t_j)$;
If $(\boldsymbol{SR}(t_i, t_j) \geq \boldsymbol{\alpha})$ **then**
put t_i and t_j into a new cluster C_{new};
$C = C \cup \{C_{new}\}$;
End if
End for
Do // Iteration: make clusters merge.
For each $C_i, C_j \in C$
If $(\boldsymbol{SR}(C_i, C_j) \geq \boldsymbol{\alpha})$
merge C_i and C_j into a new cluster C_{new};
delete C_i and C_j from C;
$C = C \cup \{C_{new}\}$;
End if
End for
Until there are not any new clusters created // Break condition
For each $t_i \in$ T // Refine clusters
If t_i exists in several clusters **then**
leave t_i in the smallest cluster and remove the redundant t_i from other clusters;
End if
End for

effect of CAP will be affected. Therefore, if there is a significant difference (greater than 2) in the number of individuals in any two clusters (except isolated points), secondary clustering should be carried out. The large cluster is divided into several small clusters to ensure that the difference of the number of individuals in any two clusters does not exceed 1. When dealing with large clusters, Algorithm 1 is used iteratively to divide them into small clusters. If the vector pairs in the large cluster have equal SR values, the large cluster is randomly divided into equal quantities.

For example, in Table 3, it is described the T-R relationship matrix for test suite T $(t_1, t_2, t_3, \cdots, t_{10})$ and requirement set Re $(r_1, r_2, r_3, \cdots, r_{12})$. The *Requirement coverage vector* for each test case can be obtained from the table, e.g., $R(t_1) = \langle 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$ because $t_1$covers r_1 and r_2. The *Similarity* for any two test cases can be calculated, such as $SR(t_1, t_2) = 81.65\%$ and $SR(t_2, t_3) = 66.67\%$. Here, the threshold of the first round clustering is set as α=50 %. According to the Clustering criterion, two test case clusters are obtained: $<t_1, t_2, t_3>$ and $<t_4, t_5, t_6 t_7, t_8, t_9, t_{10}>$. The number of individuals in the first cluster is 3 and in the second cluster is 7. The difference of the number of individuals between two clusters is 4 which is more than 2, so the secondary clustering is carried out. In the secondary clustering, the threshold α is adjusted to 80 %, and finally three clusters are obtained: $<t_1, t_2, t_3>$, $<t_4, t_5, t_6>$ and $<t_7, t_8, t_9, t_{10}>$.

3.3. Adaptive test case prioritization

Definition 9 step. When the test sequence T is executed by priority, if a test case $(t \in T)$ detects some faults, the priorities of other

unexecuted test cases in the same cluster as t are adjusted, and the step size is *Step*. For example, when *Step*=1, move its position in the sequence forward by 1.

After the test case clusters obtained, the order of test cases can be adjusted during the test execution process. At the beginning of testing, each test case in test set T is assigned an initial priority according to the traditional test case prioritization techniques, which is called pre-prioritizing T in this paper. Then, the test cases in T are executed according to the order of their priorities. If a test case belonging to a cluster C_i detects faults during the test execution, the priorities of other unexecuted test cases belonging to C_i are adaptively incremented by a *Step*. Repeat the above process until all test cases are executed. Algorithm 2 describes the process in detail.

Consider such a prioritization problem: suppose that there is a test suite T=($t_1, t_2, t_3, \cdots, t_{10}$) as shown in Table 3, and the faults detected by these test cases are shown in Table 4. Suppose that test case cluster C (C_1, C_2, C_3) is obtained from Section 3.2, which are C_1(t_1, t_2, t_3), C_2(t_4, t_5, t_6) and C_3(t_7, t_8, t_9, t_{10}). In this case, the constant *Step* is set to 1. Random prioritization is selected as the pre-prioritization technique. Assuming that T and C are the inputs of CAP algorithm, we simulate the execution of the algorithm steps as follows.

Pre-prioritization-Algorithm 2: line 1. At the beginning, random prioritization is as a pre-prioritization technique, and all test cases are assigned equal priorities. Here, an order of test cases is randomly generated as the test execution sequence, i.e., t6-t3-t10-t7-t4-t5-t1-t9-t2-t8.

Inner loop-Algorithm 2: lines 2–8. t_6 is selected as the first test case according to line 2 and t6 is executed (line 3). t_6 finds no faults (lines 5–7), by Table 4. Then, it continues executing the test cases in order and the iteration goes to the second loop. t_3 is selected (line 2) and executed (line 3). t3 finds the faults during its execution (line 5). t_1 and t_2 are in the same cluster C_1(t_1, t_2, t_3) with t_3, so the locations of t_1 and t_2 are adjusted by improving their priority level by *1* (line 6). The test order becomes as t6-t3-t10-t7-t4-t1-t5-t2-t9-t8. Repeat the above process until all test cases are executed.

Final step-Algorithm 2: line 9. The order of test cases is adjusted while executed, according to above loop. The final test execution sequence is t6-t3-t10-t7-t4-t1-t9-t8-t2-t5.

The average percentages of faults detected (APFD, detailed introduction in the next section)) is used to evaluate the efficiency of the test sequence. The APFD of the test sequence obtained by CAP is 55 %, and the APFD of the test sequence obtained by random prioritization is 47.5 %. The speed rate of fault detection can be increased by about 15 % by applying CAP.

Thus, CAP proposed in this paper can detect faults earlier than the traditional random prioritization. Therefore, CAP is feasible in improving test efficiency.

Table 4
The fault detecting of test cases.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
t_1		√	√					
t_2	√							
t_3	√			√				
t_4								
t_5								
t_6								
t_7	√							√
t_8					√			
t_9							√	
t_{10}			√	√				

4. Empirical evaluation

To investigate the effectiveness of CAP, an empirical evaluation is performed in terms of the following research questions.

- RQ1: Is CAP more effective in the rate of fault detection than other traditional prioritization techniques?

This research question aims to understand whether CAP can detect faults earlier than other traditional prioritization techniques. To answer this question, this section applies five traditional TCP techniques and five corresponding CAP techniques for comparison.

- RQ2: Can CAP be more effective than other cluster-based prioritization techniques?

Clustering analysis has already been used in some regression testing optimization strategies. This research question mainly discusses whether CAP has any advantages comparing with other existing cluster-based TCP techniques.

- RQ3: How does CAP behave comparing with the adaptive prioritization technique (AP) mentioned in literature [3]?

AP also solves the *hysteretic* problem of the traditional TCP techniques. It also adjusts the order of the unexecuted test sequence after executing a new test case. This research question mainly discusses the average percentages of faults detected and the test execution-time cost (including the dynamic adjustment time of test sequence) between CAP and AP.

Algorithm 2 Adaptive Test Case Prioritization (CAP)

Input:
$T = \{t_1, t_2, \cdots t_n\}$ // a test suite
$C = \{C_1, C_2, \cdots C_m\}$ //a set of clusters
Output:
T' // the prioritized test sequence
1 : Pre-prioritize T;
2 : **While** (exists unexecuted test cases in T) **do**
3 : Select a test case t with the highest priority from T;
4 : Execute t;
5 : **If** (t detects faults) **then**
6 : Improve the priorities of the other unexecuted test cases in the cluster C_i with one *Step*, and $t \in C_i$ (i = 1 ... m);
7 : **End if**
8 : **End while**
9 : The actual executed order of test cases is as the output T'.

- RQ4: Between two kinds of *dissimilarity* metrics, which makes CAP generate more effective prioritization results?

There are usually two kinds of *dissimilarity* metrics: "distance" and "correlation". This research question mainly discusses the influence of two metrics on the implementation of CAP.

- RQ5: In different *Step* values, will CAP produce different prioritization effects?

In order to answer this question, the values of *Step* are set as 1 and 2 respectively, so as to carry out experiments on different CAP methods in different software versions, and discuss the influence of different Step values on the implementation of CAP.

4.1. Objects of analysis

To perform the empirical evaluation, four Java systems are used for the evaluation, with their modified versions and existing test suites. These systems are from diverse application domains: *XML-Security* is from the well-known Software-artifact Infrastructure Repository (SIR) [13] and the other three are industrial projects from three companies.

XML-Security. It implements the security standards for XML and is a large size system (LOC >10,000). It has 4 original versions, 28 test classes, and 78 test methods. Two versions (version 0 and version 1) are selected as the experimental objects and record them as *XS-0* and *XS-1*. The requirements, test cases, faults and other details of each version are seen in Table 5.

CPMISS. It is a Web application for community services and has approximately 440,000 lines of Java code (LOC). Five of the regression testing versions (version 2.1–2.5) are used as the experimental objects, which are recorded as *CS-1, CS-2, CS-3, CS-4*, and *CS-5*. The requirements, test cases, faults and other details of each version are seen in Table 5.

ChipTest. It is a chip test system for hardware and has approximately 140,000 lines of Java code (LOC). It is an agile development project. Each version contains a relatively small number of requirements, and the test suite is relatively small. Version 9 and version 14 are selected randomly as the experimental objects, and record them as *CT-9* and *CT-14*. The requirements, test cases, faults and other details of each version are seen in Table 5.

SSP. It is a sell-side platform for media services and has approximately 320,000 lines of Java code (LOC). Different from the systems described above, it has only one version, but more than 1000 atomic test cases. The requirements, test cases, faults and other details of each version are seen in Table 5.

In this experiment, each test case is independent, and there is no overlap between the contents of any two test cases. The faults of the industrial projects are the actual faults of the system, and no new faults are inserted. Each fault can be detected by a test case. The faults which cannot be detected by any test cases are not included in this study.

Table 5
The detailed information of four systems for each version.

	Version	Requirement	Atomic Test Case	Test Case	Fault
XML-Security	XS-0	7	20	13	7
	XS-1	7	25	15	8
CPMISS	CS-1	71	>300	51	102
	CS-2	71	>300	51	53
	CS-3	71	>300	51	5
	CS-4	71	>300	51	11
	CS-5	71	>300	51	6
ChipTest	CT-9	5	>100	9	14
	CT-14	12	>100	12	16
SSP	SSP	41	>1000	34	152

4.2. Variables and measures

4.2.1. Independent variables

To address the research questions, an independent variable is set: TCP technique. Four types of TCP techniques are considered, as follows:

- Traditional TCP (without test clustering, without the test sequence adjustment in test execution)

Five comparison techniques are used, including Random Prioritization, Total Coverage Prioritization, Additional Coverage Prioritization, Total Adjustment Requirement Prioritization and Additional Adjustment Requirement Prioritization, which are represented by R, TC, AC, TAR and AAR respectively. These techniques do not use clustering technology in the prioritization. During the test execution, the test sequence is not adjusted.

- Cluster-based Adaptive TCP (with test clustering, with the test sequence adjustment in test execution)

Five CAP techniques corresponding to the above traditional TCP techniques are used (pre-prioritization is the corresponding traditional TCP), including Random-CAP, Total Coverage-CAP, Additional Coverage-CAP, Total Adjustment Requirement-CAP and Additional Adjustment Requirement-CAP, which are represented by R-CAP, TC-CAP, AC-CAP, TAR-CAP and AAR-CAP respectively. These techniques use clustering technology in the prioritization. During the test execution, the test sequence is adjusted adaptively.

- Cluster-based TCP (with test clustering, without the test sequence adjustment in test execution)

Four techniques that have good performance in other literatures [8,10,12] are selected as the representative of cluster-based TCP, including Cluster-based code metric random, Basic coverage maximization + one-per-cluster sampling + pursuit + random, Basic coverage maximization + random and Interleaved Clusters Prioritization, which are respectively represented by Tcl-cm-random, Cov+OPC+Pur+Ran, Cov+Ran and ICP. These techniques use clustering technology in the prioritization. During the test execution, the test sequence is not adjusted.

- Cluster-based code metric random(Tcl-cm-random): Firstly, test cases are clustered according to requirements clustering. Then the code metric is used for test case prioritization in each cluster. Finally, randomly select the clustering order.
- Basic coverage maximization + one-per-cluster sampling + pursuit + random(Cov+OPC+Pur+Ran): First, test cases are selected in the order by the basic coverage maximization technique. Then, test cases are selected by the method of one-per-cluster sampling, after the ordered sequence based on the basic coverage maximization. According to the failure pursuit method, test cases are selected, ranked after the above test sequence. Finally, the remaining unselected test cases are randomly sorted and placed after the above sequence.
- Basic coverage maximization + random (Cov+ Ran): First, test cases are selected in the order by the basic coverage maximization technique. Then the remaining unselected test cases are randomly sorted and placed, after the ordered test sequence based on the basic coverage maximization.
- Interleaved Clusters Prioritization (ICP): First, a simple agglomerative hierarchical clustering technique is used to cluster test cases. Then, intra-cluster prioritization and inter-cluster prioritization both utilize coverage-based prioritization to prioritize test cases.

Finally, test cases are interleaved from each cluster in order, forming a test sequence.

- Adaptive TCP (without test clustering, with the test sequence adjustment in test execution)

The adaptive TCP technique in literature [3] is selected for the comparative experiment in this section, which is represented by AP. It does not use clustering technology in the prioritization. During the test execution, the test sequence is adjusted adaptively.

- Adaptive TCP (AP): This technique firstly calculates the fault-detection capability of unselected test cases based on the execution information of the previous program and the latest selected test case. Then, it runs the test case t with the largest fault-detection capability on the modified program and records whether the output of t passed or failed. By repeating the preceding two steps, finally, all the test cases are prioritized and executed.

4.2.2. Dependent variable and metric

4.2.2.1. APFD. To measure how rapidly a prioritized test suite detects faults, Average Percentage of Fault Detection (APFD) [2] is commonly utilized as the evaluation index. APFD does not consider the fault severity and the test case cost. The calculation of APFD is shown in Eq. (4):

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n}, \tag{4}$$

where n is the number of test cases and m is the number of faults. TF_i is the index of the first test case of revealing the ith fault in the test execution sequence. The value of APFD varies from 0 to 100 %. Since n and m are fixed for any test sequence in the same testing, a higher APFD value indicates the earlier detecting faults by the test sequence during the testing process.

4.2.2.2. APFDc. When evaluating the speed of fault detection, it is appropriate to use the Cost-cognizant weighted Average Percentage of Fault Detection (APFDc) [14] as a metric that takes into account both test case cost and fault severities. The calculation of APFDc is shown in Eq. (5):

$$APFDc = \frac{\sum_{i=1}^{m}\left(f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} \times t_{TF_i}\right)\right)}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i}, \tag{5}$$

where t_i is the execution cost of the ith test case and f_i is the severity of the ith fault. The meaning of other variables is consistent with Eq. (4). When the costs of test cases and the severities of faults are identical, Eq. (5) can reduce to Eq. (4).

4.2.2.3. TETC. When evaluating the test case execution adjustment efficiency, it is appropriate to use the Test Execution-Time Cost (TETC) as a metric. TETC denotes the total time required for executing the entire test sequence after pre-prioritization has been determined, including any necessary adjustments made by the prioritization adjustment algorithm during execution. The calculation of TETC is shown in Eq. (6):

$$TETC = t_{last} - t_{first}, \tag{6}$$

Where t_{first} is the initiation time of the test order and t_{last} refers to its end time.

4.3. Case study design

In order to verify the effectiveness of the proposed method, six types of information need to be collected, namely: requirements, test cases, T-R relationship matrix, clusters of test cases, coverage information and fault detection information.

In the three industrial systems, requirements, atomic test cases and the coverage relationship between requirements and atomic test cases can be obtained directly. The method described in Section 3.1 is used to build test cases and create T-R matrix. In the *XML-Security* system, test cases can be obtained beforehand. Due to the lack of requirement information, this research uses the names of test cases to create requirements and T-R relationship matrix. For example, the test case named "BlockEncryptionTest" can generate the requirement titled "Encryption algorithm".

In order to obtain the clustering of test cases, the clustering algorithm described in Section 3.2 is used to cluster test cases. When doing a comparison with other cluster-based TCP techniques, it is also necessary to use their clustering methods to collect their test case clusters.

The coverage information of all versions can be collected. Function coverage is used for *XML-Security*, and requirement coverage is used for other three industrial systems. Fault detected information can be obtained from the system provider.

After collecting all the required information, the TCP techniques in the previous section are used. The APFD value, APFDc value and TETC obtained by using each TCP technique are calculated. When calculating APFDc, the costs of test cases are assumed to be equal and the fault severity levels are assigned on a scale from 1 to 10. Table 6 shows the classification of fault severity. As there are random factors in the operation of some TCP techniques, each one is executed 10 times independently, and the average values are displayed in the experimental results. The similarity threshold of clustering process is set $\alpha = 50$ % in the first clustering. The step size of dynamic adjustment is set $Step = 1$ in the first four research questions and set $Step = 1$ and 2 in the fifth research question. All the experiments are conducted on the same computer which is configured as 64-bit windows 8 operating system, Intel (R) Core (TM) i3-2130 CPU and 4GB memory.

4.4. Results and analysis

4.4.1. RQ1: comparison with traditional TCP techniques

The APFD values of five CAP techniques (R-CAP, TC-CAP, AC-CAP, TAR-CAP and AAR-CAP) for the experimental subjects based on black-box testing (*ChipTest, SSP* and *CPMISS*) and white-box testing (*XML-Security*) are presented in Table 7 along with their corresponding improvement rates compared to traditional pre-prioritization techniques. The first column represents the version information of the system, and the first row represents the name of the CAP method and its selected pre-prioritization technique. For example, the APFD of R-CAP in version *CT-14* is 55.78 %, 10.26 % higher than that of R. The results indicate that *ChipTest* exhibits the most favorable experimental

Table 6
Classification of fault severity.

	Level	Description
Minor	1	The UI lacks esthetic appeal.
	2	The user experience is hindered by inconveniences. such as misspellings and other textual errors.
Major	3	The UI exhibits deficiencies in terms of frontend data validation and error handling, among other issues.
	4	The duration of the operation and response is unacceptably prolonged.
	5	The proper implementation of secondary functions is unattainable.
Critical	6	The complete preservation of the data is unattainable.
	7	The implementation of the main functions is not feasible.
	8	The occurrence of critical operations can result in system crashes, machine shutdown, and an infinite loop.
Fatal	9	Conventional operations can result in system crashes, machine failures, and infinite loops.
	10	The user's data is breached or financial assets are involved.

Table 7
APFD of five CAP techniques and their improvement rate over the corresponding traditional techniques (%).

		R-CAP APFD	over R	TC-CAP APFD	over TC	AC-CAP APFD	over AC	TAR-CAP APFD	over TAR	AAR-CAP APFD	over AAR
Black-box	CT-9	53.41	**6.21**	65.08	**7.69**	58.92	**7.07**	65.08	**2.44**	65.08	**2.44**
	CT-14	55.78	**10.26**	73.44	**8.13**	63.7	**10.15**	73.45	**1.45**	73.45	**1.45**
	SSP	48.36	**3.13**	68.5	**0.69**	62.51	**1.5**	68.5	**1.5**	68.5	**1.5**
	CS-1	47.12	**1.23**	60.47	**1.01**	56.31	**0.6**	58.67	**0.42**	58.67	**0.42**
	CS-2	40.27	**3.02**	49.07	**0.9**	51.35	**0.29**	63.01	**0.59**	65.36	**0.41**
	CS-3	42.56	**4.29**	55.49	**3.66**	53.53	**1.18**	67.45	**1.78**	71.31	**1.68**
	CS-4	50.21	**1.29**	51.34	**0.84**	52.4	**1.16**	59.25	**0.73**	61.76	**0.24**
	CS-5	56.11	**1.64**	72.54	**1.04**	65.57	**1.01**	81.24	**0.82**	77.78	**0.86**
White-box	XS-0	49.13	**10.11**	50.99	**6.68**	53.30	**4.3**	60.99	**1.84**	60.99	**1.84**
	XS-1	51.44	**7.79**	53.30	**8.11**	57.61	**9.21**	59.22	**6.42**	59.42	**11.23**

outcomes among the black-box testing systems. Additionally, the experimental impact of CAP in the white-box testing system is also notably significant.

Based on the findings presented in Table 7, four versions (*XML-Security* and *ChipTest*) that exhibit significant effects have been selected as test objects for APFDc testing. The APFDc values of five CAP techniques and their corresponding improvement rates over the traditional pre-prioritization techniques are presented in Table 8, which categorizes the results into black-box testing and white-box testing. The experimental results demonstrate that all CAPs exhibit significant improvements in APFDc values compared to their corresponding pre-prioritization techniques have remarkable effects, thereby indicating that CAP not only improves fault detection speed but also elevates the severity level of detected faults.

The rates of improvement in APFD and APFDc for five CAPs compared to their respective conventional counterparts are illustrated in Fig. 4. In the diagram, the vertical axis represents the percentage of improvement with a unit of %. The horizontal axis represents the five CAP techniques, with two bars displayed for each technique; the blue bar indicates the rate of improvement in APFD, while the red bar denotes the rate of improvement in APFDc. The results obtained for the four system version objects are illustrated in Fig. 4(a)–(d), respectively. The CT-9 and CT-14 test data are from black-box testing, whereas the XS-0 and XS-1 test data are from white-box testing.

From Tables 7, 8 and Fig. 4, it is evident that: 1) Regardless of the traditional TCP technique selected for pre-prioritization, CAP outperforms its corresponding traditional TCP in terms of APFD and APFDc (with an improvement rate greater than zero), which confirms the ability of CAP to enhance efficiency (APFD and APFDc). 2) Numerically speaking, TAR-CAP and AAR-CAP exhibit superior values for both APFD and APFDc compared to the other three combinations, indicating that TAR-CAP and AAR-CAP exhibit optimal performance in terms of fault detection speed rate while incorporating CAP does not compromise the benefits of the original TCP (as demonstrated by TAR and AAR outperforming R, TC, and AC as shown in [7]). 3) In the comparison of traditional coverage-based techniques, AC outperforms TC, so as AC-CAP outperforms TC-CAP. 4) Regardless of whether the test strategy selects black-box testing or white-box testing, the use of CAP is effective.

4.4.2. RQ2: comparison with cluster-based TCP techniques

According to the experimental results in the previous section, AC-CAP and AAR-CAP are selected as the representatives of CAP, and compared with other cluster-based TCP methods (Tcl-cm-random, Cov+OPC+Pur+Ran, Cov+Ran and ICP). The reason why AC-CAP is selected is that it and its opponent method (other cluster-based TCP) involve a coverage-based strategy. AAR-CAP is chosen as another representative because it has the highest fault-detection rate (with the highest APFD).

Table 9 shows the APFD results of six cluster-based TCP techniques. It can be seen that AAR-CAP is superior to the other cluster-based TCP techniques to be evaluated (except for its performance in *CS-1* and *CS-4*), and its APFD is between 58.67 % and 77.78 %.

For the four coverage-based techniques: Cov+OPC+Pur+Ran, Cov+Ran, ICP and AC-CAP, their performances in different systems and versions show different trends. In the case of *XML-Security*, the fault detection rate of AC-CAP is the fastest in version *XS-0* because it has the highest APFD compared with the other three techniques. However, in *XS-1*, ICP anti-super AC-CAP achieves the best effect. From the results for *CPMISS*, the two combined prioritization techniques (Cov+OPC+Pur+Ran and Cov+Ran) perform best, but the difference between them is not very obvious and the APFD difference is only about 0.08–0.53 %. In *ChipTest* system, Cov+Ran performs best in version *CT-9* but it is surpassed by AC-CAP in *CT-14* in terms of fault detection rate. In *SSP* system, AC-CAP has the highest APFD value and becomes the best technique. Additionally, AAR-CAP outperforms several other cluster-based TCPs, regardless of whether white-box testing or black-box testing strategy is employed.

In order to evaluate the statistical significance of the results obtained, the paired samples *t*-test is used to evaluate all the results in Table 9, and the statistical results are shown in Table 10. Before the *t*-test, a K-S test has been used to confirm normal distribution of the six groups of sample data and Bonferroni correction has been applied to correct the *p*-value of *t*-test, resulting in a significance level is $\alpha = 0.025$. If the *p*-value is less than or equal to the significance level ($p \leq 0.025$), the null hypothesis can be rejected and the alternative hypothesis can be accepted.

Based on the *t*-test results presented in Table 10, it can be concluded that, AAR-CAP e outperforms other techniques such as Tcl-cm-random, Cov+OPC+Pur+Ran, Cov+Ran and ICP (as evidenced by the *t*-value of AAR-CAP is greater than 0 and the *p*-value is less than 0.025), while AC-TCP demonstrates intermediate performance among the four alternative methods. Specifically, AC-CAP is significantly better than ICP ($p =$

Table 8
APFDc of five CAP techniques and their improvement rate over the corresponding traditional techniques (%).

		R-CAP APFDc	over R	TC-CAP APFDc	over TC	AC-CAP APFDc	over AC	TAR-CAP APFDc	over TAR	AAR-CAP APFDc	over AAR
Black-box	CT-9	46.808	**7.08**	58.26	**6.16**	64.384	**6.85**	65.09	**3.91**	**65.88**	**4.29**
	CT-14	48.22	**18.46**	50.92	**17.84**	55.672	**13.22**	62.56	**8.01**	**66.27**	**7.34**
White-box	XS-0	53.872	**8.23**	57.83	**6.97**	56.483	**5.88**	60.16	**6.9**	**60.44**	**5.47**
	XS-1	52.09	**8.82**	56.7	**6.47**	59.131	**4.26**	59.52	**3.52**	**61.06**	**5.17**

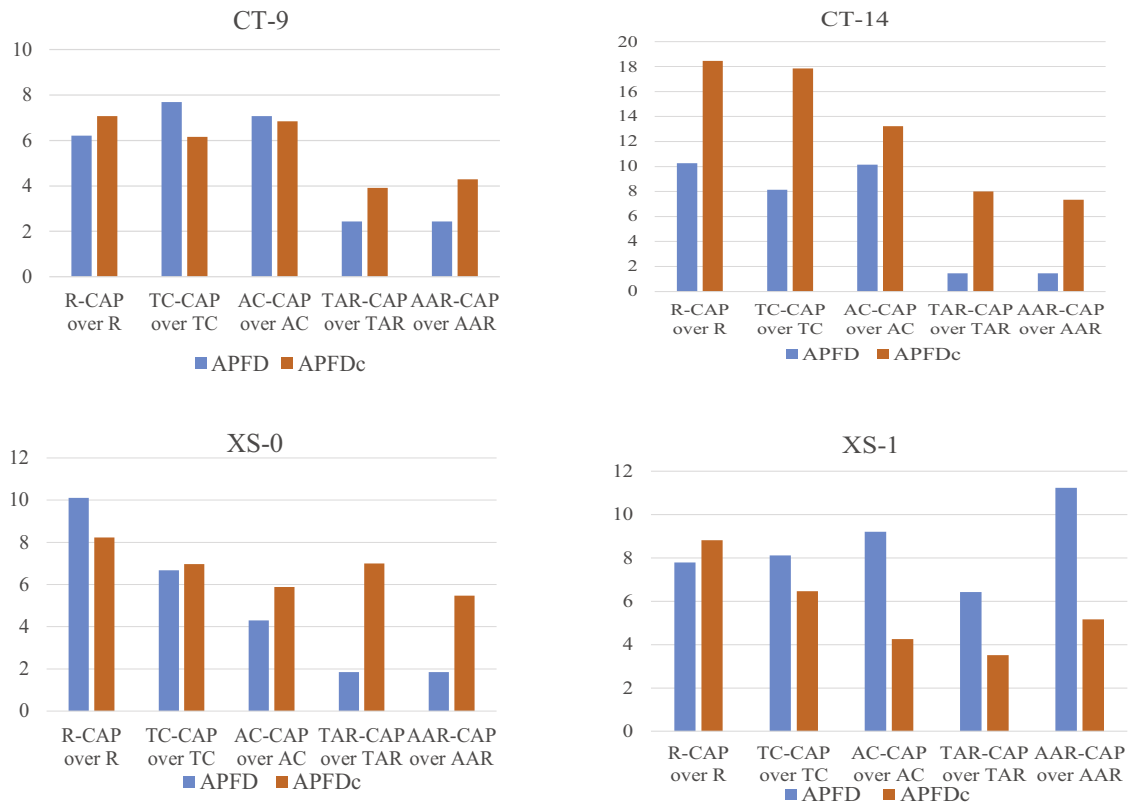


Fig. 4. Illustrates the improvement rates of APFD and APFDc for five CAPs relative to their respective conventional counterparts.

Table 9
APFD values for six cluster-based TCP techniques for 10 versions (%).

	Version	Tcl-cm-random	Cov+OPC+Pur+Ran	Cov+Ran	ICP	AC-CAP	AAR-CAP
White-box	XS-0	48.72	49.78	51.65	51.1	53.30	**60.99**
	XS-1	52.25	54.92	55.42	58.75	57.61	**59.42**
Black-box	CS-1	48.83	61.28	61.75	53.6	56.31	**58.67**
	CS-2	55.08	54.05	53.97	36.98	51.35	**65.36**
	CS-3	61.44	62.35	62.88	33.14	53.53	**71.31**
	CS-4	63.61	53.74	53.38	19.7	52.4	**61.76**
	CS-5	67.91	71.59	71.3	32.03	65.57	**77.78**
	CT-9	43.26	57.88	60.66	59.72	58.92	**65.08**
	CT-14	64.91	59.24	58.4	37.86	63.7	**73.45**
	SSP	49.21	60.52	62.02	49.99	62.51	**68.5**

Table 10
Statistical tests result from comparing the APFD values for AC-CAP and AAR-CAP to four opponent techniques.

	AC-CAP		AAR-CAP	
	p-value	*t*	*p*-value	*t*
Tcl-cm-random	0.491	0.718	0.001	5.234
Cov+OPC+Pur+Ran	0.494	−0.714	0.000	5.318
Cov+Ran	0.268	−1.180	0.001	4.627
ICP	0.009	3.345	0.002	4.251

0.009<0.025). Moreover, AC-CAP outperforms Tcl-cm-random (with a t value greater than 0), although the difference between them is not statistically significant ($p = 0.491 > 0.025$). However, when compared to the other two combination prioritization techniques (Cov+OPC+Pur+Ran and Cov+Ran), AC-CAP does not exhibit superiority over them ($t<0$). Nevertheless, it should be noted that this difference lacks statistical significance (since the p value is not less than 0.025).

Fig. 5 shows the boxplot diagram of the experimental results. The horizontal axis represents the cluster-based TCP techniques, and the vertical axis represents the APFD values. Each boxplot shows the median, upper/lower quartile and max/min APFD values achieved by a technique. The *dot* icon denotes median. Taking the median value as the standard analysis, AAR-CAP is significantly superior to other techniques, and its median point reaches the highest value of all techniques. ICP becomes the worst performing technique due to its lowest median point. Cov+OPC+Pur+Ran, Cov+Ran and AC-CAP have similar effects, and their median points locate between 50 % to 60 %. Based on the analysis of the concentration degree of APFD distribution (upper/lower quartile), the APFD distributions of Cov+OPC+Pur+Ran and Cov+Ran are the most concentrated, which converges roughly between 54 % and 63 %. The APFD distribution of AAR-CAP and AC-CAP is also relatively concentrated, and the convergence range is not more than 10 %. The APFD distributions of Tcl-cm-random and ICP are relatively scattered, and the convergence range is more than 15 %, especially ICP even exceeds 20 %.

M(Median, Q1, Q3) is used to denote the median, upper and lower

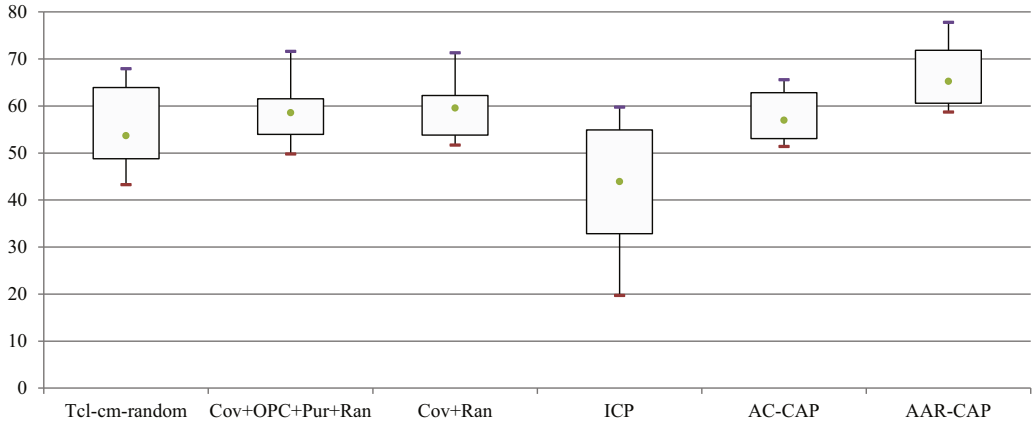


Fig. 5. Variation APFD values across all the versions for each of cluster-based TCP techniques.

quartiles of APFD for each technique. *M1-M6* represents six techniques: Tcl-cm-random, Cov+OPC+Pur+Ran, Cov+Ran, ICP, AC-CAP and AAR-CAP, respectively. Therefore, the results of six techniques are *M1(53.67, 48.80, 63.94), M2(58.56, 53.97, 61.55), M3(59.53, 53.82, 62.24), M4 (43.93, 32.86, 54.89), M5(56.96, 53.08, 62.81)* and ***M6(66.22, 60.60, 71.85)*** respectively, which clearly indicates that the overall performance of AAR-CAP is superior to other techniques, regardless of the median or upper/lower quartile. Table 11 shows the mean ranks for a pairwise test of six techniques using the Mann-Whitney test. In Table 11, a comparison of six techniques is presented in a pair-to-pair manner, resulting in 15 pairs of tests, and each group undergoes non- parametric tests of independent samples. For example, the average ranks of AC-CAP and Tcl-cm-random in group G1 are 11.6 and 9.4, respectively, indicating that AC-CAP outperforms Tcl-cm-random. Based on the pairwise comparisons, AAR-CAP demonstrates the highest level of effectiveness, followed by Cov+Ran, Cov+OPC+Pur+Ran, AC-CAP, Tcl-cm-random and ICP. The statistical analysis reveals that the selection of pre-prioritization plays a crucial role in achieving optimal effectiveness of CAP. When the pre-prioritization selects a well-executed AAR, the effectiveness of CAP is significantly enhanced, particularly in comparison to that with a relatively poorly pre-prioritized AC. The combination of this AAR and adaptive adjustment technology forms the AAR-CAP with optimal impact.

4.4.3. RQ3: comparison with the adaptive TCP technique

When compared with AP, AC-CAP and AAR-CAP are also selected as the representatives of CAP. In reference [3], $q = 0.8$ or $q = 0$ (q is a parameter in AP) may make AP produce satisfactory results, so in this evaluation, $q = 0$ is used as the parameter value. Both CAP and AP can adjust the order of test cases during the test execution, so it is significative to compare their TETC.

Table 12 reports on the APFD (in %) and TETC (in ms) measures collected from 10 executions at the prioritization process over the same system objects (10 versions), with AP, AC-CAP and AAR-CAP, respectively. Firstly, AAR-CAP is superior to the other techniques in the rate of fault detection. In addition, when comparing AC-CAP and AP, their APFD values have no significant difference in *XML-Security* and *SSP*. In some versions of *CPMISS* (*CS-2, CS-3* and *CS-5*), AP performs better than AC-CAP (APFD of AP is higher). However, in the other versions (*CS-1* and *CS-4*), AC-CAP outperforms AP. In *ChipTest*, the speed of fault detection of AC-CAP is a little faster than that of AP. Secondly, both AC-CAP and AAR-CAP have shorter TETC than AP. For example, in *CS-1*, TETC of AP is 119.57 ms, which is about 29 times of AC-CAP (4.08 ms) and 17 times of AAR-CAP (7.05 ms).

Fig. 6 shows the TETC of AP, AC-CAP and AAR-CAP in two experimental systems (*XS-0* and *CS-1*). The TETC value of AP is longer than that of the other two CAP techniques. Especially in large industrial project *CPMISS* (version 1), under 30 execution, the TETC of AP is all more than 100 ms, and the TETC of AC-CAP and AAR-CAP is all less than 20 ms. Therefore, although both AP and CAP can adjust the order of test cases during the test execution, AP takes more time to adjust test order than CAP.

Table 12
APFD values and time costs for AP, AC-CAP, and AAR-CAP (%/*ms*).

Version	AP		AC-CAP		AAR-CAP	
	APFD	TETC	APFD	TETC	APFD	TETC
XS-0	58.3	4.93	53.30	0.8	60.99	0.73
XS-1	56.79	6.35	57.61	1.34	59.42	1.85
CS-1	46.66	119.57	56.31	4.08	58.67	7.05
CS-2	63.65	72.51	51.35	2.87	65.36	3.82
CS-3	66.71	74.32	53.53	0.56	71.31	1.18
CS-4	51.53	97.81	52.4	1.24	61.76	2.01
CS-5	75.1	94.18	65.57	2.26	77.78	1.92
CT-9	47.94	2.66	58.92	0.52	65.08	0.54
CT-14	48.18	5.14	63.7	1.45	73.45	2.23
SSP	62.34	36.08	62.51	5.43	68.5	4.8

Table 11
Mean rank for a pairwise test of 6 techniques using the Mann-Whitney test.

No.	Technique	Mean Rank	No.	Technique	Mean Rank	No.	Technique	Mean Rank
G1	AC-CAP	**11.6**	G6	AAR-CAP	**13.8**	G11	Tcl-cm-random	9.5
	Tcl-cm-random	9.4		Tcl-cm-random	7.2		Cov+OPC+Pur+Ran	**11.5**
G2	AC-CAP	9.9	G7	AAR-CAP	**13.7**	G12	Tcl-cm-random	9.2
	Cov+OPC+Pur+Ran	**11.1**		Cov+OPC+Pur+Ran	7.3		Cov+Ran	**11.8**
G3	AC-CAP	9.8	G8	AAR-CAP	**13.5**	G13	Tcl-cm-random	**13**
	Cov+Ran	**11.2**		Cov+Ran	7.5		ICP	8
G4	AC-CAP	**13.8**	G9	AAR-CAP	**15.2**	G14	Cov+OPC+Pur+Ran	10.1
	ICP	7.2		ICP	5.8		Cov+Ran	**10.9**
G5	AC-CAP	7	G10	Cov+Ran	**14.3**	G15	Cov+OPC+Pur+Ran	**14.1**
	AAR-CAP	**14**		ICP	6.7		ICP	6.9

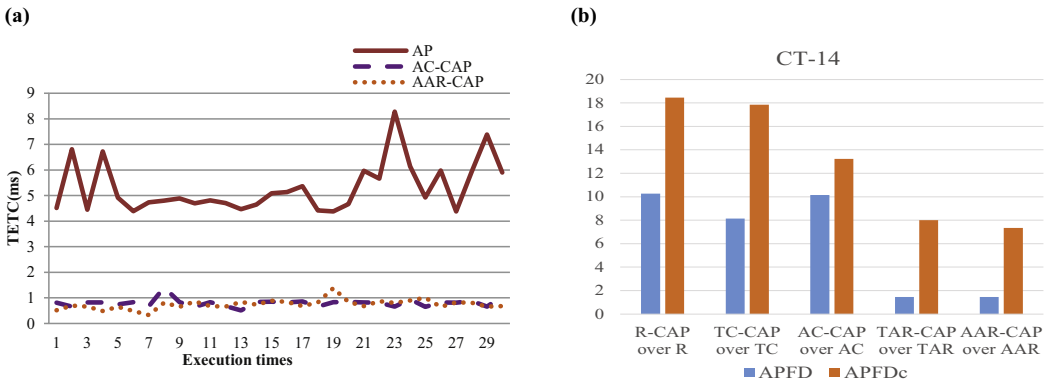


Fig. 6. TETC of AP, AC-CAP, and AAR-CAP over *XS-0* and *CS-1*.

4.4.4. RQ4: effects of dissimilarity metrics

This section discusses the influences of two *dissimilarity* metrics to the CAP methods. For the "correlation", we use the similarity definition rules in Section 3.2 is used to carry out the experiment. For the "distance", the 2-dimensional Euclidean distance is used to carry out the experiment, see Eq. (7). The steps of clustering based on "distance" are: first, the objects to be tested (test cases) are scattered randomly; Secondly, the iterative merge is carried out from the first test case according to the two-dimensional Euclidean distance between the two test cases. The total number of clusters after merging is consistent with the number of clusters obtained by using the "correlation" method.

$$d(t_i,t_j) = ED(t_i,t_j) = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2} \tag{7}$$

Table 13 reports on the APFD values collected from 30 executions at the prioritization process over the same object systems, with TC-CAP, AC-CAP, TAR-CAP and AAR-CAP, respectively, while varying *dissimilarity* metric ("distance" and "correlation"). It can be seen that the CAP methods under the "correlation" metric are substantially superior to the ones under the "distance" metric. Fig. 7 also shows this phenomenon. Fig. 7(a) is an integrated distribution for *dissimilarity* metrics and the APFD-value histogram of the CAP methods under the "correlation" is higher than those under the "distance". Fig. 7(b) shows a specific distribution based on each version of the object systems and we can also see that the solid line represented "correlation" is always higher than the dotted line represented "distance".

For the four CAP techniques: TC-CAP, AC-CAP, TAR-CAP and AAR-CAP, their performances in different priority strategies (total and additional) show different trends. Fig. 8 shows this phenomenon in detail. Fig. 8(a) shows that among the four CAP techniques, the effect of using the "correlation" metric (diagonal bar) is better than using the "distance" metric (pure color bar). Fig. 8(b) shows that when the CAP methods utilize the "distance" metric for their clustering process, the difference between the total strategy (dot column) and the additional strategy (wave column) is significant. That is to say, when "distance" is used as the *dissimilarity*, the CAP method itself choosing the additional strategy will be more effective. In Fig. 8(c), the difference between using "distance" (pure color column) and "correlation" (oblique line column) is significant for the four CAP methods, whether under the total strategy or the additional strategy, and this difference is more significant when the CAP method itself selects the total strategy than the additional strategy. That is to say, whether the total or additional strategy is selected, the CAP method will perform well in the clustering process using the "correlation" metric. And when the total strategy is selected, the CAP method will be more effective in clustering using "correlation".

4.4.5. RQ5: effects of different step values on CAP

This section discusses whether CAP has an impact on the APFD improvement rate of the corresponding traditional TCP technique under different step values (*Step*=1,2).

Figs. 9 and 10 show the box diagram of the experimental results of CAP techniques when implementing different *Step* adjustments. Among them, Fig. 9 carries out data statistics based on the system versions, that is, ten system versions on the horizontal axis; Fig. 10 carries out data statistics based on CAP techniques, that is, the horizontal axis is five kinds of CAP techniques. The vertical axis represents the APFD improvement rate. Each box represents the distribution of the APFD improvement rate under different *Step* values where × is the median.

In Fig. 9, based on the median analysis, the APFD improvement rate of *Step*=2 is better than that of *Step*=1 in each version, and their median points are higher than those of *Step*=1. Especially in *CT-14*, the median APFD improvement rate of *Step*=2 is as high as 17.08 %, which is much higher than the median 5.48 % of *Step*=1. In *CS-1, CS-2* and *CS-3*, the APFD improvement rate of *Step*=1 and *Step*=2 is not much different, all within 1 %, which means that in these three versions, whether the adjustment step size is 1 or 2, it will not have much impact on the APFD improvement rate effect.

In Fig. 10, based on the median analysis, among the five CAP techniques, the median APFD improvement rates of *Step*=2 are higher than

Table 13
APFD values at two dissimilarity methrics for TC-CAP, AC-CAP, TAR-CAP and AAR-CAP(%).

Ver.	TC-CAP		AC-CAP		TAR-CAP		AAR-CAP	
	dis.	cor.	dis.	cor.	dis.	cor.	dis.	cor.
XS-0	49.56	50.99	51.54	53.30	56.04	60.99	56.04	60.99
XS-1	50.24	53.30	54.79	57.61	60.67	59.22	60.83	59.42
CS-1	56.2	60.47	59.63	56.31	58.69	58.67	58.69	58.67
CS-2	48.9	49.07	51.21	51.35	62.43	63.01	64.27	65.36
CS-3	55	55.49	51.51	53.53	67.45	67.45	72.09	71.31
CS-4	49.07	51.34	50.2	52.4	57.82	59.25	62.45	61.76
CS-5	71.89	72.54	63.29	65.57	81.21	81.24	77.78	77.78
CT-9	56.55	58.57	56.96	58.92	63.14	65.08	63.14	65.08
CT-14	60.2	64.41	61.74	63.7	71.79	73.45	71.79	73.45
SSP	67.42	67.76	61.86	62.51	68.27	68.5	68.27	68.5

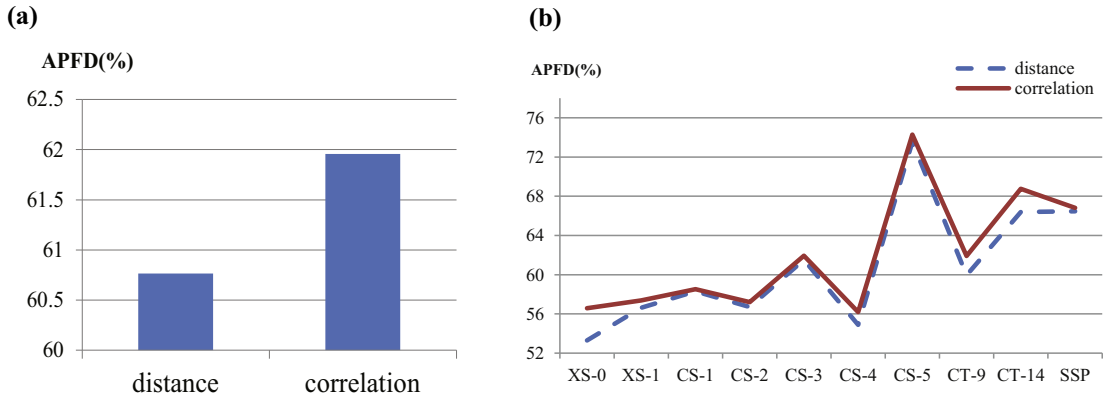


Fig. 7. The average APFD of different "dissimilarity" metrics on integrated distribution and specific distribution.

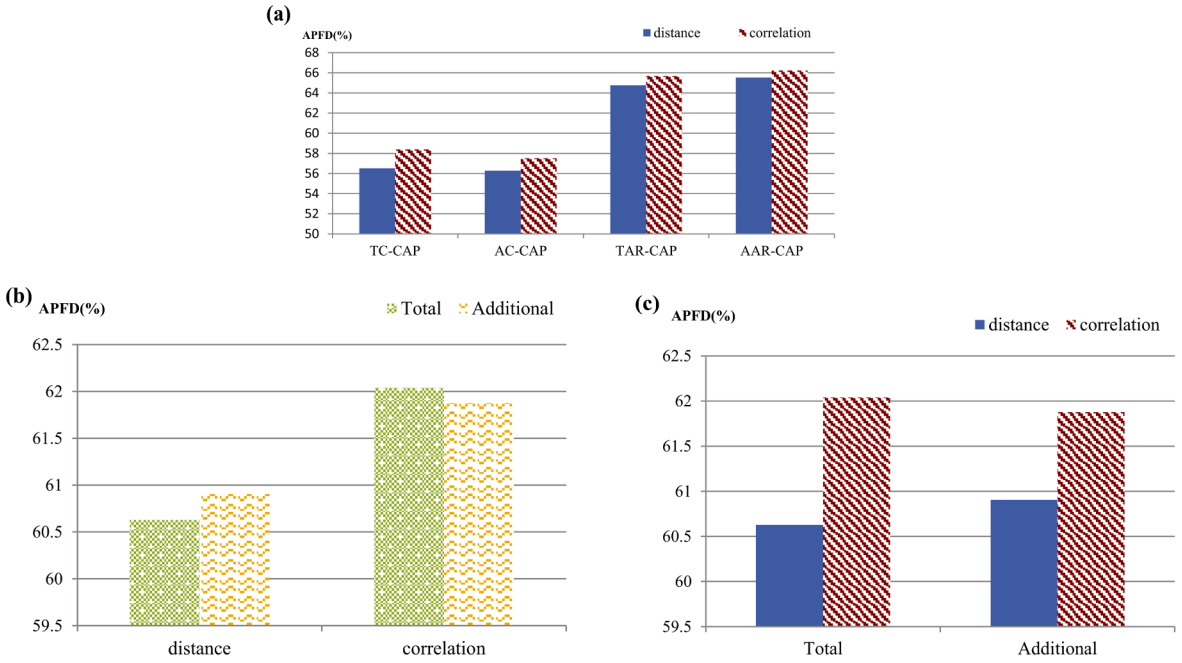


Fig. 8. The average APFD of different "dissimilarity" metrics at four CAP methods.

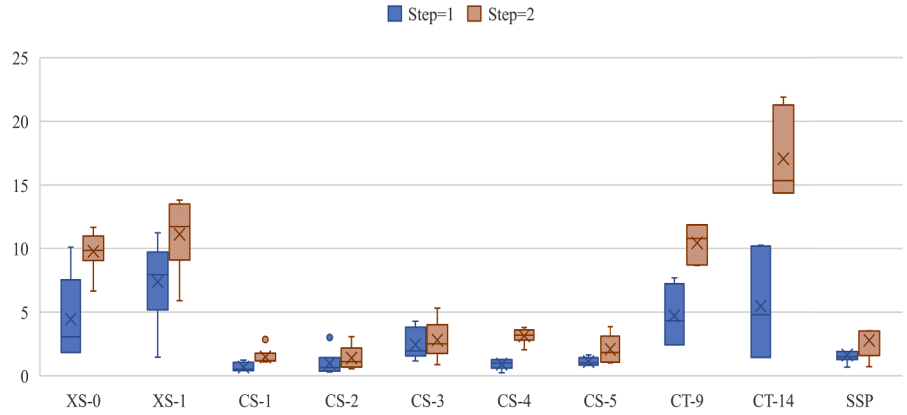


Fig. 9. Boxplots of the improvement rate of APFD compared with the corresponding traditional TCPs after the implementation of different Steps in CAP of different systems and versions.

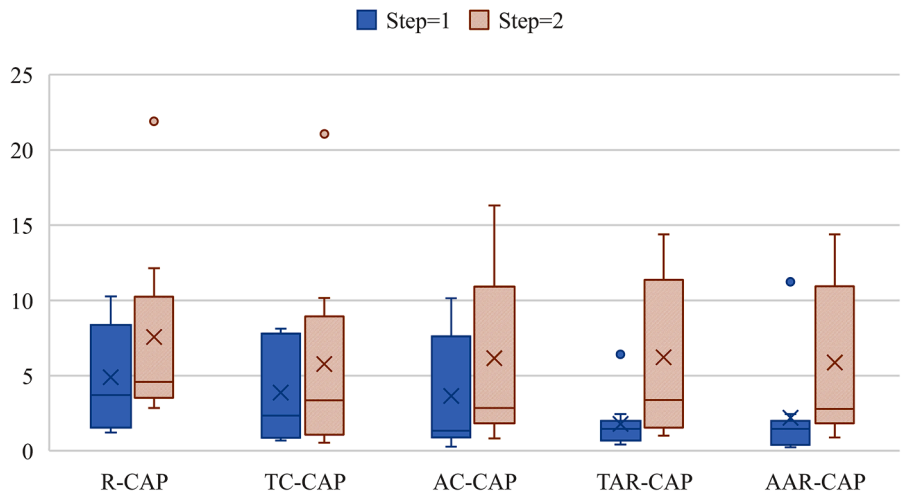


Fig. 10. Boxplots of the improvement rate of APFD compared with the corresponding traditional TCPs after the implementation of different *Steps* for five CAPs.

those of *Step*=1, which indicates that no matter which traditional TCP technique is selected for pre-prioritization, the APFD improvement effect of *Step*=2 for TCP adjustment step size is better than that of *Step*=1. Among them, the APFD improvement rate of TAR-CAP compared with TAR reaches the maximum difference when *Step* is 1 and 2 respectively, and *Step*=2 is better than *Step*=1. That is, when TAR is selected as the pre-prioritization technique and *Step*=2 is selected as the adjustment step size, the test prioritization effect of CAP can reach the best. In addition, the difference of APFD improvement rate of TC-CAP under different *Steps* is the smallest, but it is also close to 2 %, that is, when TC is selected as the pre-prioritization technique, *Step*=2 is still better than *Step*=1 to obtain the better prioritization effect.

Taking AAR-CAP as the representative analysis of CAP, Fig. 11 shows the comparative distribution of the APFD improvement rate of AAR-CAP compared with AAR, after AAR-CAP adopted different prioritization adjustment *Step* (*Step*=1,2) experiments in the test execution. Fig. 11(a) shows the distribution based on different versions of different systems. It can be seen that *Step*=2 outperforms *Step*=1 in the ten versions selected, except in *CS-3*. In *CT-14*, the APFD improvement rate of *Step*=2 reaches the maximum value, which is 14.38 %, indicating that when the system object is *CT-14*, selecting *Step*=2 for prioritizing adjustment can achieve the best prioritization effect. However, in *XS-1*, the difference between the APFD improvement rate of *Step*=2 and *Step*=1 is small, about 0.1 %, that is, when the system object is *XS-1*, the selection of *Step*=1 or 2 has little impact on AAR-CAP itself. Fig. 11(b) shows the average comparison of APFD improvement rate. the APFD improvement rate of *Step*=2 is significantly higher than that of *Step*=1, indicating that AAR-CAP has better effect on the APFD improvement rate than AAR when the adjustment step size is *Step*=2.

4.5. Discussion and implications

The aforementioned experimental results demonstrate that CAP can enhance the effectiveness of test case prioritization.

First of all, irrespective of the traditional TCP employed as a pre-prioritization technique, CAP demonstrates superior performance compared to its corresponding traditional TCP. This verifies that CAP can improve the performance of its corresponding traditional TCP. Although there were slight variations in the results obtained from different pre-prioritization techniques, the only differences observed were in terms of degrees of superiority. Among several CAPs, AAR-CAP exhibits the highest fault-detection speed rate. In addition, regardless of whether comparing single or hybrid cluster-based TCP techniques, CAP outperforms them in terms of fault-detection speed rate. The potential explanations are as follows: Firstly, the prioritization scope of CAP lies in test execution, with more refined regulation sorting superimposed on the basis of the traditional TCP. As long as the clustering method is accurate, CAP will undoubtedly outperform traditional pre-prioritization techniques. Therefore, CAP is not in conflict with the traditional TCP method; instead, it can leverage the superior traditional TCP technique as a pre-prioritization mechanism. Secondly, the cluster-based TCP approach is essentially the traditional TCP approach and, in theory conforms to the first explanation as well. CAP responds more promptly to test prioritization by dynamically adjusting the test sequence during execution, resulting in a naturally superior performance compared to cluster-based TCP. Further investigation is

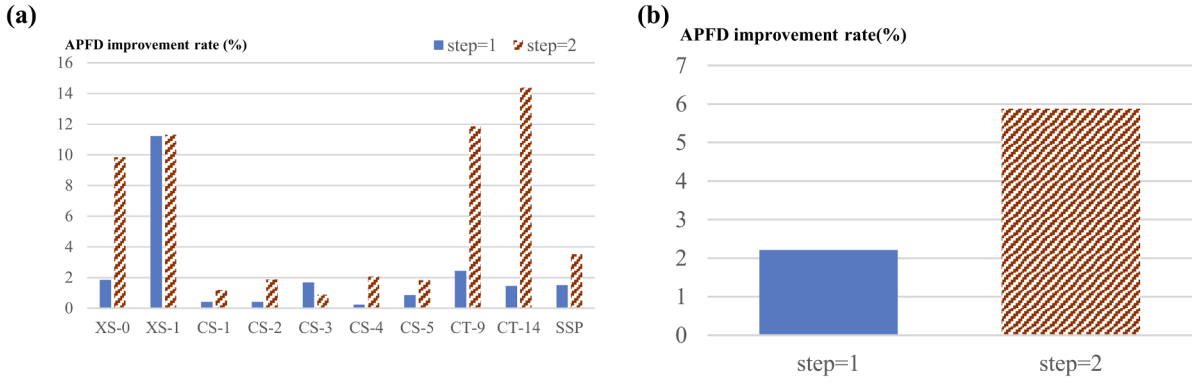


Fig. 11. The improvement rate of APFD of AAR-CAP compared with the corresponding AAR under different *steps*.

necessary to determine whether a cluster-based TCP superimposed CAP outperforms AAR-CAP in terms of performance.

Secondly, the majority of experimental subjects in this study underwent a black-box testing procedure, during which black-box test data were collected. The CAP algorithm demonstrates strong performance on these subjects. *XML-Security* is the only system subject based on white-box testing in this paper, while CAP also demonstrates effective performance on *XML-Security*. For example, in Table 7, the APFD of R-CAP in version *XS-0* is 49.13 %, which represents a significant improvement of 10.11 % compared to that of R. The analysis of the CAP algorithm principle reveals that it is designed on the basis of black-box testing (referring to the fundamental definition in Section 3). The potential factors contributing to the successful performance of CAP on a white-box testing system are as follows: 1) The "requirement" in the experiment process has been finely-grained to the function, enabling effective utilization of the CAP Algorithm 2) The clustering and adaptive adjustment of CAP can be achieved as long as there exists a relationship between test cases and requirements, which is also present in the white-box testing system. The aforementioned statement provides valuable insights for the future implementation of CAP. When applying CAP to the white-box testing strategy, appropriate modifications can be made to the parameter definition in order to enhance the testing objectives, thereby facilitating faster fault localization.

Thirdly, upon analyzing the results obtained from APFD and TETC measurements, it is evident that CAP outperforms AP in terms of fault-detection speed rate and overhead reduction. AAR-CAP outperforms AP in terms of fault detection speed, and AC-CAP and AAR-CAP exhibit lower test-execution time costs than AP. One possible explanation is that the AP method recalculates the priority of remaining test cases after each execution of a test case, resulting in a total number of calculations equal to the number of test cases. In contrast, the CAP method only adjusts the priority of unexecuted test cases when faults are detected, resulting in significantly fewer calculations.

Fourthly, for the *dissimilarity* metrics in the clustering process, the CAP method utilizing “correlation” clustering outperforms that utilizing “distance” clustering. In particular, when employing the total strategy for test case prioritization, the CAP method utilizing “correlation” clustering proves to be more efficacious. The possible explanation is that the “correlation” clustering method clusters test cases based on their correlation characteristics, effectively revealing the occurrence of faults and the spread of faults in the similar locations. The “distance” clustering method relies solely on the distance between test cases, disregarding the business correlation characteristics of test cases. Therefore, the resulting clusters generated by this method are less effective in quickly detecting faults compared to those generated by the “correlation” clustering method.

Fifthly, for the different adjustment *Step*, when the CAP method utilizes *Step*=2 for prioritization adjustments, the improvement rate of APFD over the corresponding traditional TCP is superior to that achieved by using *Step*=1. Especially, when TAR is employed as the pre-prioritization technique of CAP, *Step*=2 exhibits a significantly superior effect on test prioritization to *Step*=1. In other words, the optimal performance of TAR-CAP can be achieved in *Step*=2.

4.6. Limitations

First, the experimental results exhibit diversity across different systems, with CAP demonstrating a comparatively high improvement rate in *XML-Security* and *ChipTest* when compared to their corresponding traditional TCP counterparts, but a relatively low improvement rate in *CPMISS* and *SSP*. Upon analysis of the characteristics of *XML-Security* and *ChipTest*, it has been observed that firstly (condition 1), both systems exemplify version iterations, with each iteration introducing new functionality (approximately 50 %+). Secondly (condition 2), the number of test cases detecting faults in each iteration is less than 50 % of the total number of test cases, indicating a high-quality system development. Thirdly (condition 3), fault occurrences are not repeated due to effective repairs after a fault occurs in previous versions. Finally (condition 4), faults are evenly distributed across test cases that detect them. Conversely, *CPMISS* and *SSP* exhibit different characteristics. For *CPMISS*, although it has iterative versions, minimal new features are introduced in each iteration (not satisfying condition 1); The number of fault-detected test cases exceeds 50 % in some versions (not meeting condition 2), and these faults have been observed repeatedly in different versions (not fulfilling condition 3); Certain special test cases detect a relatively large number of faults (not meeting condition 4). As for *SSP*, it lacks iterative versions and bug fixes (conditions 1 and 3 are not met); although the number of fault-detected test cases does not exceed 50 % of the total count (meeting condition 2), the faults are not evenly distributed among them (mot satisfying condition 4). This analysis highlights the importance of high-quality system development (with a low ratio of fault-detection test cases and no recurring faults), iterative incorporation of new features (e.g., agile-based development) and achieving an even distribution of faults among fault-detected test cases to enhance the utilization of CAP within a system. In case the system development quality is inadequate, it is advisable to initially enhance the system quality through traditional and simple TCP methods before employing CAP.

Secondly, upon conducting a thorough analysis of fault types, the faults detected by CAP may indicate the same error or multiple errors of the same type. While this feature facilitates pinpointing the location of faults, it may not be optimal for detecting diverse faults. Therefore, if only concerned with the quantity of faults rather than diversity of fault types, CAP is suitable to detect more faults promptly. However, if aiming to identify multiple types of faults as soon as possible, limitations may arise when utilizing CAP.

Third, the experimental results revealed that the dissimilarity calculated using the “correlation” method outperforms that based on the Euclidean distance. However, it should be noted that in this study, the term "correlation" solely refers to the correlation between test cases and requirements. A more refined calculation of the “correlation” could potentially yield different and improved outcomes. In the subsequent research, we intend to incorporate the fault similarity factor into the calculation of the "correlation", thereby exploring the varied effects that different calculation methods may produce.

4.7. Threats to validity

This section discusses some of the potential threats to the validity of our study.

4.7.1. Threats to internal validity

One internal validity is the assignment of the clustering threshold α, which depends on each system’s unique characteristics. In general, a higher value for α leads to greater similarity among individuals within the same cluster. This study employs the median method to set α at 50% to mitigate this threat.

4.7.2. Threats to external validity

The threats to external validity mainly come from the experimental subjects, experimental data and faults used in the experiment. To mitigate the threat posed by the experimental subjects, one SIR open source test program and three industrial systems are utilized as the experimental subjects. Although the experimental subjects selected are still limited and the obtained results cannot represent all types of system tests, open source SIR objects greatly contribute to experiment fairness. The industrial project systems come from various field scenarios, enabling practical application testing of research content, which is also highly illustrative. The second threat arises from the test data of different subjects, which is relatively authentic but incomplete for this study. For incomplete data (such as missing requirements in SIR), supplementary data was simulated based on the scenario. The third threat

lies in faults which were mitigated by utilizing real faults from object systems during experiments.

5. Related work

The TCP problem was initially defined by the Elbaum and Rothermel team [4]. In its early stages, they primarily focused on white-box testing of TCP [2,4,5,6,14,15], with a typical example being coverage-based TCP, which prioritizes test cases based on program coverage and utilizes a greedy algorithm for ordering execution. They have proposed two metrics, namely APFD and APFDc [4,14], to evaluate the effectiveness of TCP based on whether fault severity and test case execution overhead are taken into consideration. Luo et al. [16] analyzed the representative degree of mutation faults and real faults through a mutation-based approach, and conducted an empirical investigation on the efficacy of TCP techniques under real faults and mutation faults. Huang et al. [17] introduced a novel criterion for code composition coverage and proposed a TCP technique based on this criterion, known as CCCP. Mondal and Nasre [18] focused on a code-coverage based regression test-prioritization solution (Colosseum), which prioritizes test-cases with smaller overall displacements and executes them early in the regression test-execution cycle. Yaraghi et al. [19] focused on ML-based TCP and have carefully defined a comprehensive set of features aimed at predicting the probability of regression failure, based on related studies, and a data model that captures entities and their relations in a typical CI environment. In recent years, TCP research based on white-box testing has oriented towards deep neural networks. Mu et al. [20] proposed a multi-objective deep learning test optimization method for test optimization in the field of deep learning, which provides a basis for test sequencing optimization in the field of artificial intelligence. Huang et al. [21] summarized the security of deep neural networks from the aspects of verification and testing. Kim et al. [22] studied the test adequacy of deep neural networks and proposed a fine-grained test adequacy framework SADL. TCP under white-box testing utilizes source code and structured information to sort test cases. Although the effect is remarkable, the workload is huge and the cost is expensive.

While white-box testing is more effective, black-box testing is less expensive and easier to execute. Some data [23] shows that the difference in the average percentage value of fault disclosure of TCP technology under the classification based on white box and black box test is only 2 % to 4 %. Srikanth et al. [24] proposed PORT, a system-level requirement prioritization testing algorithm. This approach computes the priority of test cases based on the mapping relationship between requirements and test cases, and arranges them in a value-oriented manner. Saraswat et al. [25] summarized test case prioritization and optimization techniques. Fan et al. [26] prioritized test cases based on key test case extraction. Butool et al. [27] prioritize test cases according to the complexity of coverage requirements. Considering comprehensively, our CAP approach primarily employs black-box testing to examine the test subjects.

As the number of regression testing rounds and obtained test data increase, investigating the impact of historical information on TCP from a resource perception perspective has become a research hotspot, and TCP enters the stage of regression TCP. Kim and Porter [28] utilized historical data generated from regression testing to prioritize test cases based on resource awareness. Eghbali et al. [29] used a set of defect prediction models to guide TCP technology and reorganized test cases based on the likelihood of covering error-prone code units, effectively addressing the issue of redundant test cases coverage for identical code behavior. He and Li et al. [30] used reinforcement learning to optimize continuous integration testing in a continuous integration environment, with further exploration into the design of reward functions and strategies. Jackson and Silvia [31] proposed a TCP approach called COLEMAN to make TCP problems work better in continuous integration environments. Our CAP method also considers the specificities of regression testing in the selection of pre-prioritization methods, and chooses TAR and AAR as the pre-prioritization methods from previous research.

Empirical observations indicate that numerous test cases exhibit similar behavior. Dickinson et al. [9] employed cluster analysis of execution profiles to detect faults in operation execution. The results show that the cluster-based filtering procedure is more effective than simple random sampling in identifying faults within the operation execution population. Miranda et al. [32] introduced a series of fast TCP technologies to find similarities by borrowing algorithms commonly used in the big data field, and to provide scalable similarity-based test case prioritization in white box and black box testing. Sarika et al. [33] analyzed and compared various clustering methods utilized in TCP, with the density-based K-means clustering (DBK-means) method demonstrating superiority over other techniques by achieving the highest fault detection rate. Jinfu et al. [34] proposed an adaptive random sequence method based on clustering using black box information. This method uses clustering technology to build test sequence, making neighbor test cases of each test case as dissimilar as possible. The aforementioned clustering methods all cluster test cases prior to test execution. The selection prioritization principle of test cases aims to cover as many distinct clusters as possible. Our CAP approach distinguishes itself from others by dynamically adjusting the order of subsequent test cases upon detecting faults during execution. Once the faults clearly identified, the likelihood of similar faults occurring within the same cluster will increase. Therefore, CAP will elevate the priority of other test cases in that cluster with hopes of locating any potential errors earlier.

6. Conclusions and future work

To address the issue of lagging utilization of fault-detected information during test execution, this paper proposes a cluster-based adaptive test case prioritization approach (CAP). CAP not only incorporates the clustering analysis methods but also allows for direct use of fault-detected information in the current regression cycle to adaptively adjust the order of test cases. This paper provides a detailed description of the design and algorithm of CAP, and its efficiency is validated through comparison with other TCP techniques.

The experimental results demonstrate that CAP outperforms other traditional TCPs, cluster-based TCPs, and another adaptive TCP in terms of fault detection efficiency within the same time frame. Specifically, among the five CAP techniques (R-CAP, TC-CAP, AC-CAP, TAR-CAP and AAR-CAP), AAR-CAP exhibits superior performance. Due to the difference in execution time domain between traditional TCP and CAP (traditional TCP is performed before test execution, while CAP is performed during test execution), integrating them can lead to higher efficiency and lower cost by selecting more effective traditional TCP as a pre-prioritization technique in the future.

In order to assess the performance of the CAP method, a specialized comparison was conducted on the *dissimilarity* metrics (utilized in the clustering process). The results indicate that under the “correlation” metric, CAP exhibits superior performance. Additionally, if the total strategy is selected for prioritization within CAP, “correlation” will play a significant role. Secondly, the *Step* size for prioritization adjustment was compared across different values (*Step*=1,2). Through experiments, it has been found that when *Step*=2, CAP exhibits a higher APFD improvement rate compared to its corresponding traditional TCP than in the case of *Step*=1.

Benefiting from the effectiveness of the experiments, it is a potential future research direction to investigate diverse clustering analysis methods and adaptive adjustment techniques during test case execution. Furthermore, exploring how to set thresholds more accurately and examining the impact of different thresholds on test results are also worthy of further investigation.

CRediT authorship contribution statement

Xiaolin Wang: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft. **Sulan Zhang:** Writing – review & editing, Supervision.

Declaration of Competing Interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Xiaolin Wang reports financial support was provided by the Natural Science Foundation of Zhejiang Province of China.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the Natural Science Foundation of Zhejiang Province under grant No. LQ22F020004.

References

[1] G. Rothermel, M.J. Harrold, A safe, efficient regression test selection technique, ACM Trans. Softw. Eng. Methodol. 6 (2) (1997) 173–210.
[2] S. Elbaum, A.G. Malishevsky, G. Rothermel, Test case prioritization: a family of empirical studies, IEEE Trans. Softw. Eng. 28 (2) (2002) 159–182.
[3] D. Hao, X. Zhao, L. Zhang, Adaptive test-case prioritization guided by output inspection, in: Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference, Kyoto, Japan, 2013, pp. 169–179.
[4] G. Rothermel, R.H. Untch, C. Chu, et al., Test case prioritization: an empirical study, in: Proceedings of the 1999 IEEE International Conference on Software Maintenance, Oxford, UK, 1999, pp. 179–188.
[5] S. Elbaum, A.G. Malishevsky, G. Rothermel, Prioritizing test cases for regression testing, in: Proceedings of the ACM SIGSOFT 2000 International Symposium on Software Testing and Analysis, Portland, United states, 2000, pp. 102–112.
[6] G. Rothermel, R.H. Untch, C. Chu, et al., Prioritizing test cases for regression testing, IEEE Trans. Softw. Eng. 27 (10) (2001) 929–948.
[7] X. Wang, H. Zeng, History-based dynamic test case prioritization for requirement properties in regression testing, in: Proceedings of the International Workshop on Continuous Software Evolution and Delivery, Austin, United states, 2016, pp. 41–47.
[8] S. Yoo, M. Harman, P. Tonella, et al., Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge, in: Proceedings of the International Symposium on Software Testing and Analysis, Chicago, USA, 2009, pp. 201–212.
[9] W. Dickinson, D. Leon, A. Podgurski, Finding failures by cluster analysis of execution profiles, in: Proceedings of the 23rd International Conference on Software Engineering, Toronto, Canada, 2001, pp. 339–348.
[10] D. Leon, A. Podgurski, A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases, in: Proceedings of the 14th International Symposium on Software Reliability Engineering, Denver, United States, 2003, pp. 442–453.
[11] W. Dickinson, D. Leon, A. Podgurski, et al., Pursuing failure: the distribution of program failures in a profile space, ACM SIGSOFT Softw. Eng. Notes 26 (5) (2001) 246–255.
[12] M.J. Arafeen, H. Do, Test case prioritization using requirements-based clustering, in: Proceedings of the IEEE 6th International Conference on Software Testing, Verification and Validation, Luxembourg, Luxembourg, 2013, pp. 312–321.
[13] H. Do, S. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact, Empir. Softw. Eng. 10 (4) (2005) 405–435.
[14] S. Elbaum, A. Malishevsky, G. Rothermel, Incorporating varying test costs and fault severities into test case prioritization, in: Proceedings of the 23rd International Conference on Software Engineering, Toronto, Canada, 2001, pp. 329–338.
[15] S. Elbaum, G. Rothermel, S. Kanduri, et al., Selecting a cost-effective test case prioritization technique, Softw. Qual. J. 12 (3) (2004) 185–210.
[16] Q. Luo, K. Moran, D. Poshyvanyk, et al., Assessing test case prioritization on real faults and mutants, in: Proceedings of the International Conference on Software Maintenance and Evolution, Madrid, SPAIN, 2018, pp. 240–251.
[17] R. Huang, Q. Zhang, D. Towey, et al., Regression test case prioritization by code combinations coverage, J. Syst.Softw. 169 (2020), 110712.
[18] S. Mondal, R. Nasre, Colosseum colosseum: regression test prioritization by delta displacement in test coverage, IEEE Trans. Softw. Eng. 48 (10) (2022) 4060–4073, https://doi.org/10.1109/TSE.2021.3111169.
[19] A.S. Yaraghi, M. Bagherzadeh, N. Kahani, L.C. Briand, Scalable and accurate test case prioritization in continuous integration contexts, IEEE Trans. Softw. Eng. 49 (4) (2023) 1615–1639, https://doi.org/10.1109/TSE.2022.3184842.
[20] Y.Z. Mu, Z. Wang, X. Chen, et al., A deep learning test optimization method using multi-objective optimization, Ruan Jian Xue Bao/J. Softw. (in Chinese). 33 (7) (2022) 2499–2524.
[21] X. Huang, D. Kroening, W. Ruan, et al., A Survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability, Comput. Sci. Rev. 37 (2020), 2020.100270.
[22] J. Kim, R. Feldt, S. Yoo, Guiding deep learning system testing using surprise adequacy, in: Proceedings of the 41st International Conference on Software Engineering, IEEE, 2019, pp. 1039–1049.
[23] C. Henard, M. Papadakis, M. Harman, et al., Comparing white-box and black-box test prioritization, in: Proceedings of the 2016 IEEE/ACM 38th IEEE International Conference on Software Engineering, Austin, United states, 2016, pp. 523–534.
[24] H. Srikanth, S. Banerjee, L. Williams, et al., Towards the prioritization of system test cases, Softw. Test. Verif. Reliab. 24 (4) (2014) 320–337.
[25] P. Saraswat, A. Singhal, A. Bansal, A review of test case prioritization and optimization techniques, Softw. Eng. 731 (2019) 507–516.
[26] S.P. Fan, L. Wan, N.M. Yao, et al., Test case sorting method based on key use cases extracted, Acta Electon. Sin. 50 (1) (2022) 149–156, in Chinese.
[27] R. Butool, A. Nadeem, M. Sindhu, et al., Improving requirements coverage in test case prioritization for regression testing, in: Proceedings of the 22nd International Multitopic Conference (INMIC), IEEE, 2019, pp. 1–6.
[28] J.M. Kim, A. Porter, A history-based test prioritization technique for regression testing in resource constrained environments, in: Proceedings of the 24th International Conference on Software Engineering, 2002, pp. 119–129.
[29] S. Eghbali, V. Kudva, G. Rothermel, et al., Supervised tie breaking in test case prioritization, in: Proceedings of the 41st IEEE/ACM International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2019, pp. 242–243.
[30] L.L. He, Y. Yang, Z. Li, et al., Reward of reinforcement learning of test optimization for continuous integration, J. Softw. 30 (5) (2019) 1438–1449, in Chinese.
[31] J.A. d. Prado Lima, S.R. Vergilio, A multi-armed bandit approach for test case prioritization in continuous integration environments, IEEE Trans. Softw. Eng. 48 (2) (2020) 453–465, https://doi.org/10.1109/TSE.2020.2992428.
[32] B. Miranda, E. Cruciani, R. Verdecchia, et al., Fast approaches to scalable similarity-based test case prioritization, in: Proceedings of the 40th International Conference on Software Engineering, Gothenburg, SWEDEN, 2018, pp. 222–232.
[33] S. Chaudhary, A. Jatain, Performance evaluation of clustering techniques in test case prioritization, in: 2020 International Conference on Computational Performance Evaluation (ComPE), IEEE, 2020, pp. 699–703.
[34] Jinfu, Lili Chen, et al., Test case prioritization for object-oriented software: an adaptive random sequence approach based on clustering, J. Syst. Softw. 135 (2018) 107–125.