Clustering Analysis of Function Call Sequence for Regression Test Case Reduction

Rongcun Wang*, ‡, Rubing Huang†§, Yansheng Lu*¶ and Binbin Qu*||

*School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan 430074, China

†School of Computer Science and Telecommunication Engineering
Jiangsu University, Zhenjiang, Jiangsu 212013, China
‡rcwang@hust.edu.cn
§rbhuang@mail.ujs.edu.cn
¶lys@hust.edu.cn
||qubinbindust@gmail.com

Received 10 October 2013
Revised 17 November 2013
Accepted 25 April 2014

Regression test case reduction aims at selecting a representative subset from the original test pool, while retaining the largest possible fault detection capability. Cluster analysis has been proposed and applied for selecting an effective test case subset in regression testing. It groups test cases into clusters based on the similarity of historical execution profiles. In previous studies, historical execution profiles are represented as binary or numeric function coverage vectors. The vector-based similarity approaches only consider which functions or statements are covered and the number of times they are executed. However, the vector-based approaches do not take the relations and sequential information between function calls into account. In this paper, we propose cluster analysis of function call sequences to attempt to improve the fault detection effectiveness of regression testing even further. A test is represented as a function call sequence that includes the relations and sequential information between function calls. The distance between function call sequences is measured not only by the Levenshtein distance but also the Euclidean distance. To assess the effectiveness of our approaches, we designed and conducted experimental studies on five subject programs. The experimental results indicate that our approaches are statistically superior to the approaches based on the similarity of vectors (i.e. binary vectors and numeric vectors), random and greedy function-coverage-based maximization test case reduction techniques in terms of fault detection effectiveness. With respective to the cost-effectiveness, cluster analysis of sequences measured using the Euclidean distance is more effective than using the Levenshtein distance.

Keywords: Test case reduction; cluster analysis; vector; sequence; cost-effectiveness.

Corresponding author.
1. Introduction

Regression testing is a time-consuming and very expensive activity. The activity accounts for as much as half of the costs of software maintenance [1]. With the everlasting evolution of software, the size of test suite increases over time. Generally speaking, regression testing is executed under resources constraints (e.g. time and human resources). Rerunning all existing test cases in the original test suite usually causes the failure of delivery, so it is infeasible in practice. Instead of this, only a part of test cases is executed. A potential risk is that reduced test suites might decrease the fault detection capability [2]. In order to improve the effectiveness of regression testing, various regression testing techniques have been proposed, such as test case reduction [3, 4], test case selection [5], and test case prioritization [6]. This study mainly focuses on test case reduction in regression testing phase. The technique aims to temporarily select a representative subset from the original test pool according to some criteria, while retaining the fault detection capability as much as possible. In this sense, test case selection is similar to test case reduction. But, the majority of selection techniques are modification-aware, i.e. these techniques mainly focus on the identification of the modified parts of the program under test [7].

Up-to-date, many test case reduction techniques have been proposed and experimentally studied [2–5]. Most of these techniques select a subset of test cases from the original test pool according to some specific coverage criteria (e.g. path [2], statement [8], function [9], and def-use [10]). In practice, a test suite usually consists of some test cases designed not for code coverage, but for exercising production features, exceptional behaviors, or specification items [11]. For this reason, coverage-based test case reduction techniques are not always effective. More recently, cluster analysis has been applied to regression testing and observation-based testing as an efficient method [12–14]. Cluster analysis groups test cases into clusters based on the similarity of the historical execution profiles exercised by test cases. The result of cluster analysis is that test cases in a cluster are similar to one another and different from the objects in other clusters. Test cases with the similar execution profiles will be expected to be clustered into the same groups in which test cases have the similar fault detection capability. Then, the representative test cases are extracted from each cluster based on a certain sampling strategy [13]. Finally, the performance of the reduced test suite is evaluated by a certain measure.

In previous studies on clustering analysis of test cases, historical execution profiles are represented as function coverage vectors (FCV), such as numeric vectors [12, 13, 15] or binary vectors [16, 17]. Those test cases with similar execution profiles are clustered into the same group. The function-coverage-vector-based similarity methods effectively reflect the dynamic behavior of test case execution. However, they only consider whether a function or statement is executed and the number of times a function or statement is exercised. They do not consider additional important information, i.e. the relations and sequential information between function calls. In this case, the vector-based similarity methods may not always generate an efficient
subset of test cases. In fact, some faults are really sensitive to sequential information and the call relations between function calls. The vector-based methods cannot well distinguish those tests related to sequential information and the relations between function calls.

In this paper, we propose cluster analysis of function call sequences (FCS) measured using the Euclidean distance for test case reduction, which can be used to produce a temporary subset of the test suite, rather than permanently eliminate test cases [7]. Our proposed approach attempts to improve the fault detection effectiveness for test case reduction even further. Compared with function coverage vectors, function call sequences consider not only sequential information about function executions, but also the relations between function calls. Additionally, in order to reduce the impact of the random sampling strategy on the fault detection effectiveness, we use a new sampling strategy, i.e. the max-min sampling strategy [19]. In contrast to the traditional sampling strategy, i.e. N-per-cluster sampling strategy [12], the new sampling strategy takes both the diversities of intra-cluster and the dissimilarities of inter-cluster into account.

To investigate the effectiveness of our proposed approaches, we have designed and conducted empirical studies using five subject projects space, flex, gzip, sed, and make. Experimental results show that sequence-based similarity approaches can detect more faults than vector-based similarity, random, and greedy function-coverage-based maximization test case reduction techniques in terms of the fault detection effectiveness. For the same type of structural profile, the max-min sampling strategy outperforms the traditional random sampling strategy. Due to reducing the randomness generated by the random sampling strategy, the max-min sampling strategy produces more reliable test suites. Over all, the contributions of this paper can be summarized as follows:

- **This study proposes cluster analysis of function call sequences to improve the fault detection effectiveness in regression testing.** In contrast to function coverage vectors, function call sequences consider additional important information to guide test case selection, including sequential information and the relations between functions calls.

- **We design and construct empirical studies in five programs so as to compare our proposed approaches with other reduction techniques, including random, greedy function-coverage-based maximization, and vector-based similarity approaches. The results illustrate that cluster analysis of function call sequence measured using the Euclidean distance is more effective than other reduction techniques.**

The rest of this paper is organized as follows: Section 2 summarizes related works. In Sec. 3, this study describes our approaches for test case reduction in more details. Experimental design and results analysis are presented in Secs. 4 and 5. The threats to validity are discussed in Sec. 6. Section 7 describes the conclusions and future work.
2. Related Work

In this section, we summarize related works including coverage-based and similarity-based test case selection and reduction techniques.

2.1. Similarity-based test case reduction

Hemmati et al. proposed similarity-based test case selection technique to achieve scalable model-based testing [18]. They applied genetic algorithm to select a subset of test cases generated from UML state machines, while preserving the fault revealing capability to the maximum extent. In that sense our approaches are very similar to theirs, i.e. both of them select diversified test cases to improve the effectiveness of testing. The most distinctive difference is that their method is applied to black-box testing, i.e. execution information of test cases is unavailable. Test case selection was performed before executing the test cases. In model-based testing, test cases were abstract rather than concrete. On the contrary, test cases we applied are concrete. Test case selection in our proposed approach is performed after executing the test cases. Similarly, Ledru et al. used string distances for test case prioritization [20]. All test cases were viewed as strings. Before executing test cases, they were ordered based on the similarity of strings. Unlike the above two methods, our approaches use dynamic profile information generated by executing test cases to implement test case reduction.

More recently, cluster analysis has also been proposed to implement test case prioritizing [13] and selection [16]. Masri et al. empirically compared coverage-based filtering to distribution-based filtering techniques with respect to the effectiveness of revealing defects [15]. In fact, distribution-based filtering techniques were similar with similarity-based selection techniques because both of techniques were implemented based on the similarity of program elements (e.g. statements, function calls, and basic blocks) test cases exercised. Chen et al. used semi-supervised clustering to improve regression test selection techniques [17]. All these methods represented execution profiles exercised by tests as numeric vectors [12, 13, 15] or binary vectors [16, 17], i.e. which functions or statements were executed or the number of times they were executed. Compared with vector-based methods, our approaches represent execution profiles as function call sequences by collecting sequential information and the relations between function calls. Similarly, Wang et al. [19] proposed cluster analysis of multiple structural profiles, but their methods were computationally expensive. Also, vector-based methods sampled test cases from each cluster by the random sampling strategy. However, our approach sampled test cases by the max-min distance strategy so as to select a diversified test suite to the maximum extent.

2.2. Coverage-based test case reduction

Compared with similarity-based test case reduction techniques, the underlying assumption of coverage-based test case reduction techniques is that the test suites which achieve more coverage (e.g. statements or functions) can detect more faults.
The reduced test suite can cover the equivalent elements as the original test suite according to some criteria [3, 4, 10]. Harrold et al. presented a heuristic algorithm to identify redundant and obsolete test cases in the original test suite [3]. Jerrfey et al. modified Harrold’s heuristic algorithm and presented a novel test suite reduction approach with selective redundancy [10]. The approach with selective redundancy decreased the fault detection loss at the expense of a small increase in the size of the reduced suite.

McMaster et al. also proposed a new coverage criterion, i.e. call-stack based coverage, to implement test case reduction [9]. A test case was represented as a set of maximum depth call stacks. Execution of a reduced test suite generated the same set of unique maximum depth call stacks as execution of its original test suite. Similarly, Smith et al. also proposed an approach of test case reduction and prioritization with call trees [21]. A call tree represented a program’s method invocations. The approximation algorithms were used to obtain a subset of the original tests. The reduced test suite can cover the same call tree paths as the original test suite.

Previous studies based on coverage maximization are not good enough to capture the functionality and deeper runtime characteristics [22]. Most of coverage-based reduction techniques aim at developing different approximate algorithms for test suite reduction, rather than precise algorithms. Compared with coverage-based reduction techniques, our approaches can generate a reduced test suite of any size according to the test budget and deadlines. The results of cluster analysis also provide a means for developers to quantitatively compare a program’s behavior. The comparison aids to debug faulty programs.

3. Methodology

Most of regression test case reduction techniques make use of code coverage information to select a high-performance subset from the original test case pool. We also collect execution profiles to make an improvement of the fault detection effectiveness in regression testing.

3.1. Overview

By executing test cases, we collect profile information, including the number of times functions were executed, sequential information, and the relations between function calls. Based on the similarity of profile information exercised by test cases, cluster analysis is applied to select a representative subset from the original test pool so as to eliminate redundant test cases. Figure 1a illustrates the use of cluster analysis of two different types of structural profiles to generate a reduced test suite.

First, we use dynamic instrumentation technique to collect execution profiles and construct two different types of structural profiles. And then, the distances between

---

*a In Fig. 1, P stands for the original programs. P' represents the modified programs, i.e. P' evolves from P.*
pair-wise tests are calculated by two different types of distance functions. \textit{K-means} clustering [23] is employed to partition test cases into clusters. Finally, test cases are extracted from each cluster to create a reduced test suite by different sampling strategies. The fault detection capability of the reduced test suite is evaluated to verify the effectiveness of different reduction techniques.

3.2. Execution profile collection

With the dynamic binary instrumentation tool PIN [24], we collect execution profiles and construct two different types of structural profiles \textit{FCV} and \textit{FCS}. In general, the programs under test consist of public library functions and internal functions (i.e. user-defined functions). Instrumenting all function calls can more accurately reflect the dynamic behaviors of tests. As a result, the lengths of function call sequences can be added greatly. In other words, this incurs increased costs for calculating sequences similarity. Instead of instrumenting all functions, we only instrument the user-defined functions.

3.2.1. Function coverage vector

The function coverage vector is defined as follows:

\textbf{Definition 1 (Function Coverage Vector, FCV).} All functions exercised by a test case can be represented as a vector called function coverage vector. Let $FN$ represent the set of functions in the program under test, $FN = \langle f_1, f_2, \ldots, f_n \rangle$, where $f_i$ stands for the name of the $i$th function and $n$ is equal to the number of functions in the program under test.

According to the element type, function coverage vectors are represented as two different types: binary vectors and numeric vectors.

![Fig. 1. Cluster analysis overview for test case reduction.](image-url)
• **Binary vector**
  The functions covered by each test case are represented as a binary vector ($BFCV$), like $V = \langle v_1, v_2, \ldots, v_n \rangle$. If $f_i$ is covered by a test, then $v_i$ corresponding to $f_i$ is equal to 1, otherwise 0. The binary vector model takes into account whether a function is covered or not.

• **Numeric vector**
  $FCV$ can also be implemented with numeric entries ($NFCV$), i.e. $v_i (v_i \geq 0)$ represents the number of times that function $f_i$ is executed. In contrast to the binary vector model, $NFCV$ makes full consideration of the frequency that a function is executed. Indeed, coverage information is very useful to quantitatively compare the similarity of execution profiles between pair-wise tests as coverage information reflects the dynamic trace of test cases. But, both of the two vector models ignore sequential information and the relations between function calls.

3.2.2. **Function call sequence**

The function call sequence is defined as follows:

**Definition 2 (Function Call Sequence, FCS).** Let $r$ represent the function call relation, $r = a \rightarrow b$, where $a$ and $b$ stand for the function names of the caller and the callee, respectively. $FCS$ is an ordered list of relation $r$, $FCS = \langle r_1, r_2, \ldots, r_m \rangle$, where $r_i$ is regarded as an element in $FCS$ and $m$ represents the number of function calls during a test execution. The relation $r$ is viewed as a whole. It has the nature of indivisibility.

The sequence contains not only the relations between calls, but also complete sequential information between function calls. For example, function call sequences of test case $t_1$ and $t_2$ are $cs_1$ ($\langle a \rightarrow b, b \rightarrow c, c \rightarrow c, b \rightarrow d, d \rightarrow e, a \rightarrow f \rangle$) and $cs_2$ ($\langle a \rightarrow b, b \rightarrow d, d \rightarrow f, b \rightarrow c, c \rightarrow c, a \rightarrow e \rangle$), respectively. Although $cs_1$ and $cs_2$ have different call relations and sequential information, function coverage vectors of $t_1$ and $t_2$ implemented with binary entries are ever identical, i.e. both of them are $\langle a, b, c, d, e, f \rangle$. Similarly, function coverage vectors implemented with numeric entries are also ever identical, i.e. both of them are $\langle 2, 3, 3, 2, 1, 1 \rangle$.

3.3. **Distance measures**

A distance measure takes execution profiles as inputs and outputs the positive real number as the distance between two test cases. We apply two distance measures, i.e. Euclidean distance and Levenshtein distance [25], qualitatively compare the diversities between test cases.

3.3.1. **Euclidean distance**

$N$-dimensional Euclidean distance is used to measure the distance between pair-wise function coverage vectors as it is the most commonly used and easily calculated. Let
X and Y stand for two profiles’ vectors, \(X : (x_1, x_2, \ldots, x_n)\) and \(Y : (y_1, y_2, \ldots, y_n)\). The distance between them is defined as follows:

\[
d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}
\] (1)

The distance between two binary vectors can be directly measured by the Euclidean distance. In order to calculate the distance between two numeric vectors, the number of times that each function was executed is normalized. The goal of normalization is to eliminate the variation of absolute frequency. Let \(t_i\) be the number of times that function \(f_i\) of profile \(p\) was executed. Let \(p_i\) denote the normalized value of \(t_i\) and \(p_i\) is defined as follows:

\[
p_i = \frac{t_i - \min_i}{\max_i - \min_i}
\] (2)

In Eq. (2), \(\min_i\) and \(\max_i\) represent the maximum and minimum number of times that function \(f_i\) was executed in all profiles, respectively. Having achieved normalization, the distance between profile \(p\) and \(q\) is calculated according to the Eq. (1). For numeric vectors, we still use Eq. (1), rather than the proportional-binary dissimilarity metric [15], to calculate the distance between them. The main reason is that the proportional-binary metric mixes the execution frequency and binary information.

Suppose that both of test case \(t_1\) and \(t_2\) cover the same functions, but the number of times that a certain covered function was executed is unequal. When the profiles exercised by \(t_1\) and \(t_2\) are represented as binary vectors, the distance between them is equal to 0. In other words, they are grouped into the same cluster. However, when the profiles are represented as numeric vectors, the distance between them may be unequal. The two test cases may be clustered into different groups.

This study also uses the Euclidean distance to calculate the distance between pair-wise sequences as previous study [19] has verified that the Levenshtein distance is expensive for the calculation of the distance between pair-wise test cases. When we calculate the distance between pair-wise sequences using Eq. (1), we construct a global vector, like \(GV = (cr_1, cr_2, \ldots, cr_n)\), which contains all call relations generated by all test cases at first. \(cr_i (1 \leq i \leq n)\) represents a call relation. Then, we transform each of function call sequences into a vector called function call vector, like \(FCV = (fcv_1, fcv_2, \ldots, fcv_n)\). If the call relation \(cr_i (1 \leq i \leq n)\) is generated by a certain test case, then \(fcv_i\) corresponding to \(cr_i\) is equal 1, otherwise 0. Function call vectors can also be implemented with numeric entries. In this case, each element in function call vectors represents the number of times that a call relation is called. Finally, we calculate the distance between pair-wise test cases using the Euclidean distance.

The distance of calculating pair-wise sequences is different from that of calculating pair-wise function coverage vectors because function call sequences contain the
call relations, whereas function call vectors only contain coverage information. In preliminary experiments, the number of times that call relations are executed can aid to improve the fault detection effectiveness. Therefore, we implement function call vectors with numeric entries.

### 3.3.2. Levenshtein distance

This study employs the Levenshtein Distance to measure the similarity between pairwise sequences. The Levenshtein distance between two sequences is defined as the minimum number of edit operations that are necessary to transform one sequence into the other [26]. The allowable edit operations are insertion, deletion, and substitution. The distance between two sequences is proportional to the number of edit operations. For instance, suppose that the program under test includes eight functions, i.e. \( a, b, c, d, e, f, h, \) and \( i \). By executing test case \( t_1 \) and \( t_2 \), we obtain the following function call sequences:

\[
\begin{align*}
\text{cs}_1 & : (a \to b, b \to c, b \to d, d \to e, e \to f, e \to h) \\
\text{cs}_2 & : (a \to b, b \to d, d \to e, e \to h, e \to f, a \to c)
\end{align*}
\]

When we calculate the distance between \( \text{cs}_1 \) and \( \text{cs}_2 \) using the Euclidean distance, the distance is equal to \( \sqrt{2} \). In order to calculate the Levenshtein distance between \( \text{cs}_1 \) and \( \text{cs}_2 \), firstly, the relation \( b \to c \) is deleted in \( \text{cs}_1 \). And then, the relation \( e \to h \) is inserted after the relation \( d \to e \). Finally, the relation \( e \to h \) in \( \text{cs}_1 \) is replaced with the relation \( a \to c \) in \( \text{cs}_2 \). Similarly, we can also convert the sequence \( \text{cs}_2 \) into the sequence \( \text{cs}_1 \) with the same edit costs because the distance between \( \text{cs}_1 \) and \( \text{cs}_2 \) is symmetric. The minimum number of edit operations between \( \text{cs}_1 \) and \( \text{cs}_2 \), i.e. the Levenshtein distance, is equal to 3.

The binary function coverage vectors of test case \( t_1 \) and \( t_2 \) are both \( v : (1,1,1,1,1,0,0) \) because they covered the same functions. The Euclidean distance between the two test cases is equal to 0, i.e. the two cases are considered to be identical although they have different function call relations. If the two profiles are represented as numeric vectors, the Euclidean distance between them is unequal in terms of the two test cases \( t_1 \) and \( t_2 \). The distance between them is very close. Consequently, they may still be clustered into the same group. In this example, we can see that \( FCS \) really captures more information about function calls than \( FCV \).

### 3.4. Clustering algorithm

\( K \)-means clustering algorithm is used in our experiments. The main reason is that \( k \)-means clustering has been verified to be efficient and simple for generating high quality clusters. It performed reasonably well in our preliminary experiments. \( K \)-means clustering can be implemented by assigning the value of parameter \( k \) in advance. The value of \( k \) has a great influence on the quality of clustering. In previous studies [15, 16, 18], the researchers did not explicitly elaborate the rule of identifying
the value of $k$. In view of this situation, this study discusses how to pick the optimal value of parameter $k$ in Sec. 4.

$K$-means clustering uses the centroid of a cluster to represent that cluster. The mean value of all objects in the same cluster is regarded as the centroid. Nevertheless, the mean value of all objects is meaningless because all objects are in non-Euclidean space. An alternative is to pick one object in each cluster to represent the cluster itself, i.e. the object has the minimum average distance to the other objects in the cluster. The representative object is called the clustroid [27]. Its pseudo-code is described in Algorithm 1. $k$ selected test cases the clustroids of $k$ clusters; The time complexity of $k$-means algorithm is $O(nki)$, where $n$, $k$, and $i$ represent the number of objects, the number of clusters, and the number of iterations, respectively. In most cases, $k$ and $i$ are far less than $n$. Therefore, its time complexity is $O(n)$.

**Algorithm 1 K-means clustering**

**Input:** The number of clusters, $k$; A set of $n$ test cases, $S$

**Output:** $k$ clusters $(C_1, C_2, \ldots, C_k)$

1: $CS \leftarrow \{\}$
2: Pick the first test case $t$ from $S$ randomly;
3: $CS \leftarrow CS \cup \{t\}$;
4: $S \leftarrow S - \{t\}$;
5: **while** $|CS| < k$ **do**
6: Pick the test case $t$ whose minimum distance from $CS$ is as large as possible;
7: $CS \leftarrow CS \cup \{t\}$;
8: $S \leftarrow S - \{t\}$;
9: **end while**
10: Make $k$ selected test cases the clustroids of $k$ clusters;
11: **for** each test case $t \in S$ **do**
12: Find the clustroid $cd$ ($cd \in C_i$) to which $t$ is closet;
13: $C_i \leftarrow C_i \cup \{t\}$;
14: $S \leftarrow S - \{t\}$;
15: Adjust the clustroid of $C_i$;
16: **end for**
17: **return** $k$ clusters;

3.5. **Sampling strategy**

Having created clusters, test cases are selected from each cluster by two different types of sampling strategies: random strategy and max-min distance strategy.

- **Random strategy**

  Random strategy means to sample test cases from each cluster at random. If we plan to sample $n$ test cases from each cluster by random strategy, when the number of test cases in a certain cluster is less than or equal to $n$, then all these test cases will
be selected. When \( n = 1 \), the strategy is called one-per-cluster sampling, i.e. one test case is extracted from each cluster. Dickinson et al. proposed adaptive sampling [12], i.e. if the first test case selected in each cluster is failed, then all remaining test cases in the cluster are also selected. By analyzing clustering results, we find that a small part of clusters comprise both passed tests and failed tests because they have similar execution profiles. If we use the adaptive sampling, the size of reduced test suite will be greatly increased. However, the fault detection effectiveness corresponding to the reduced test suite could not be significantly improved.

- Max-min distance strategy

The randomness generated by random strategy influences the reliability of the fault detection effectiveness. Therefore, this study applies the max-min distance strategy to reduce the impact of the randomness on the fault detection effectiveness. In order to describe the strategy, we first define the distance between a test case \( t \) and a set of test cases \( TCS \) as follows:

\[
d(t, TCS) = \min \{d(t, t_i) \mid t \notin TCS \text{ and } t_i \in TCS\}.
\]

In Eq. (3), the distance is called the minimal set distance and both of \( t \) and \( TCS \) belong to the same cluster. As a representative object, the clustroid is selected in each cluster at first. And then, the test case that has the maximum distance to the clustroid will be selected. Within the same cluster, the test case that has the maximum minimal set distance to test cases already selected will be preferentially selected from the unselected test cases.

In contrast to the random strategy, the max-min strategy takes into account the diversities of selected test cases as far as possible, and reduces the impact of random selection on the diversity of selected test cases. Cluster analysis takes full consideration of the dissimilarities of inter-cluster. Similarly, the max-min strategy also considers the diversities of intra-cluster.

3.6. Evaluation

In order to evaluate the effectiveness of our proposed approaches, two alternative measures are introduced: one is the fault detection effectiveness, and the other is the reduction percentage in the size of reduced test suite [11]. Let \( T \) represent the original test suite and \( T' \) denote the reduced test suite. Let \( |F| \) and \( |F'| \) denote the number of faults detected by \( T \) and \( T' \) over the faulty versions, respectively. For each reduced test suite, we calculate the percentage in the fault detection effectiveness (PFDE):

\[
PFDE = \frac{|F'|}{|F|} \times 100\%.
\]

Let \( |S_1| \) and \( |S_2| \) represent the sizes of two reduced test suites generated by two different test case reduction techniques with the same PFDE. To compare two different reduction techniques, the reduction percentage (RP) in the size of reduced test suite is also calculated.
4. Experiment

In this section, we conducted some experiments to evaluate the effectiveness of our proposed approaches by comparing cluster analysis of two different types of structure profiles for test case reduction with random and greedy function-coverage-based maximization test case reduction techniques. In order to objectively and fairly evaluate the effectiveness of different reduction techniques, all algorithms are performed on a Ubuntu 11.10 system running on a Intel(R) Core(TM) i3 CPU 3.10 GHz with 4 GB memory.

4.1. Subject program

Our experiments were conducted on the programs space, flex, gzip, sed, and make\(^b\), which were widely used to validate the effectiveness of different reduction techniques. All the five subject programs are written in C language. Descriptive information about the selected programs is presented in Table 1. The last two columns show execution time for the whole test suite and average execution time for each test case on different subject programs.

Space is designed by the European Space Agency (ESA) to interpret an array definition language (ADL). The space program consists of a single base version and 38 modified versions. These modified versions include real faults, rather than seeded faults. Some versions (e.g. \(v_1, v_2, v_{32}\)) are eliminated because their faults are not detected by any tests. Also, some versions that have too many failed tests are also eliminated. For example, fault-revealing tests account for 93.13% of the total tests in version \(v_6\). Even though we select a part of test cases from the test suite at random, the fault in \(v_6\) can be easily found. Based on the above analysis, we choose 30 versions from 38 modified versions in total.

<table>
<thead>
<tr>
<th>Program name</th>
<th>Number of versions</th>
<th>Line of code</th>
<th>Number of functions</th>
<th>Test suite size</th>
<th>Execution time (ms)</th>
<th>Average time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>38</td>
<td>6199</td>
<td>136</td>
<td>13585</td>
<td>13780</td>
<td>1.01</td>
</tr>
<tr>
<td>Flex</td>
<td>5</td>
<td>10459</td>
<td>148</td>
<td>567</td>
<td>2330</td>
<td>4.11</td>
</tr>
<tr>
<td>Gzip</td>
<td>5</td>
<td>5680</td>
<td>104</td>
<td>217</td>
<td>4410</td>
<td>20.61</td>
</tr>
<tr>
<td>Sed</td>
<td>7</td>
<td>14427</td>
<td>255</td>
<td>370</td>
<td>430</td>
<td>1.62</td>
</tr>
<tr>
<td>Make</td>
<td>5</td>
<td>33545</td>
<td>268</td>
<td>1043</td>
<td>72800</td>
<td>69.80</td>
</tr>
</tbody>
</table>

\(^b\)http://sir.unl.edu/php/showfiles.php
**Flex** is the fast lexical analyzer which recognizes lexical patterns in text. The program consists of a base version and 5 modified versions. The 5 modified versions evolved from the base version. The faulted versions corresponding to the 5 modified versions can be built by manually seeding faults. Some faults can be detected by none of test cases, e.g., the fault $f_2$, $f_{12}$, and $f_{13}$ in version $v_1$ so these faults are eliminated. Likewise, some faults can be detected by more than 50% test cases are also eliminated, such as the fault $f_2$ and $f_3$ in version $v_5$, etc. Consequently, we only select 32 faults from 81 faults contained by 5 modified versions in total.

**Gzip**, **sed**, and **make** are all unix utilities. **Gzip** is used to compress and decompress files. It consists of a base version and 5 modified versions. Some faults, for example, all faults contained by version $v_3$, can be detected by none of test cases, so these faults are eliminated. Likewise, some faults detected by more than 50% test cases are also eliminated, such as $f_1$ and $f_6$ in version $v_2$. Only 13 faults are selected from 59 faults for the program **gzip** in total. **Sed** is a stream-oriented non-interactive text editor. Those faults detected by more than 50% test cases or none of test cases are eliminated. Only 23 faults are selected from 32 faults. **Make** is used to compile and link programs for generating executables and other non-source programs. For the program **make**, we only eliminate the fault $f_9$ in version $v_1$ because more than 50% test cases can detect the fault. In a word, we select 34 faults from 35 faults.

### 4.2. Experimental setup

#### 4.2.1. Research questions

To evaluate the effectiveness of our approaches, we examined the following three research questions:

**RQ1:** Is cluster analysis of function sequence better than cluster analysis of function coverage vector implemented with binary entries or numeric entries in terms of the fault detection effectiveness?

Previous empirical studies have shown that vector-based methods can improve the effectiveness of regression testing. The goal of RQ1 is to check whether sequential information and the relations between function calls can aid to improve the fault detection effectiveness even further.

**RQ2:** Can test suites generated by the approaches we proposed preserve more fault-detecting capability than greedy function-coverage-based maximization algorithm and random-based test case reduction techniques with the same size of reduced test suites?

[^http://directory.fsf.org/wiki/GNU]
Similar to the study [9], we also use the greedy function-coverage-based (FCB) reduction method as one of comparison baselines. The greedy coverage-based reduction methods select a subset of test cases from the perspective of function or statement coverage. However, similarity-based methods generate a subset from the perspective of diversity of test cases. The goal of RQ2 is to check the effectiveness of three different types of reduction techniques.

**RQ3:** *Is the max-min distance strategy better than random sampling strategy in the context of cluster analysis of function call sequence measured using the Euclidean distance?*

Random sampling strategy has been verified to be an efficient sampling strategy. The randomness generated by random sampling strategy influences the effectiveness of the similarity-based reduction techniques. In this case, this study uses the max-min sampling strategy to cluster analysis of function call sequence measured using the Euclidean distance.

**RQ4:** *What is the best technique for cluster-based reduction techniques in terms of the cost-effectiveness?*

If a reduction technique improves the fault detection effectiveness with great time cost, it may be infeasible in practice. For this purpose, we analyze time cost of cluster-based reduction techniques to guide practical use.

To answer RQ1, we empirically compared our proposed approaches with the vector-based (i.e. BFCV and NFCV) test case reduction and random test case reduction techniques in terms of the fault detection effectiveness. To answer RQ2, we compared the sequence-based similarity test case reduction with greedy function-coverage-based maximization test case reduction in terms of the fault detection effectiveness. To answer RQ3, we experimentally compared the max-min strategy with the random strategy by sampling the same number of test cases for four different types of test case reduction techniques. To answer RQ4, we analyzed time cost including real time and time complexity to guide practical use.

### 4.2.2. Experiment step

Our proposed approaches require three main steps to obtain a reduced test suite. The experimental steps are summarized as follows:

- **Instrumentation**
  
  With the dynamic instrumentation tool PIN, we instrument the user-defined functions and construct two different types of structural profiles.

- **Clustering analysis**

  For function call sequences, two distance measures are applied for calculating the distance between pair-wise test cases. And then, similarity matrixes are constructed. Finally, $k$-means algorithm is implemented for clustering regression test cases.
• **Sampling test cases**

By one of sampling strategies as described in Sec. 3, a part of test cases is extracted from each cluster and merged into a reduced test suite. The fault detection effectiveness of the reduced test suite is calculated by Eq. (4).

### 4.3. Identification of the value of $k$

The identification procedure, i.e. finding the value of the parameter $k$, is based on the similarity characteristics of test cases. If the value of $k$ is too large or too small, the $k$-means algorithm produces low-quality clustering results. In order to produce high quality clustering results, the appropriate value of $k$ should be carefully picked. In previous study [17], the value of $k$ is usually set to 0.5%–2% of the number of test cases in the original test case pool. An alternative method [12] is that the value of $k$ can be assigned according to the number of sampled test cases, i.e. the value of $k$ is equal to the number of the selected test cases. Suppose that we create a reduced test suite including 20 test cases, the value of $k$ is equal to 20.

There are drawbacks with the above two methods, i.e. both of methods for setting the value of $k$ only take the number of test cases into account, rather than the distribution characteristics of test cases. In order to estimate the optimal value of $k$, we employ the gap statistic algorithm [28] to generate high-quality clustering results. This estimate algorithm considers the distribution of test cases, so it is very general and applicable to any distance measure. Table 2 shows the value of parameter $k$ for each subject program.

### 4.4. Greedy function-coverage-based algorithm

Coverage-based test case reduction technique has been verified to be an effective method for improving the effectiveness of regression testing. Therefore, we choose greedy function-coverage-based maximization algorithm as one of our comparison baselines. The greedy function-coverage-based maximization algorithm works as follows: A test case that yields the greatest function coverage is selected at first. And then, the next test case to be selected is the one that covers the maximum functions not yet covered. The process is repeated until all functions have been covered by at least one test case or the number of test cases selected is equal to the number of

<table>
<thead>
<tr>
<th>Program name</th>
<th>Value of $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sed</td>
<td>10</td>
</tr>
<tr>
<td>Flex</td>
<td>12</td>
</tr>
<tr>
<td>Gzip</td>
<td>8</td>
</tr>
<tr>
<td>Make</td>
<td>25</td>
</tr>
<tr>
<td>Space</td>
<td>65</td>
</tr>
</tbody>
</table>

Table 2. The value of $k$ for each subject program.
assigned tests. When multiple test cases cover the same number of functions not yet covered, we select the test case that yields the greatest function coverage from these test cases, rather than doing this randomly. Given a test suite containing $m$ test cases and a program containing $n$ functions, selecting a test has cost $O(mn)$. If we generate a reduced test suite containing $r$ test cases, the cost of greedy function-coverage-based algorithm is $O(mnr)$.

5. Experimental Results and Analysis

We attempt to sample different proportions of test cases to observe the effects of different sampling strategies and structural profiles on the fault detection effectiveness. In order to eliminate the randomness of the random sampling strategy, 1000 test suites were generated for each sampling proportion with different random seeds. For each reduced test suite, we recorded how many faults were detected and how many tests were selected. The mean of faults detected is calculated for each sampling proportion.

Figures 2 and 3 show our experimental results. Let $RB$ represent the random-based test case reduction technique. Let $FCSE$ and $FCSL$ represent sequence-based similarity reduction techniques measured using the Euclidean distance and the Levenshtein distance, respectively. The x-axis of each graph indicates the size of tests selected; While the y-axis of each graph indicates the mean percentage in fault detection effectiveness.

5.1. Statistical analysis

Since the random sampling strategy has the characteristic of randomness, statistical tests are used to assess the effectiveness of different reduction techniques. By checking the distribution of the fault detection effectiveness for each reduction technique on

![Fig. 2. Faults detection effectiveness of different test case reduction techniques for space, flex, and gzip.](image)
each subject program, we find that the data follow normal distribution when sampling proportion is small. However, the data follow skewed distribution with the increase of sampling proportion. Particularly, the fault detection effectiveness generated by greed function-coverage-based maximization algorithm is equal 100% on the program `make` no matter how large the sampling proportion is. In such a case nonparametric statistical tests may be more suitable for checking the difference between our approach and other reduction techniques. Therefore, the nonparametric "Mann-Whitney U-test" [29] and the "Vargha-Delaney A-test" [30] are used to qualitatively and quantitatively compare our proposed approaches and other reduction techniques.

For the nonparametric "Mann-Whitney U-test", the null hypothesis is defined to state that there is no difference between the results generated by our approaches and other reduction techniques. When we reject the null hypothesis at a significance level \( \alpha = 0.05 \), we say that the improvement of our approaches over other techniques is

![Fig. 2. (Continued) (c) Random Sampling Strategy (flex)](image)

![Fig. 2. (Continued) (d) Max-Min Sampling Strategy (flex)](image)

![Fig. 2. (Continued) (e) Random Sampling Strategy (gzip)](image)

![Fig. 2. (Continued) (f) Max-Min Sampling Strategy (gzip)](image)
statistically significant. To assess the magnitude of the improvement, this study uses the Vargha-Delaney A-test, which is a nonparametric effect size measure and is also advocated in [31, 32]. In our context, the effect size is interpreted to be a difference in the fault detection effectiveness. In [30] Vargha and Delaney suggest an A-statistic of greater than 0.64 (or less than 0.36) is indicative of a “medium” effect size and greater than 0.71 (or less than 0.29), of a “large” effect size [31].

By observing Figs. 2 and 3, we find that FCSL can detect more faults than FCSE in terms of the fault detection effectiveness. If FCSE is statistically superior to other reduction techniques, then FCSL is more statistically superior. Therefore, Table 3 only shows significance information on the comparisons of FCSE and the other test case reduction techniques. In Table 3, N represents the number of test cases selected from each cluster. For each subject program, Table 3 only shows a part of statistical

<table>
<thead>
<tr>
<th>Tests Selected</th>
<th>% Faults Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>20</td>
<td>50%</td>
</tr>
<tr>
<td>40</td>
<td>60%</td>
</tr>
<tr>
<td>60</td>
<td>70%</td>
</tr>
<tr>
<td>80</td>
<td>80%</td>
</tr>
<tr>
<td>100</td>
<td>90%</td>
</tr>
<tr>
<td>120</td>
<td>95%</td>
</tr>
<tr>
<td>140</td>
<td>100%</td>
</tr>
</tbody>
</table>

By observing Figs. 2 and 3, we find that FCSL can detect more faults than FCSE in terms of the fault detection effectiveness. If FCSE is statistically superior to other reduction techniques, then FCSL is more statistically superior. Therefore, Table 3 only shows significance information on the comparisons of FCSE and the other test case reduction techniques. In Table 3, N represents the number of test cases selected from each cluster. For each subject program, Table 3 only shows a part of statistical
Table 3. The statistical comparisons between FCSE and the other test case reduction techniques. RB and FCB represent random-based and greedy function-coverage-based test case reduction techniques, respectively. BFCV and NFCV represent cluster analysis of function coverage vector implemented with binary and numeric entries, respectively. The “MWU” column shows the difference of the Mann-Whitney-U. (1 represents statistically significant difference). If the difference is statistically significant, the magnitude of “A-test” is shown.

<table>
<thead>
<tr>
<th>TCRT</th>
<th>MWU</th>
<th>A-test</th>
<th>p-value</th>
<th>MWU</th>
<th>A-test</th>
<th>p-value</th>
<th>MWU</th>
<th>A-test</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Space N = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td>1</td>
<td>0.69</td>
<td>0.0000</td>
<td>1</td>
<td>0.63</td>
<td>0.0000</td>
<td>1</td>
<td>0.70</td>
<td>0.0000</td>
</tr>
<tr>
<td>FCB</td>
<td>1</td>
<td>0.63</td>
<td>0.0000</td>
<td>1</td>
<td>0.61</td>
<td>0.0011</td>
<td>1</td>
<td>0.62</td>
<td>0.0000</td>
</tr>
<tr>
<td>BFCV</td>
<td>1</td>
<td>0.67</td>
<td>0.0000</td>
<td>1</td>
<td>0.62</td>
<td>0.0000</td>
<td>1</td>
<td>0.68</td>
<td>0.0000</td>
</tr>
<tr>
<td>NFCV</td>
<td>1</td>
<td>0.64</td>
<td>0.0000</td>
<td>1</td>
<td>0.63</td>
<td>0.0000</td>
<td>1</td>
<td>0.65</td>
<td>0.0000</td>
</tr>
<tr>
<td>Space N = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td>1</td>
<td>0.73</td>
<td>0.0000</td>
<td>1</td>
<td>0.76</td>
<td>0.0000</td>
<td>1</td>
<td>0.74</td>
<td>0.0000</td>
</tr>
<tr>
<td>FCB</td>
<td>1</td>
<td>0.66</td>
<td>0.0000</td>
<td>1</td>
<td>0.58</td>
<td>0.0021</td>
<td>1</td>
<td>0.59</td>
<td>0.0016</td>
</tr>
<tr>
<td>BFCV</td>
<td>1</td>
<td>0.67</td>
<td>0.0000</td>
<td>1</td>
<td>0.57</td>
<td>0.0035</td>
<td>1</td>
<td>0.56</td>
<td>0.0067</td>
</tr>
<tr>
<td>NFCV</td>
<td>1</td>
<td>0.68</td>
<td>0.0000</td>
<td>1</td>
<td>0.60</td>
<td>0.0000</td>
<td>1</td>
<td>0.56</td>
<td>0.0127</td>
</tr>
<tr>
<td>Space N = 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td>1</td>
<td>0.65</td>
<td>0.0000</td>
<td>1</td>
<td>0.92</td>
<td>0.0000</td>
<td>1</td>
<td>0.85</td>
<td>0.0000</td>
</tr>
<tr>
<td>FCB</td>
<td>1</td>
<td>0.58</td>
<td>0.0037</td>
<td>1</td>
<td>0.60</td>
<td>0.0000</td>
<td>1</td>
<td>0.62</td>
<td>0.0000</td>
</tr>
<tr>
<td>BFCV</td>
<td>1</td>
<td>0.58</td>
<td>0.0039</td>
<td>1</td>
<td>0.76</td>
<td>0.0000</td>
<td>1</td>
<td>0.73</td>
<td>0.0000</td>
</tr>
<tr>
<td>NFCV</td>
<td>1</td>
<td>0.58</td>
<td>0.0022</td>
<td>1</td>
<td>0.75</td>
<td>0.0000</td>
<td>1</td>
<td>0.70</td>
<td>0.0000</td>
</tr>
<tr>
<td>Space N = 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td>1</td>
<td>0.96</td>
<td>0.0000</td>
<td>1</td>
<td>0.95</td>
<td>0.0000</td>
<td>1</td>
<td>0.95</td>
<td>0.0000</td>
</tr>
<tr>
<td>FCB</td>
<td>1</td>
<td>0.65</td>
<td>0.0000</td>
<td>1</td>
<td>0.62</td>
<td>0.0136</td>
<td>1</td>
<td>0.53</td>
<td>0.0283</td>
</tr>
<tr>
<td>BFCV</td>
<td>1</td>
<td>0.59</td>
<td>0.0010</td>
<td>1</td>
<td>0.59</td>
<td>0.0036</td>
<td>1</td>
<td>0.57</td>
<td>0.0023</td>
</tr>
<tr>
<td>NFCV</td>
<td>1</td>
<td>0.60</td>
<td>0.0020</td>
<td>1</td>
<td>0.58</td>
<td>0.0000</td>
<td>1</td>
<td>0.53</td>
<td>0.0258</td>
</tr>
<tr>
<td>Space N = 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td>1</td>
<td>0.96</td>
<td>0.0000</td>
<td>1</td>
<td>0.93</td>
<td>0.0000</td>
<td>1</td>
<td>0.92</td>
<td>0.0000</td>
</tr>
<tr>
<td>FCB</td>
<td>1</td>
<td>0.65</td>
<td>0.0000</td>
<td>1</td>
<td>0.61</td>
<td>0.0138</td>
<td>1</td>
<td>0.61</td>
<td>0.0025</td>
</tr>
<tr>
<td>BFCV</td>
<td>1</td>
<td>0.59</td>
<td>0.0010</td>
<td>1</td>
<td>0.59</td>
<td>0.0036</td>
<td>1</td>
<td>0.55</td>
<td>0.0193</td>
</tr>
<tr>
<td>NFCV</td>
<td>1</td>
<td>0.60</td>
<td>0.0000</td>
<td>1</td>
<td>0.62</td>
<td>0.0009</td>
<td>1</td>
<td>0.55</td>
<td>0.0163</td>
</tr>
<tr>
<td>Space N = 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td>1</td>
<td>0.96</td>
<td>0.0000</td>
<td>1</td>
<td>0.93</td>
<td>0.0000</td>
<td>1</td>
<td>0.92</td>
<td>0.0000</td>
</tr>
<tr>
<td>FCB</td>
<td>1</td>
<td>0.95</td>
<td>0.0000</td>
<td>1</td>
<td>0.73</td>
<td>0.0063</td>
<td>1</td>
<td>0.61</td>
<td>0.0113</td>
</tr>
<tr>
<td>BFCV</td>
<td>1</td>
<td>0.55</td>
<td>0.0021</td>
<td>1</td>
<td>0.54</td>
<td>0.0127</td>
<td>1</td>
<td>0.53</td>
<td>0.0136</td>
</tr>
<tr>
<td>NFCV</td>
<td>1</td>
<td>0.55</td>
<td>0.0034</td>
<td>1</td>
<td>0.54</td>
<td>0.0124</td>
<td>1</td>
<td>0.53</td>
<td>0.0127</td>
</tr>
</tbody>
</table>
results because there is no statistical difference between our proposed approach and other alternative reduction techniques with large sampling proportions.

5.2. **RQ1: Our approaches versus vector-based methods**

In the case of small sampling size, Figs. 2 and 3 clearly show that cluster-based and greedy function-coverage-based maximization test case reduction techniques retain the better fault detection capability than random reduction with the same size of test suites except the program *sed*. For two different types of structural profiles, the sequence-based approaches outperform binary vector-based and numeric vector-based approaches with the same sampling proportions in terms of the fault detection effectiveness. The experimental results illustrate that the use of cluster analysis of execution profiles containing dynamic execution information (i.e. sequential information and the relations between function calls) can actually improve the effectiveness of test case reduction. As shown in Table 3, *FCSE* has more often “medium” effect size over *BFCV* and *NFCV*.

Although *FCSE* is measured using the Euclidean distance, it detects more faults than vector-based approaches. This suggests that the call relations between function calls can more accurately distinguish test cases. Comparing to *FCSE*, *FCSL* makes more significant improvement than vector-based approaches. This shows that the combination of sequential information with the call relations between function calls can aid to further improve the fault detection effectiveness.

Take the example of *space* in Fig. 2(a), the improvement is statistically significant using the *Mann-Whitney U-test*. In order to detect more than 90% of faults in the *space*, our approach needs 300 test cases. Both of *BFCV* and *NFCV* need more than 400 test cases. The *RP* of our approach improves more than 33.3%. The numeric vector-based approach performs slightly better than the binary vector-based approach. The most plausible explanation for the improvement is that the numeric vector-based approach considers the number of times that the functions covered by
test cases are executed. Furthermore, we find that when test cases have minor function coverage, similarity-based approaches do not make more improvements than greedy function-coverage-based maximization algorithm.

5.3. RQ2: Our approach versus greedy function-coverage-based maximization and random reduction techniques

Our proposed approaches can detect more faults than the greedy function-coverage-based maximization in all programs except the make. The improvement is statistically significant. The most likely explanation is that software faults are usually isolated in small clusters. In addition, some test cases are designed for exercising production features or exceptional behaviors, rather than for code coverage. Using A-test, our approach has “medium” and “large” effect sizes over greedy function-coverage-based maximization and random reduction technique, respectively. For the make, when the sampling proportion is close to 2.3%, greedy function-coverage-based maximization algorithm can detect all faults. By observing the reduced test suites, we find that test cases $t_{850}$ and $t_{852}$ can detect all the selected faults. Test cases $t_{850}$ and $t_{852}$ have greater function coverage. In this case, greedy function-coverage-based maximization algorithm outperforms other alternative reduction techniques.

Note that the greedy function-coverage-based maximization algorithm may fall into a dilemma, i.e. having selected $n$ maximal coverage tests, there are some functions not covered by previously selected test cases. In some cases, those tests cannot cover more functions because the occurrences of faults prevent them from covering more functions. In other words, those test cases which coverage more functions cannot always effectively detect faults. Take the example of sed, from $N = 5$ to $N = 10$, 72 test cases (i.e. accounting for 19.5% of the whole test suite) only achieve 3% of improvement in the fault detection effectiveness.

Additionally, by observing Fig. 3(a) we find that random reduction technique performs better than our expectation with respect to the fault detection effectiveness. The most likely explanation is that the rate of fault detection is very high, and most of test cases have minor function coverage. In these experiments, most of test cases can detect more than one fault. Particularly, for the flex and sed most of test cases can detect more than 4 faults. Take the flex case in Fig. 2(c), when the sampling proportion is more than 20%, all faults can be detected by random test case reduction. By observing Fig. 3(a), we also find that random test case reduction outperforms other reduction techniques when sampling proportion is less than 18.9%. In case of high fault rate and large sampling size, there is no statistically significant difference between random reduction technique and other alternative reduction techniques.

5.4. RQ3: The max-min sampling strategy versus the random sampling strategy

By comparing Fig. 2 with Fig. 3, we find that the max-min distance strategy outperforms the random sampling strategy with the same sampling proportion in the
context of cluster analysis of function call sequence measured using the Euclidean
distance. The new strategy distinguishes the differences of test cases within a cluster
and reduces the effect of the randomness generated by the random sampling strategy
on the fault detection effectiveness. The strategy provided a diversified test suite at
large. This confirmed further that the diversity of test cases can aid to detect much
more faults.

By observing from the Figs. 2 and 3, we find the fault detection effectiveness
increases along with the sizes of reduced test suites. The finding is consistent with
[4, 9, 11], i.e. the fault detection effectiveness generally decreases as the percentage
reduction in test suite size increases. The improvement of the fault detection
effectiveness slows down gradually. The main reason may be that some faults can
be detected by few test cases. Furthermore, we find an interesting phenomenon,
i.e. test cases that reveal the different faults are partitioned into the same cluster
as they have the similar execution profiles. For future work, we will refine the
clustering results by combining cluster analysis with information flow analysis
even further.

By these experiments, we find there are no clear differences between random test
suite reduction technique and other alternative reduction techniques for large sample
size in the fault detection effectiveness. For example, if we randomly sample 40% test
cases from the original test pool in the program space, the fault detection effect-
iveness of the reduced test suite is almost 100%. In practice, test case reduction is
mostly used for selecting a small sample of test cases. Consequently, we only select a
small sample of test cases for comparing the effectiveness of different reduction
techniques.

5.5. RQ4: Cost analysis

In order to more objectively justify the effectiveness of different reduction techniques,
this study compared the costs in terms of computational complexity and the actual
time required for the similarity calculation between pair-wise test cases. The total
time includes the time required for the selection and the calculation of the distance
between pair-wise test cases. Compared with the time required for the calculation,
the time required for the selection is negligible. Moreover, the time required to select
a reduced test suite is the same for cluster analysis of function coverage vectors and
our approaches. However, the time required to calculate the distance between pair-
wise test cases is different. Therefore, we only compare the time required to calculate
the distance between pair-wise test cases. Table 4 presents the time cost (in seconds)
and average length information for cluster-based reduction techniques on different
subject programs. The second and fifth columns show the average length of
sequences and vectors, respectively. The third and fourth columns show the time cost
of constructing similarity matrices by sequence-based approaches (e.g. FCSL and
FCSE). The last two columns give the time cost for constructing similarity matrices
by vector-based approaches (e.g. BFCV and NFCV).
In most cases, the lengths of function call sequences are relatively short. Once the program under test is a recursive program, the average lengths of function call sequences can be greatly increased. For example, when test cases \texttt{t7} and \texttt{t8} are run in the program \texttt{gzip}, the number of times that the functions \texttt{send_bits()} and \texttt{logest_match()} were executed is 16572 and 9031, respectively. The function call sequences generated by executing test cases \texttt{t7} and \texttt{t8} greatly increase the average lengths of all sequences. If test cases \texttt{t7} and \texttt{t8} are not executed, the average lengths of sequences are only 89.9. Note that time cost is also related to the size of test suite.

The Euclidean distance can be implemented with time complexity $O(\max(v_1, v_2))$ (i.e. $O(n)$) and space complexity $O(\max(v_1, v_2))$ (i.e. $O(n)$), where $v_1$ and $v_2$ are the lengths of two function coverage vectors. The Levenshtein edit distance can be calculated by the dynamic programming algorithm with time complexity $O(s_1 s_2)$ (i.e. $O(n^2)$) and space complexity $O(\max(s_1, s_2))$ (i.e. $O(n)$), where $s_1$ and $s_2$ are the lengths of two function call sequences being compared. By comparing the time complexity, we find that the Levenshtein distance is highly computationally expensive.

By observing Table 4, we find that cluster analysis of function call sequences using the Levenshtein distances consumes the most time. Cluster analysis of function call sequences using the Euclidean distance consumes almost the same time cost as cluster analysis of function coverage vectors, i.e. the costs of the two approaches are comparable. But, in terms of the fault detection effectiveness, cluster analysis of function call sequences using the Euclidean distance can detect more faults than cluster analysis of function coverage vectors. The approach, i.e. cluster analysis of function call sequences using the Euclidean distance, is viewed as the most cost-effective. Although cluster analysis of function call sequences using the Levenshtein distance has the most time cost, it is still very valuable. Especially, testing for embedded software, in some scenarios, usually depends on hardware or requires access to the dedicated test infrastructures where each of test cases can be very expensive to run [33]. In the context, our proposed approaches are more valuable.

### Table 4. Time cost analysis about sequence-based and vector-based test case reduction techniques for each subject programs.

<table>
<thead>
<tr>
<th>Program name</th>
<th>Average length of sequence</th>
<th>Time cost of sequence Levenshtein</th>
<th>Time cost of sequence Euclidean</th>
<th>Length of vector</th>
<th>Time cost of binary vector</th>
<th>Time cost of numeric vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sed</td>
<td>76.8</td>
<td>6.8</td>
<td>1.3</td>
<td>255</td>
<td>0.5</td>
<td>1.1</td>
</tr>
<tr>
<td>Flex</td>
<td>529.9</td>
<td>576.7</td>
<td>4.3</td>
<td>148</td>
<td>1.9</td>
<td>3.7</td>
</tr>
<tr>
<td>Gzip</td>
<td>416.2</td>
<td>81.5</td>
<td>0.8</td>
<td>104</td>
<td>0.4</td>
<td>0.7</td>
</tr>
<tr>
<td>Make</td>
<td>678.8</td>
<td>1511.5</td>
<td>10.7</td>
<td>268</td>
<td>4.5</td>
<td>9.1</td>
</tr>
<tr>
<td>Space</td>
<td>484.3</td>
<td>1242.9</td>
<td>29.9</td>
<td>136</td>
<td>13.8</td>
<td>27.2</td>
</tr>
</tbody>
</table>

5.6. **Discussion**

The main motivation for similarity-based techniques is to provide a scalable testing approach [18]. The input of our approaches is a set of function call sequences which
represent test cases. The more test cases in the original test suite are, the larger the scale of the inputs. Additionally, the input is also related to the length of function call sequences. The longer the lengths of function call sequences are, the larger the average cost of calculating the distance between pair-wise test cases is. The structure of the program under test has important impact on the scalability. If the program under test is a recursive program, the average length of function call sequences also is greatly increased. When the program under test is a recursive program, determining how to reduce the length of function call sequences becomes a key issue. In future work, we also will consider seeking for a solution for this issue. In short, the scalability of our proposed approaches increases with the size of test suite and the length of function call sequences.

6. Threats to Validity

Although the experiments were carefully designed and implemented, this study is not free from threats to its validity. These potential threats are summarized as follows:

- **Internal validity**
  The conclusions of this study could be affected by the value of parameter $k$. If the value of $k$ is too large or too small, $k$-means clustering will produce low-quality clustering results. In order to reduce the effect of the value of $k$ on the clustering results, we use the gap statistics algorithm to pick the optimal value of $k$. In contrast to previous studies, the optimal value of $k$ is picked based on the distribution characteristics of test cases, rather than the number of total test cases or the number of sampled test cases. Additionally, all profiles are collected by instrumenting the original programs. If there is a fault in the original programs, the validity of the collected profiles may be threatened by the fault. The threat can slightly impact our conclusion.

- **External validity**
  Our results only rely on five medium-sized subject programs. The five programs including the real faults and the seeded faults are from different domains with different characteristics, e.g. different number of faults, different number of functions, and different sizes of test suites. They may not be representative of all other programs. The threat can be addressed by selecting larger scale and more representative industrial programs in future work.

- **Construct validity**
  For measuring the effectiveness of different test case reduction techniques, as previous studies, the measure (i.e. fault detection effectiveness) is introduced. This may be insufficient for evaluating the effectiveness of different methods because different methods have different execution costs. In order to objectively evaluate different reduction techniques, the study analyzed time costs from two aspects,
i.e. computational complexity and the real time required for generating a reduced test suite. Also, we did not distinguish the severities of different faults. The severities of faults might influence on the effectiveness of different reduction techniques.

7. Conclusions

Regression test case reduction aims at selecting a subset for the original test suite, while retaining the fault detection capability as much as possible. In previous studies, cluster analysis of function coverage vectors has been proposed. The vector-based similarity approaches do not always generate an efficient subset of test cases, as they do not consider sequential information and the relations between function calls.

For this purpose, we presented cluster analysis of function call sequences for test case reduction. Our approaches were experimentally compared with other reduction techniques including vector-based, random and greedy function-coverage-based maximization techniques. The experimental results indicate that sequence-based approaches can outperform vector-based and random test case reduction techniques with small sampling size in terms of the fault detection effectiveness. The results suggest that sequential information and the relations between function calls can aid to improve the fault detection effectiveness. With respect to the cost-effectiveness, cluster analysis of function call sequences measured using Euclidean distance is more effective than using the Levenshtein distance. Moreover, the study provides a good candidate for practical use. The cluster-based reduction techniques provide a more flexible approach for selecting a representative subset from the original test pool, i.e. the size of the subset can be assigned by testers according to the constrained testing resource.

For future work, our study will be extended to large scale industrial projects to validate the effectiveness of our approach. The order of test case execution will also be considered to improve the effectiveness of test case prioritization even further. This study is based on the assumption that all test cases have the same execution costs. The assumption may not hold in practice. Consequently, we will also take into account the influences of execution costs on the performance of reduced test suites. Last but not least, in these experiments, we find that with the same sampling proportion some faults detected by our approach cannot be detected by the greedy coverage maximization, and vice versa. This suggests that coverage-based and similarity-based test case reduction techniques should be integrated more tightly so as to drive new generation of test case reduction algorithms.

Acknowledgments

The authors are grateful to Paolo Tonella, from Fondazione Bruno Kessler (FBK), for helpful discussions and valuable suggestions.
References


