Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 8343910, 19 pages
http://dx.doi.org/10.1155/2016/8343910

Research Article

Empirical Study of the Effects of Different Similarity Measures on Test Case Prioritization

Rongcun Wang,[1] Shujuan Jiang,[1] Deng Chen,[2] and Yanmei Zhang[1]

[1]*School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu 221116, China*
[2]*Hubei Provincial Key Laboratory of Intelligent Robot, Wuhan 430073, China*

Correspondence should be addressed to Rongcun Wang; rcwang@cumt.edu.cn

Received 18 January 2016; Accepted 7 April 2016

Academic Editor: Emilio Insfran

Similarity-based test case prioritization algorithms have been applied to regression testing. The common characteristic of these algorithms is to reschedule the execution order of test cases according to the distances between pair-wise test cases. The distance information can be calculated by different similarity measures. Since the topologies vary with similarity measures, the distances between pair-wise test cases calculated by different similarity measures are different. Similarity measures could significantly influence the effectiveness of test case prioritization. Therefore, we empirically evaluate the effects of six similarity measures on two similarity-based test case prioritization algorithms. The obtained results are statistically analyzed to recommend the best combination of similarity-based prioritization algorithms and similarity measures. The experimental results, confirmed by a statistical analysis, indicate that Euclidean distance is more efficient in finding defects than other similarity measures. The combination of the global similarity-based prioritization algorithm and Euclidean distance could be a better choice. It generates not only higher fault detection effectiveness but also smaller standard deviation. The goal of this study is to provide practical guides for picking the appropriate combination of similarity-based test case prioritization techniques and similarity measures.

1. Introduction

Regression testing executes an existing test suite on a changed program to assure that the changes cannot adversely impact the original components. With the continual evolution of software systems, the size of a test suite usually grows very large [1]. It may be infeasible to reexecute the whole test suite within limited time and resources. Regression testing can be time-consuming and expensive. It accounts for as much as half of the cost of software maintenance [2]. Many techniques, such as test case selection [3, 4], test case minimization [5, 6], and test case prioritization [7, 8], have been proposed to speed up regression testing. Test case selection and reduction improve the effectiveness of regression testing by executing a subset of the original test suite. Test case prioritization rearranges the execution order of test cases so that the test cases with higher priority are executed earlier. In this way bug fixing and debugging can start early so as to reduce the cost of software maintenance. Particularly, when the cost of executing the whole test suite is huge, regression testing can get significant benefits from test case prioritization [9].

Actually, test case prioritization aims at finding an execution order which would detect faults fastest, that is, maximizing the rate of fault detection. The fault detection information cannot be known in advance before executing prioritized test suite. Therefore, test case prioritization techniques have to depend on surrogates, for example, code coverage information with different levels of granularity, expecting that early maximizing surrogate will result in increasing the rate of fault detection [10]. Test case prioritization can be applied to many scenarios, for example, general test case prioritization [11] and version specific prioritization [12]. General test case prioritization is not confined to the specific version of the software under test (SUT). Therefore, we only focus on general test case prioritization in regression testing.

The studies on test case prioritization mainly develop along two directions: coverage-based prioritization and

similarity-based prioritization. Most of coverage-based techniques resort to greedy [7, 11] or metaheuristic search algorithms [13, 14] to maximize the possible coverage. The underlying assumption of coverage-based prioritization techniques is that the execution order which achieves early more coverage is more likely to detect more faults. In general, greedy algorithms construct the optimal test case ordering according to the priority of each test case assigned by code coverage information. However, metaheuristic and evolutionary search algorithms utilize heuristics to seek solutions within the search space, that is, the set of all possible ordering of the original test suite. The original test suite often contains some test cases which are designed for exercising production features or exceptional behaviors, rather than for code coverage [15]. In this sense code coverage might be inadequate to ensure the achievement of higher rate of fault detection [16, 17].

More recently, similarity-based test case prioritization techniques have been developed. As the most representative prioritization techniques, clustering-based [18–20], ART-based [21–23], and other similarity-based test case prioritization techniques [24, 25] have aroused wide interest. The underlying assumption of similarity-based prioritization techniques is that test case diversity can aid in detecting more faults. The test cases within the same cluster are similar, but not the same in terms of fault detection capability. In other words, clustering-based prioritization techniques cannot guarantee the diversity of selected test cases to the maximum extent. Furthermore, most of clustering-based approaches depend on an important parameter, that is, the number of clusters. The parameter value has a great influence on the quality of clustering. Similarly, ART-based test case prioritization cannot also guarantee the diversity of test cases to the maximum extent.

Similarity measures have been widely applied to many applications, such as information retrieval and document clustering. More recently, similarity measures are also applied to software testing. Similarity-based test case prioritization techniques depend on similarity metrics. The topologies of different similarity metrics vary. The distances between pairwise test cases may change with similarity metrics. However, previous studies on similarity-based test case prioritization techniques only use one of these similarity metrics to select or prioritize test cases. Few studies concern the effects of similarity metrics on regression test case prioritization. Moreover, the successful stories of similarity-based test case prioritization techniques inspire us to empirically evaluate more similarity measures. For this reason we take a deeper look into the effects of six similarity measures, including Jaccard Index (JI) [21], Gower-Legendre (GL), Sokal-Sneath (SS), Euclidean distance (ED) [18, 26, 27], cosine similarity (CS) [28], and proportional-binary distance metric (PD) [18, 19, 25], on similarity-based test case prioritization algorithms. Unlike clustering-based prioritization algorithms, the algorithms we investigated are nonparametric. This paper presents results from an empirical study that compares the effects of six similarity measures on two similarity-based test case prioritization algorithms applied to 8 C programs, ranging from 272 to 33545 lines of code. This study aims at providing practical guides for picking the appropriate combination of similarity-based test case prioritization algorithms and similarity measures.

Over all, the contributions of this paper can be summarized as follows:

(i) This study proposes a global similarity-based test case prioritization algorithm based on the comparison of similarity measures between test cases in a given test case suite. Compared with the ART-based prioritization algorithm, the global similarity-based prioritization algorithm generates more robust and better results.

(ii) This study empirically evaluates the effects of six similarity measures on two similarity-based test case prioritization algorithms. To the best of our knowledge, it is the first empirical study of the effects of different similarity measures on regression test case prioritization in the context of white-box testing.

(iii) The experimental results illustrate that Euclidean distance is more efficient in finding faults than other measures. It generates not only higher fault detection capability, but also smaller standard deviation. The combination of the global similarity-based prioritization algorithm and Euclidean distance could be the best choice.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 describes two similarity-based test case prioritization algorithms, while Section 4 introduces six similarity measures. Sections 5 and 6 present our empirical study as well as results analysis. The threat to validity is discussed in Section 7. The conclusion is drawn in Section 8, followed by our future work.

2. Related Work

In previous studies, researchers have proposed many test case prioritization techniques. Rothermel et al. [7] proposed several coverage-based techniques including statement coverage and branch coverage. Elbaum et al. [11] extended the techniques in [7] and presented coarse-granularity prioritization techniques, that is, function-coverage-based prioritization techniques. Most of coverage-based prioritization techniques primarily focused on two classical greedy algorithms: total greedy algorithm and additional greedy algorithm. Since the two greedy algorithms may produce suboptimal results, Li et al. [14] applied metaheuristic algorithms to code coverage-based prioritization. Their goal for test case prioritization is different from ours as we aim to increase fault detection rate, rather than code coverage. Additionally, Walcott et al. [13] proposed a time-aware prioritization technique based on a genetic algorithm to reorder test cases with given time constraints.

Ledru et al. [24] used string distances for test case prioritization, incorporating Hamming, Levenshtein, Cartesian, and Manhattan distances. They empirically evaluated the effects of string edit distances on test case prioritization. All test cases were viewed as strings. Before executing test cases, test

Table 1: Test case prioritization techniques investigated in this paper.

Label	Name	Description
T_1	Random	Random prioritization
T_2	Original	Original order of test cases generated
T_3	ART-bn	ART-based prioritization at branch coverage level
T_4	GSB-bn	Global similarity-based prioritization at branch coverage level
T_5	addtl-fn	Additional function coverage prioritization
T_6	addtl-st	Additional statement coverage prioritization
T_7	addtl-br	Additional branch coverage prioritization

cases were ordered based on the similarity of strings. In that sense our study is very similar to their study; that is, both of them evaluated the effects of different distance measures on test case prioritization. The most distinctive difference is that their method is applied to black-box testing; that is, test case prioritization is performed before executing the test cases. On the contrary, test case prioritization in our study is performed after executing the test cases. Moreover, the distance measures we applied are different from theirs. Similarly, Hemmati et al. introduced a family of similarity-based test case selection techniques for model-based testing [29]. They also analyzed the impact of similarity functions on similarity-based test case selection [30]. In model-based testing, test cases generated from UML state machines are abstract rather than concrete. The method proposed by Hemmati et al. is applied to select a subset of the generated test suite without considering the order of selected test cases. Unlike the above two methods, our approach uses dynamic profile information generated by executing test cases to reschedule the execution order of test cases.

As one of similarity-based prioritization techniques, clustering-based test case prioritization approaches have been also discussed. Yoo et al. [31] combined clustering with expert knowledge to achieve scalable prioritization. The process of prioritization depended on human efforts, which is different from our approach. Carlson et al. [20] implemented new prioritization techniques that incorporated a clustering approach and utilized three types of information, that is, code coverage, code complexity, and history data on real faults. The experimental results showed that test case prioritization that utilizes a clustering approach can improve the effectiveness of test case prioritization techniques. Dickinson et al. [18] presented distribution-based filtering and prioritizing techniques incorporating sampling methods.

More recently, similarity-based algorithms have been applied to regression test case prioritization. Ramanathan et al. [32] used Levenshtein distance to similarity-based test prioritization. Jiang et al. [21] proposed a new family of coverage-based ART techniques, which were statistically superior to the RT-based techniques in detecting faults. Fang et al. [25] introduced several similarity-based test case prioritization techniques based on the edit distances of ordered sequences. The execution profiles of test cases were transformed into the ordered sequences of program entities (e.g., statement or branch). The distance of two test cases was defined as the edit distance of their ordered sequences calculated by the proportional distance. Differently from Jiang et al. [21], Fang et al. [25], and Ramanathan et al. [32], where only one similarity measure was used, we empirically evaluated six similarity measures.

Table 2: Motivating example.

Test case	Branch							
	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
t_1	√	√	√				√	√
t_2					√	√	√	√
t_3	√	√	√			√	√	√
t_4	√	√	√	√				

3. Similarity-Based Test Case Prioritization

This section describes two similarity-based test case prioritization algorithms. As comparing baselines, coverage-based test case prioritization techniques are also discussed. Table 1 lists test case prioritization techniques investigated in this paper.

3.1. Motivating Example. A simple example, based on branch coverage, is presented in Table 2. Equation (1) shows the distance matrix calculated by Euclidean distance. Coverage-based test case prioritization techniques aim to achieve full branch coverage as soon as possible. The test case ordering generated by total branch coverage prioritization is S_1: $\langle t_3, t_1, t_2, t_4\rangle$ or S_2: $\langle t_3, t_1, t_4, t_2\rangle$. The test case ordering generated by additional branch coverage prioritization generates is S_3: $\langle t_3, t_2, t_4, t_1\rangle$ or S_4: $\langle t_3, t_4, t_2, t_1\rangle$. However, the optimal test case orderings for this example are S_5: $\langle t_2, t_4, t_3, t_1\rangle$ and S_6: $\langle t_4, t_2, t_3, t_1\rangle$. Although the above two coverage-based prioritization techniques are proposed from a coverage standpoint, they cannot cover all branches as soon as possible:

$$\begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 0.0 & & & \\ 2.2 & 0.0 & & \\ 1.0 & 2.0 & 0.0 & \\ 1.7 & 2.8 & 2.0 & 0.0 \end{pmatrix} \end{array}. \quad (1)$$

Input: A test suite T: $\{t_1, t_2, \ldots, t_n\}$ with n test cases
Output: A prioritized ordering P: $\langle p_1, p_2, \ldots, p_n\rangle$ of test suite T
(1) $P \leftarrow \emptyset$;
(2) randomly select one test case t from T;
(3) $P \leftarrow \langle t\rangle$;
(4) $T \leftarrow T\backslash\{t\}$;
(5) **while** $T \neq \emptyset$ **do**
(6) $C \leftarrow \emptyset$; //C represents a candidate set;
(7) PE $\leftarrow \emptyset$; //PE saves the program entities covered by test cases belonging to C;
(8) PE$'$ $\leftarrow \emptyset$;
(9) $U \leftarrow T$; //U temporarily saves all test cases that are not prioritized;
(10) **repeat**
(11) randomly select one test t_i from U;
(12) PE $\leftarrow$ PE $\cup$ PE$'$; //PE$'$ saves the program entities covered by t_i;
(13) $C \leftarrow C \cup \{t_i\}$;
(14) $U \leftarrow U\backslash\{t_i\}$;
(15) **until** (PE$'$ $\cup$ PE == PE) or ($U == \emptyset$);
(16) **for** test case $c_i \in C$ **do**
(17) calculate the distance between c_i to P;
(18) **end**
(19) select test case c_i that is the farthest away from P;
(20) $p \leftarrow \langle p_1, p_2, \ldots, c_i\rangle$;
(21) $T \leftarrow T\backslash\{t\}$;
(22) **end**
(23) **return** P

Algorithm 1: ART-based prioritization algorithm.

3.2. ART-Based Prioritization Algorithm. Adaptive Random Testing (ART) is proposed initially by Chen et al. [33] to substitute random testing for test case generation. Previous experimental studies have shown that ART can aid in further improving the fault detection effectiveness over random testing. The diversity of test cases in the input domain is the foundation of the improvement. Based on this, Jiang et al. [21] used the idea of ART to regression test case prioritization. Thus, they proposed an ART-based prioritization algorithm, which preserves the partial diversity of code coverage information. Its pseudocode is described in Algorithm 1.

First, this algorithm randomly selects one test case t from the original test suite T as the first element of a prioritized ordering P and deletes t from T (Line (1)–Line (4)). And then, a candidate set of test cases C is constructed according to the code coverage of test cases in T. The process of constructing a candidate set (Line (10)–Line (15)) is repeated until the newly added test case does not increase program entities (e.g., function, statement, or branch) coverage. Finally, the test case in C that is the farthest away from the prioritized ordering P is preferentially selected and added to the tail of P (Line (16)–Line (20)). The prioritization process is repeated until all test cases in T have been prioritized (Line (5)–Line (22)).

Determining the distance between a test case t and a set of test cases S is a key point of this algorithm. The distance is defined as follows:

$$d(t, S) = \min\left\{d\left(t, t_i\right) \mid t \in C,\ t_i \in S\right\}. \quad (2)$$

The distance is called the minimum set distance [34], that is, the minimum distance between pair-wise test cases. The distance can be calculated by a certain distance measure. Similarly, the average or maximum set distance can be also defined. We use the minimum set distance as previous studies have empirically verified that it shows good performance.

Given a test suite with m test cases and a program with n statements, this algorithm requires times $O(m^2)$ and $O(m^3n)$ in the best and worst cases, respectively.

3.3. Global Similarity-Based Prioritization Algorithm. The diversity of test cases aids in improving their fault revealing power [29, 30]. ART-based prioritization algorithm selects a test case that is the farthest from the prioritized test suite from a candidate set. Since the candidate set is a subset of not yet prioritized test cases, this algorithm does not guarantee the diversity of test cases to a maximum extent. Inspired by Hemmati et al. [29], we propose a global similarity-based test case prioritization algorithm. This algorithm selects a test case from all not yet prioritized test cases, rather than a subset of them. Consequently, it can maximize the diversity of prioritized test cases. Its pseudocode is described in Algorithm 2.

First, this algorithm selects test case t_k that has the maximum average distance and deletes it from T. Test case t_k is viewed as the most different from all other test cases. Test case t_k is added to the tail of prioritized ordering P. Then the distance from each test case t in T to t_k is viewed as the minimal set distance from t to P. The process of test case prioritization mainly includes two steps. The first step is to seek a test case t_i from T that is the farthest from the last test case in P (Line (16)–Line (21)). Test case t_i is added to the

```
     Input: A test suite T: {t_0, t_1, ..., t_{n-1}} with n test cases
     Output: A prioritized sequence P: ⟨p_0, p_1, ..., p_{n-1}⟩ of test suite T
(1)  calculate the distance between pair-wise test cases in T;
(2)  P ← ∅;
(3)  double dist_min[n];
(4)  for i ← 0 to n − 1 do
(5)      dist_min[i] = 0.0;
(6)  end
(7)  select test case t_k that has the maximum average distance from T;
(8)  max ← k;
(9)  T ← T\{t_max};
(10) P ← ⟨t_max⟩;
(11) for i ← 0 to n − 1 do
(12)     dist_min[i] = dist(t_i, t_max);
(13) end
(14) repeat
(15)     max_dist = dist_min[0];
(16)     for i ← 0 to n − 1 do
(17)         if dist_min[i] > max_dist and t_i ∈ T then
(18)             max_dist = dist_min[i];
(19)             max ← i;
(20)         end
(21)     end
(22)     add t_max to the tail of P;
(23)     T ← T\{t_max};
(24)     for i ← 0 to n − 1 do
(25)         if dist_ min[i] > dist(t_i, t_max) and t_i ∈ T then
(26)             dist_min[i] = dist(t_i, t_max);
(27)         end
(28)     end
(29) until T is empty;
(30) return P
```

Algorithm 2: Global similarity-based prioritization algorithm.

tail of P and deleted from T. The second step is to update the minimal set distance from each test t in T to P by comparing the distances from t to P before and after adding t_i to P (Line (24)–Line (28)). The process of prioritization is repeated until T is empty (Line (14)–Line (29)).

The most expensive operation of this algorithm is the calculation of the distances between pair-wise test cases. Note that this algorithm only compares the distances between not yet prioritized test cases and prioritized test cases, rather than recalculating their distances in the process of test case prioritization. In this sense, this algorithm reduces the cost of prioritization. Given a test suite with m test cases and a program with n branches, this algorithm requires time $O(m^2n)$. Compared with the ART-based prioritization algorithm, this algorithm significantly decreases the time complexity. Also, this algorithm is scalable; that is, once the new test cases are generated, the distances among the original test cases are not recalculated.

From the above example, the average distance from test case t_1 to other test cases is $(2.2 + 1.0 + 1.7)/3$. The average distances of test casees t_2, t_3, and t_4 are $(2.2 + 2.0 + 2.8)/3$, $(1.0 + 2.0 + 2.0)/3$, and $(1.7 + 2.8 + 2.0)/3$, respectively. Since test case t_2 has the maximum average distance, it is the first element in prioritized ordering S. The distances from test cases t_1, t_3, and t_4 to t_2 are 2.2, 2.0, and 2.8, respectively. Since test case t_4 has the maximum distance to test case t_2, test case t_4 is the second element in prioritized ordering S. The distances from test cases t_1 and t_3 to test case t_4 are 1.7 and 2.0, respectively. Since 1.7 is less than 2.2, the minimum set distance from test case t_1 to prioritized ordering S is 1.7. The minimum set distance from test case t_3 to prioritized ordering S is 2.0. Since 1.7 is less than 2.0, test case t_3 is the third element. Finally, test case t_1 is added to the tail of S. The prioritized ordering generated by the global similarity-based test case prioritization technique is S_{10}: $\langle t_2, t_4, t_3, t_1\rangle$; that is, it is the optimal test case ordering. Compared with other test case prioritization techniques we studied, the global similarity-based test case prioritization technique can cover all branches more quickly.

3.4. Coverage-Based Prioritization Techniques. Coverage-based prioritization techniques have been widely studied. These techniques can usually be implemented at three different coverage levels: function, statement, and branch.

(i) Total Coverage Maximization Prioritization. Total coverage maximization prioritization techniques schedule test cases in descending order of total coverage achieved. When more

than one test case has the same total coverage, the algorithm selects one at random. The prioritization process is repeated until all test cases have been prioritized.

(ii) Additional Coverage Maximization Prioritization. Additional coverage maximization prioritization techniques work as follows: A test case that yields the greatest coverage is selected at first. And then, the next test case to be selected is the one that can cover the maximum program entities not yet covered by previously selected test cases. The process is repeated until all program entities have been covered by at least one test case.

Having prioritized test cases in this manner, if remaining test cases cannot add the additional coverage, we schedule them in the same way. If more than one test case yields the same additional coverage in the process of prioritization, we randomly select one. Additional coverage maximization prioritization is based on feedback. In other words, previously selected test cases are used to provide guidance to select next test case. On the contrary, total coverage maximization prioritization is no feedback. For a test suite and program with m test cases and n statements, respectively, total coverage maximization prioritization requires time $O(mn)$, while the cost of additional coverage maximization prioritization is $O(m^2n)$.

3.5. Other Prioritization Techniques

(i) Random Prioritization. We apply random prioritization, that is, randomly scheduling all test cases in the original test suite, to facilitate our empirical study.

(ii) No Prioritization. As a base comparison, the initial order of test cases generated is also considered.

4. Similarity Measures

In this section, we describe the similarity measures incorporating Jaccard Index, Gower-Legendre, Sokal-Sneath, Euclidean, cosine similarity, and proportional-binary distance metric. Moreover, we also give the semantic explanations for the application of these measures.

4.1. Profile Representation. Let E: $\langle e_1; e_2; \ldots; e_n\rangle$ represent the set of all program entities in SUT, where e_i is the ith program entity. Profile information, that is, all program entities covered by a test case, can be represented as a binary coverage vector V: $\langle v_1, v_2, \ldots, v_n\rangle$. If e_i is covered, v_i is equal to 0, 1 otherwise. Similarly, the vector can also be implemented with numeric entries; that is, v_i represents the number of times that e_i is executed. Profile information of both numerical input and nonnumeric input can be represented into coverage vector. In this sense, the distance between pair-wise vectors is considered as the distance between the corresponding test cases.

4.2. Jaccard Index and Its Variants. Suppose that the set of program entities covered by test cases t_1 and t_2 are E_{t_1} and E_{t_2}, respectively. We define the similarity between test cases t_1 and t_2 based on Jaccard Index and its variants (i.e., Gower-Legendre and Sokal-Sneath) in the general formula as follows:

$$s(t_1, t_2) = \frac{\left|E_{t_1} \cap E_{t_2}\right|}{\left|E_{t_1} \cap E_{t_2}\right| + w\left(\left|E_{t_1} \cup E_{t_2}\right| - \left|E_{t_1} \cap E_{t_2}\right|\right)}, \quad (3)$$

where $|E_{t_1} \cap E_{t_2}|$ is the number of matched program entities between E_{t_1} and E_{t_2}, while the unmatched pairs between them is $(|E_{t_1} \cup E_{t_2}| - |E_{t_1} \cap E_{t_2}|)$. Equation (3) computes the number of program entities covered by t_1 and t_2 directly, while the unmatched pairs between E_{t_1} and E_{t_2} are weighted based on their contribution to the similarity. When w is equal to 1, the above formula is called Jaccard Index. If $w = 2$ and 1/2, this formula is called Sokal-Sneath measure and Gower-Legendre measure, respectively. For Jaccard Index and its variants, the corresponding distance is $d(t_1, t_2) = 1 - s(t_1, t_2)$.

4.3. Euclidean Distance. Let V_1: $\langle v_{11}, v_{12}, \ldots, v_{1n}\rangle$ and V_2: $\langle v_{21}, v_{22}, \ldots, v_{2n}\rangle$ represent two binary coverage vectors generated by executing test cases t_1 and t_2. The Euclidean distance between test cases t_1 and t_2 can be defined as follows:

$$d(t_1, t_2) = \sqrt{\sum_{i=1}^{n} (v_{1i} - v_{2i})^2}. \quad (4)$$

4.4. Proportional Distance. Let P: $\langle p_1, p_2, \ldots, p_n\rangle$ and Q: $\langle q_1, q_2, \ldots, q_n\rangle$ stand for two coverage vectors implemented with numeric entries by executing test cases t_1 and t_2. The number of times that each program entity was executed is normalized so as to eliminate the variation of absolute frequency. The proportional distance between test cases t_1 and t_2 can be defined as follows:

$$d(t_1, t_2) = \sqrt{\sum_{i=1}^{n} \left(\frac{|p_i - q_i|}{\max_i - \min_i}\right)^2}, \quad (5)$$

where $\max_i$ and $\min_i$ represent the maximum and minimum numbers of times that the ith program entity is executed in all profiles, respectively. When the difference between $\max_i$ and $\min_i$ is equal to 0, that is, the ith program entity cannot be covered or the number of times that it is executed is the same, the corresponding item is also equal to 0.

4.5. Cosine Similarity. The binary coverage vectors covered by test cases t_1 and t_2 are X: $\langle x_1, x_2, \ldots, x_n\rangle$ and Y: $\langle y_1, y_2, \ldots, y_n\rangle$, respectively. The similarity between test cases t_1 and t_2 can be defined as follows:

$$s(t_1, t_2) = \frac{x^t \cdot y}{\|x\| \|y\|}, \quad (6)$$

where x^t is a transposition of vector x and $\|x\|$ is the Euclidean norm of vector x. The Euclidean norm of vector x can be calculated according to $\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$. Similarly, $\|y\|$ is the Euclidean norm of vector y. In essence, s is the cosine of the angle between vectors x and y, which is a nonmetric measure [35]. For cosine similarity, the corresponding dissimilarity is defined as $d(t_1, t_2) = 1 - s(t_1, t_2)$.

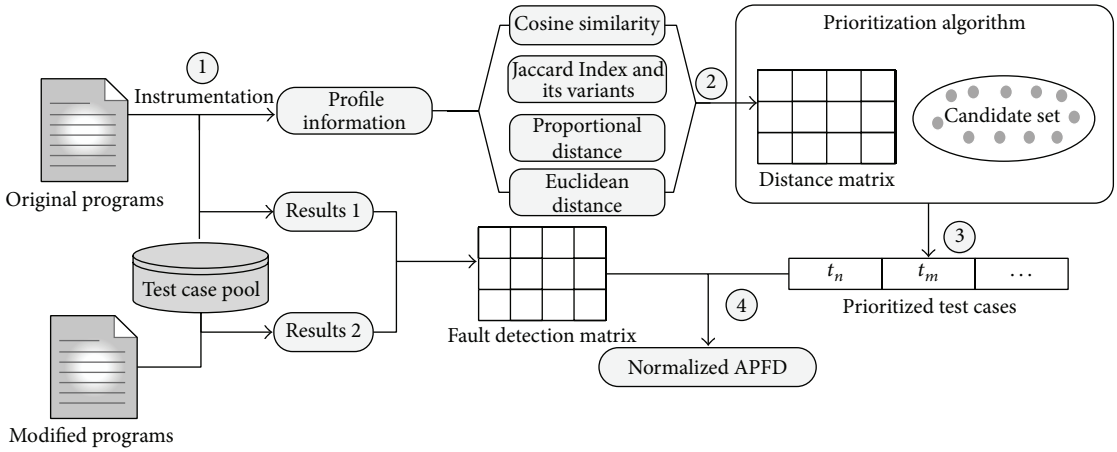


Figure 1: Overview of similarity-based test case prioritization.

5. Empirical Study

Similarity-based test case prioritization techniques mainly rely on three variables: coverage criteria (e.g., function, statement, and branch), similarity functions, and prioritization algorithms. Previous studies have shown that fine-granularity coverage techniques can aid in improving rate of fault detection [6, 11]. Therefore, we only concerned similarity-based prioritization techniques at branch coverage level. We empirically investigated 12 variants (6 similarity measures × 2 prioritization algorithms) of similarity-based regression test case prioritization techniques over every subject program. Additionally, as comparison baselines, 3 variants of coverage-based prioritization techniques were also empirically studied. All algorithms are performed on a Ubuntu 11.10 system running on an Intel® Core™ i3 CPU 3.10 GHz with 4 GB main memory.

5.1. Overview. Figure 1 summarizes the implementation process of similarity-based test case prioritization techniques, which mainly includes four steps.

(1) Instrumentation. With the dynamic instrumentation tool `gcov` (https://gcc.gnu.org/), we collect execution profiles and construct branch coverage vectors. Compared to static instrumentation, dynamic instrumentation only needs to write a small amount of code, so it significantly improves the flexibility of testing. The flexibility is improved at the expense of a small increase of the time overhead.

(2) Distance Calculation. The distance between pair-wise test cases is calculated by a certain distance measure.

(3) Test Case Prioritization. Test cases are prioritized based on the distance between pair-wise test cases by different prioritization strategies.

(4) Evaluation. The fault detection matrix is constructed based on the relation between faults and test cases for every subject program. A column and row indicate one fault and test case, respectively. If a fault can be detected by a test case, the corresponding element to the test case and fault in the matrix is equal to 1, 0 otherwise. The fault detection effectiveness of a prioritized test suite is evaluated according to its corresponding fault detection matrix.

5.2. Research Questions. In the empirical study, we address the following four specific research questions.

RQ1. Do different similarity measures have significant effects on the ART-based test case prioritization technique? The question helps us to seek which similarity measures are the most effective in the context of the ART-based test case prioritization technique.

RQ2. Do different similarity measures have significant effects on the global similarity-based test case prioritization technique? Similar to *RQ1*, the question helps us to identify which similarity measures are the most effective in the context of the global similarity-based test case prioritization technique.

RQ3. Does the global similarity-based test case prioritization outperform the ART-based test prioritization in terms of normalized average percentage of fault detection (NAPFD)? The answer to this question helps us to make a choice of test case prioritization techniques when only a part of test suite prioritized is executed. In other words, we empirically determine which combinations are the most effective test case prioritization with reduction in terms of NAPFD.

RQ4. Can the most effective similarity-based prioritization techniques be as effective as coverage-based prioritization techniques? Coverage-based test case prioritization has been empirically verified to be a promising technique. We can determine which variant of test case prioritization is more effective in terms of NAPFD.

5.3. Experiment Programs. Our experiments were conducted over eight subject programs from the Software Infrastructure Repository (SIR) [36]. All the eight programs are written

Table 3: Descriptive information for subject programs.

Program name	Number of version	Lines of code	Test suite size
gzip	14	3883–5096	217
make	33	12609–17153	1043
sed	18	4711–9204	370
totinfo	23	272-273	1052
schedule	8	290–294	2650
schedule2	10	261–263	2710
print_tokens	7	341–343	4130
print_tokens2	10	350–355	4115

in the C language. These programs contain 5 small-sized Siemens programs and 3 medium-sized UNIX programs with real and seeded faults. Descriptive information about the selected programs is presented in Table 3, where the lines of code are calculated by the tool "SLOCCount" (http://www.dwheeler.com/sloccount/).

sed, *gzip*, and *make* are all UNIX utilities (http://directory.fsf.org/wiki/GNU). *sed* is a stream-oriented noninteractive text editor. It consists of a base version and 5 modified versions. Only 18 faults are selected from 32 seeded faults. *make* is used to compile and link programs to generate executables. It consists of a base version and 4 modified versions. We select 33 faults in all. *gzip* is used to compress and decompress files. Some faults, such as f_1 and f_6 in version v_2, are eliminated as these faults can be detected by more than 20% of test cases. Only 14 faults are selected from 59 faults from the program *gzip* in total. In a word, we eliminate the fault versions whose faults cannot be detected by any test case. Likewise, if a fault can be detected by more than 20% of test cases, the fault is also excluded.

The other 5 subject programs are the Siemens programs: *schedule* and *schedule2* are priority schedulers, *printtokens* and *printtokens2* are lexical analyzers, and *totinfo* computes statistics given input data. The number of versions, that is, the number of faults selected for every subject program, is presented in the second column of Table 3.

5.4. Evaluation Metrics. The average percentage of faults detected (APFD) [37] is commonly used to evaluate the effectiveness of a prioritization technique implemented on a whole test suite. Let T be a test suite containing n test cases and let F be a set of m faults exposed by T. Let TF_i be the first test case in a prioritized test suite that detects fault i. APFD is defined as follows:

$$\text{APFD} = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n}. \quad (7)$$

The value of APFD ranges from 0 to 1. The larger the value of APFD, the higher the effectiveness of prioritized technique. The application of APFD assumes that the whole test suite can be run and find all of the faults. Only a part of test suite is often executed in regression testing under limited resources [6, 38]. The reduced test suite could reduce the capability of fault detection. In this sense, APFD is unsuitable for measuring the effectiveness of a prioritization technique implemented on a part of test suite. Therefore, we use a metric, called Normalized APFD [39], which includes information on both fault finding and time of detection. Let TF_i be the first test case in the prioritized and reduced test suite T' of T that detects fault i. Let n' be the size of T'. Normalized APFD is defined as follows:

$$\text{NAPFD} = p - \frac{TF_1 + TF_2 + \cdots + TF_m}{n'm} + \frac{p}{2n'}, \quad (8)$$

where p represents the quotient of the number of faults detected by the prioritized and reduced test suite divided by the number of faults detected by the whole test suite. If a fault i is never detected by T', TF_i is set to 0. If $n' = n$, that is, all test cases can be executed and S' can detect all faults in F, APFD and NAPFD are equivalent.

Table 4: An example of the fault detection capability of test suite T.

Test case	Fault			
	f_1	f_2	f_3	f_4
t_1				
t_2		√	√	
t_3		√		
t_4	√			
t_5	√		√	√

We give a simple example so as to intuitively understand the difference between APFD and NAPFD. Table 4 shows the relation between test suite T and fault set F.

There are two test case prioritization methods M_1 and M_2, respectively. Test case orderings M_1 and M_2 generated are S_{M_1}: $\langle t_3, t_4, t_1, t_2, t_5\rangle$ and S_{M_2}: $\langle t_3, t_1, t_5, t_4, t_2\rangle$, respectively. The APFD values are $S_{M_1}(\text{APFD}) = 1 - (1 + 2 + 4 + 5)/(4\times5) + 1/(2\times5)$ and $S_{M_2}(\text{APFD}) = 1 - (1+3+3+3)/(4\times5) + 1/(2\times5)$, that is, 0.5 and 0.6. If only two test cases can be executed under limited resources, the NAPFD values of them are $S_{M_1}(\text{NAPFD}) = 2/4 - (1 + 2)/(2\times4) + (2/4)/(2\times2)$ and $S_{M_2}(\text{NAPFD}) = 1/4 - 1/(2\times4) + (1/4)/(2\times2)$, that is, 0.25 and 0.1875. If the whole test suite can be excused, M_2 is better than M_1. When only two test cases are excused, we get the opposite results. From the above example, the application of NAPFD may be more suitable for evaluating test case prioritization than that of APFD.

Since all prioritization algorithms in this study have the nature of randomness, we repeated 100 times for every prioritization algorithm with different random seeds. The mean value of normalized average percentage of fault detection for every sample was calculated.

6. Experimental Results

We focus on test case prioritization with reduction. The main reasons include two aspects: one is that regression testing is usually executed under constrained resources; that is, only parts of test cases are executed; the other is that only evaluating the effectiveness of a whole test suite may hide important information on test case prioritization techniques.

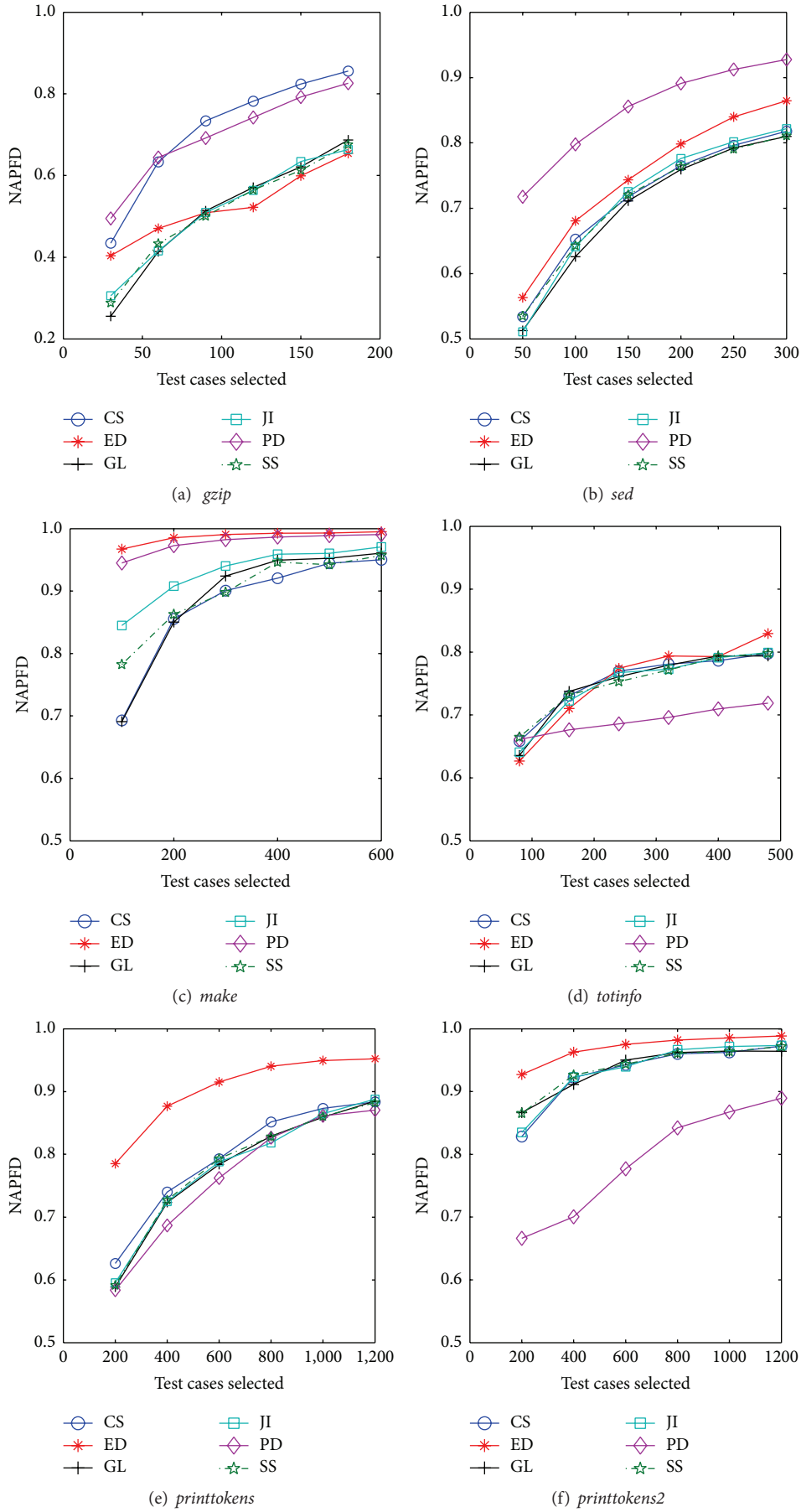
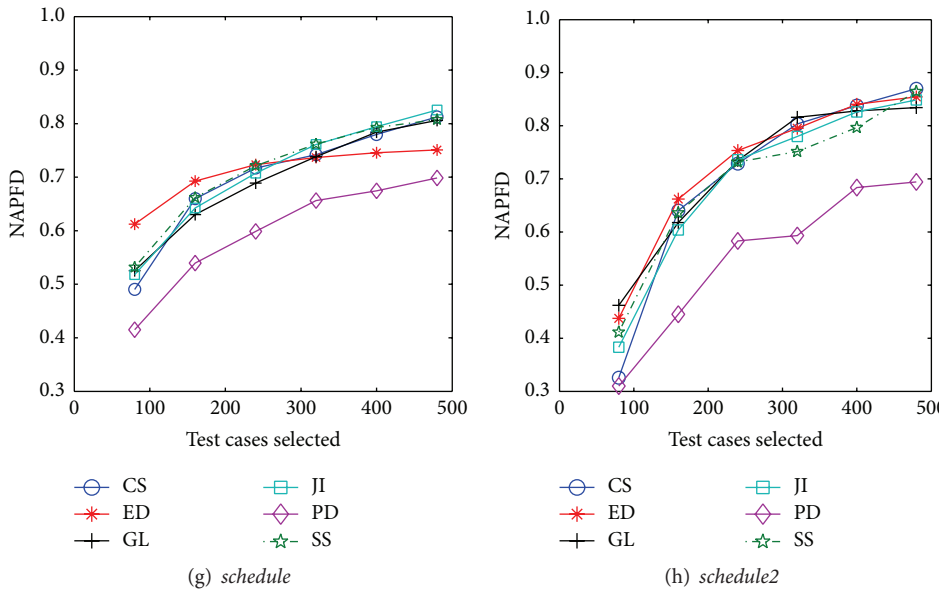


Figure 2: Continued.



(g) *schedule* (h) *schedule2*

Figure 2: NAPFDs of the ART-based prioritization algorithm using different distance measures.

For example, the top 400 test cases generated by ART-based test case prioritization can detect all faults over the program *print_tokens*. The size of the original test suite in the program *print_tokens* is 4130. The number of faults seeded is only 7. When we evaluated the NAPFD of the original test suite, the NAPFD is very close to 1. Similarly, the NAPFD generated by other prioritization techniques is also close to 1. In other words, there is no statistically significant difference between different prioritization techniques. Therefore, we took 6 samples starting from 5% to 30% with the increment of 5% for every program in order to look deeper into the statistical difference between different prioritization techniques.

6.1. Experiment 1: Evaluating the Effects of Six Similarity Measures on the ART-Based Prioritization Algorithm. Figure 2 shows our experimental results on the ART-based test case prioritization algorithm using six similarity measures. In Figure 2, each figure shows the mean values in NAPFD for six similarity measures. The x-axis of each graph indicates the number of tests selected, while the y-axis indicates the mean value in NAPFD.

From Figure 2, we find that Euclidean distance performs better than other similarity measures in terms of the ART-based prioritization algorithm. Cosine similarity, Gower-Legendre, and Sokal-Sneath are comparable to Jaccard Index with respect to NAPFD. Compared with other similarity distances, the proportion distance produced lower NAPFDs with the same sampling proportion.

Having seen Figure 2, we further conducted one-way analysis of variances (ANOVAs) [40] to verify whether the mean values of NAPFD distributions for different similarity measures differ significantly. For each subject programs, a null hypothesis is defined to state that there is no statistically significant difference between different similarity measures at 5% significance level. If the p value is less than 0.05, we successfully reject the null hypothesis. In other words, there is statistically significant difference between different similarity measures. By ANOVAs, we find that the p value was less than 0.05 for every sample over every subject program; that is, different measures have significant effects on the ART-based test case prioritization algorithm.

We calculated the standard deviation of every sample to observe which measures can generate more stable results. The standard deviations are shown in Table 5. From Table 5, we find that Euclidean distance generated smaller standard deviation than other similarity measures with the same sampling proportion. Jaccard Index generated smaller standard deviations than its variants. Cosine similarity was better than Jaccard Index over medium-sized programs, while they are very close over small-sized programs. In general, the standard deviation decreases gradually with the increase of the sampling proportion. The possible explanation is that the higher the sampling proportion, the more the detected faults.

In addition, we further conducted the nonparametric Mann-Whitney U test [41, 42] to qualitatively compare the statistical differences between each pair of similarity measures at 5% significance level. For the nonparametric Mann-Whitney U test, the null hypothesis is defined to state that there is no difference between the results generated by each pair of similarity measures. When we reject the null hypothesis at a significance level $\alpha = 0.05$, we say that two similarity measures are statistically different. Table 6 summarizes the statistical significance of differences between each pair of similarity measures, where each element (m, n) denotes the "win/tie/loss" (win: the number of times that the mth row measure performs significantly better than nth column measure; tie: the number of times that there is no significant difference between the mth row measure and the

Table 5: The standard deviations of different measures generated by the ART-based prioritization algorithm.

Program	Measure	Standard deviation					
		$N = 1$	$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$
gzip	JI	0.11	0.10	0.09	0.08	0.07	0.07
	CS	0.07	0.07	0.06	0.04	0.04	0.03
	ED	0.06	0.06	0.04	0.05	0.04	0.03
	GL	0.12	0.10	0.11	0.09	0.08	0.09
	PD	0.11	0.13	0.13	0.12	0.13	0.10
	SS	0.13	0.11	0.10	0.10	0.09	0.08
sed	JI	0.04	0.04	0.03	0.02	0.02	0.02
	CS	0.04	0.04	0.03	0.02	0.02	0.01
	ED	0.04	0.04	0.03	0.03	0.03	0.01
	GL	0.05	0.04	0.03	0.03	0.03	0.02
	PD	0.04	0.03	0.03	0.03	0.02	0.02
	SS	0.05	0.04	0.03	0.02	0.02	0.02
make	JI	0.20	0.12	0.07	0.09	0.05	0.04
	CS	0.06	0.05	0.03	0.02	0.02	0.01
	ED	0.07	0.03	0.02	0.02	0.01	0.00
	GL	0.29	0.16	0.05	0.05	0.06	0.04
	PD	0.19	0.11	0.07	0.07	0.06	0.05
	SS	0.25	0.18	0.12	0.05	0.06	0.05
totinfo	JI	0.07	0.04	0.03	0.03	0.03	0.03
	CS	0.06	0.03	0.02	0.02	0.02	0.02
	ED	0.07	0.06	0.05	0.05	0.04	0.03
	GL	0.06	0.04	0.03	0.02	0.02	0.02
	PD	0.13	0.09	0.08	0.08	0.07	0.06
	SS	0.06	0.04	0.03	0.02	0.02	0.03
schedule	JI	0.11	0.09	0.10	0.10	0.08	0.09
	CS	0.11	0.10	0.10	0.08	0.09	0.09
	ED	0.07	0.04	0.03	0.03	0.01	0.01
	GL	0.15	0.12	0.09	0.10	0.09	0.09
	PD	0.11	0.15	0.15	0.14	0.13	0.11
	SS	0.13	0.12	0.09	0.10	0.09	0.09
schedule2	JI	0.16	0.14	0.12	0.10	0.07	0.07
	CS	0.16	0.14	0.11	0.10	0.08	0.06
	ED	0.11	0.08	0.08	0.07	0.08	0.07
	GL	0.15	0.14	0.12	0.09	0.08	0.08
	PD	0.18	0.20	0.18	0.18	0.16	0.16
	SS	0.18	0.14	0.12	0.10	0.09	0.07
print_tokens	JI	0.11	0.09	0.08	0.08	0.07	0.05
	CS	0.12	0.10	0.08	0.07	0.06	0.05
	ED	0.07	0.05	0.04	0.03	0.03	0.02
	GL	0.13	0.10	0.08	0.07	0.07	0.05
	PD	0.12	0.10	0.07	0.07	0.07	0.05
	SS	0.13	0.10	0.07	0.07	0.06	0.05
print_tokens2	JI	0.05	0.04	0.03	0.02	0.02	0.01
	CS	0.07	0.04	0.02	0.02	0.02	0.01
	ED	0.03	0.02	0.01	0.01	0.01	0.00
	GL	0.07	0.04	0.02	0.02	0.02	0.02
	PD	0.06	0.04	0.04	0.03	0.02	0.01
	SS	0.06	0.04	0.03	0.02	0.02	0.01

Table 6: A summary of the statistical significance of differences between each pair of similarity measures over 48 different experimental settings (8 programs × 6 sampling rates based on the ART prioritization algorithm).

	CS	ED	GL	JI	PD
ED	(27/13/8)	—	—	—	—
GL	(3/29/16)	(6/15/27)	—	—	—
JI	(8/12/28)	(5/14/29)	(2/38/8)	—	—
PD	(12/15/21)	(13/9/26)	(17/12/19)	(18/11/19)	—
SS	(2/33/13)	(5/10/33)	(25/20/3)	(2/39/7)	(19/11/18)

*n*th column measure in terms of NAPFD; loss: the number of times that the *m*th row measure performs significantly worse than the *n*th column measure) value.

From Table 6, Euclidean distance performed better than other similarity measures based on the ART prioritization algorithm. For example, in the best case, Euclidean distance outperformed Sokal-Sneath on 33 settings, while Sokal-Sneath outperformed Euclidean distance only on 5 setting. In the worst case, Euclidean distance outperformed proportional distance on 26 settings, while the proportional distance outperformed Euclidean distance only on 13 settings.

By the nonparametric Mann-Whitney *U* test, we find that Euclidean distance performed very well over all subject programs. For the most of samples, there is statistically significant difference between Euclidean distance and other similarity measures we studied at 5% significance level. Moreover, Euclidean distance produced smaller standard deviation than other similarity measures with the same sampling proportion. This means that it can produce more stable and reliable results in practice. Using Euclidean distance reduces the risk of missing faults.

Cosine similarity produced comparable results to Jaccard Index over the most of subject programs. Also, cosine similarity generated smaller standard deviations than Jaccard Index over medium-sized programs. For small-sized programs, the standard deviation of cosine similarity was roughly equal to that of Jaccard Index. The NAPFD and the standard deviation of Jaccard Index were roughly equal to those of its variants with the same sampling proportion. Furthermore, there was no statistically significant difference between Jaccard Index and its variants. The proportional distance generated smaller NAPFD than other similarity measures with the same sampling proportion except the program *sed*. Moreover, the difference of NAPFD between the proportional distance and other similarity measures was statistically significant at 5% significance level.

6.2. Experiment 2: Evaluating the Effects of Six Similarity Measures on the Global Similarity-Based Prioritization Algorithm. Similar to Experiment 1, we also took 6 samples for every subject program so as to look deeper into the effects of different similarity measures on the globe similarity-based prioritization algorithm. The experimental results are shown in Figure 3, where the x-axis of each graph indicates the number of tests selected, while the y-axis shows the mean value of each similarity measure in NAPFD.

From Figure 3, cosine similarity performed better than or as good as Jaccard Index. For instance, cosine similarity improved more than 15% compared to Jaccard Index with respect to NAPFD over the program *gzip*. Likewise, Euclidean distance also performed consistently better than Jaccard Index for smaller samples. In most cases, the results of Jaccard Index were very close to those of its variants. The proportional distance measure showed inferior results over most of programs.

We conducted ANOVAs to verify whether different similarity measures generate the significant effects on the global similarity-based prioritization algorithm at 5% significance level. Having executed ANOVAs, we find that the p value was significantly less than 0.05 for every sample across every subject program. In other words, different measures have significant effects on the global similarity-based prioritization algorithm. We calculated the standard deviation generated by the global similarity-based prioritization algorithm. The standard deviations are shown in Table 7.

Like Experiment 1, we further conducted the nonparametric Mann-Whitney *U* test to qualitatively compare the statistical significance of differences between each pair of similarity measures. Table 8 summarizes the statistical results. From Table 8, we can draw the same conclusion as Experiment 1, that is, Euclidean distance outperformed other similarity measures based on the global similarity prioritization algorithm. In the best case, Euclidean distance outperformed the proportional distance on 42 settings. In the worse case, Euclidean distance outperformed cosine similarity on 23 settings, while cosine similarity outperformed it only on 9 settings.

By the nonparametric Mann-Whitney *U* test, we find that cosine similarity performed very well over the medium-sized programs as it has higher NAPFD than other similarity measures. Moreover, there was statistically significant difference between cosine similarity and other similarity measures except Euclidean distance in most cases. Also, cosine similarity can generate smaller standard deviation than Jaccard Index. For small-sized programs, the means of Jaccard Index were slightly higher than those of cosine similarity. But there was no statistically significant difference between Jaccard Index and cosine similarity. In other words, cosine similarity performs well similar to Jaccard Index over small-sized programs at least.

For small-sized programs, Euclidean distance performed better than Jaccard Index in terms of NAPFD. The means of Euclidean distance statistically differed from those of Jaccard Index. Moreover, Euclidean distance had smaller standard deviation than Jaccard Index. This means that Euclidean distance can generate more stable results than Jaccard Index. In a word, the application of Euclidean distance reduces the risk of missing faults in terms of the global similarity-based test case prioritization algorithm.

Although Jaccard Index and its variants emphasize different aspects, there were no statistically significant differences between them. The standard deviations of Jaccard Index were slightly smaller than those of its variants, that is, Sokal-Sneath and Gower-Legendre. By the nonparametric Mann-Whitney *U* test, we also find that proportion distance

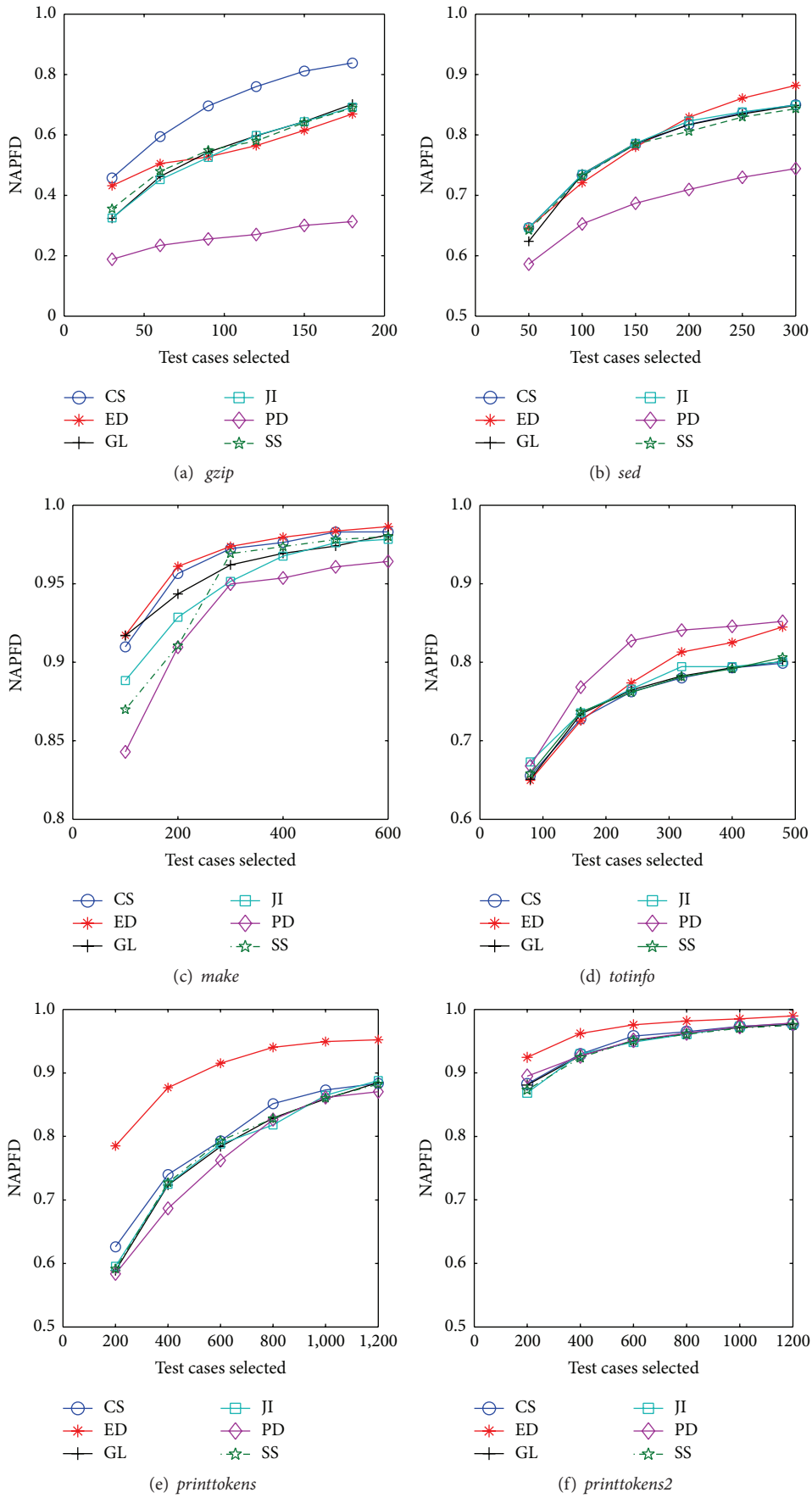


Figure 3: Continued.

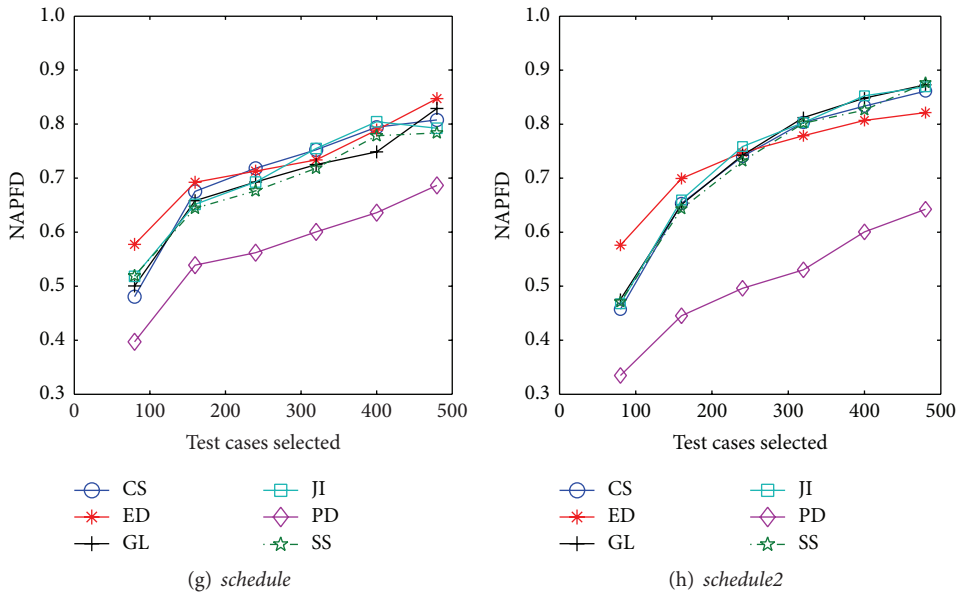


Figure 3: NAPFDs of the global similarity-based prioritization algorithm using different distance measures for every subject program.

generated lower NAPFDs than other similarity measures over all subject programs except for the program *totinfo*. The differences between the proportion distance and other similarity measures were statistically significant. Compared with other similarity measures, the proportion distance had a larger standard deviation. Although it considers more information, that is, the frequency of executed branch, than Euclidean distance, it cannot perform better.

6.3. ART-Based Prioritization Algorithm versus Global Similarity-Based Prioritization Algorithm. There are numerous ART-based techniques. The techniques used are both white-box-based regression test case prioritization [21] and black-box-based regression test case prioritization [22]. We conducted the empirical study in the context of white-box-based regression testing. In this sense, we only empirically compared our approach with the ART-based prioritization algorithm [21]. The ART-based prioritization algorithm is similar to the global similarity-based prioritization algorithm; that is, the next test case to be selected is the one that has the maximal distance to the prioritized test cases. However, the candidate sets of the two algorithms are different in the process of prioritization. The ART-based prioritization algorithm requires the construction of a candidate set that is used to save the test cases to be prioritized temporarily. The candidate set is a subset of all remaining test cases that are not yet prioritized. On the contrary, the global similarity-based prioritization algorithm does not require the construction of a candidate set. In this sense, the global similarity-based prioritization algorithm can assure the global diversity of prioritized test cases while the ART-based prioritization only keeps the local diversity.

This experiment aids in validating whether the global diversity can detect faults earlier than the local diversity. Since Euclidean distance shows good performance under the two prioritization algorithms, we only compared the two algorithms using Euclidean distance. Like Experiments 1 and 2, we conducted the nonparametric Mann-Whitney U test at 5% significance level to observe the statistically significant difference between the two prioritization algorithms using Euclidean distance. The goal of t-test is to determine which prioritization algorithm is more effective. The "win/tie/lose" of the global similarity-based prioritization versus the ART-based prioritization algorithm is (25/20/3). The result shows that the global similarity-based prioritization algorithm was significantly better than the ART-based prioritization algorithm. Among the 48 different experimental settings, the global similarity-based prioritization algorithm performed better than the ART-based prioritization on 25 settings. Nevertheless, the ART-based prioritization only outperformed the global similarity-based prioritization on 3 settings. The result suggests that the global diversity aids in detecting faults earlier than the local diversity. Also, the global similarity-based prioritization algorithm reduces the effect of randomness generated by the ART-based prioritization algorithm on regression test case prioritization.

6.4. Global Similarity-Based Prioritization and Other Comparison Baselines. The additional prioritization techniques were empirically compared with the global similarity-based prioritization algorithm using Euclidean distance, as well as random prioritization and no prioritization. We calculated the mean values of NAPFD and the standard deviations across all techniques for every subject program. Table 9 presents the experimental results of different prioritization techniques, where T_8 denotes the global similarity-based prioritization algorithm using Euclidean distance.

We conducted the nonparametric Mann-Whitney U test based on the results six prioritization techniques generated. The statistical results are shown in Table 10. From Table 10, we find that T_8 and the additional prioritization algorithms performed significantly better than random prioritization

Table 7: The standard deviations of different measures generated by the global similarity-based prioritization algorithm.

Program	Measure	Standard deviation					
		$N = 1$	$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$
gzip	JI	0.11	0.10	0.09	0.08	0.07	0.07
	CS	0.04	0.02	0.02	0.01	0.01	0.01
	ED	0.04	0.02	0.03	0.01	0.02	0.02
	GL	0.09	0.09	0.10	0.09	0.08	0.08
	PD	0.03	0.03	0.03	0.02	0.02	0.01
	SS	0.12	0.10	0.10	0.09	0.09	0.07
sed	JI	0.03	0.03	0.02	0.02	0.01	0.01
	CS	0.03	0.02	0.01	0.01	0.01	0.01
	ED	0.03	0.02	0.01	0.01	0.01	0.01
	GL	0.03	0.03	0.02	0.01	0.01	0.02
	PD	0.03	0.01	0.01	0.01	0.01	0.00
	SS	0.04	0.03	0.02	0.02	0.02	0.02
make	JI	0.15	0.08	0.08	0.04	0.03	0.03
	CS	0.06	0.03	0.02	0.02	0.01	0.01
	ED	0.03	0.02	0.01	0.00	0.00	0.00
	GL	0.08	0.09	0.05	0.05	0.03	0.02
	PD	0.01	0.00	0.00	0.00	0.00	0.00
	SS	0.18	0.13	0.04	0.03	0.03	0.02
totinfo	JI	0.06	0.03	0.03	0.03	0.01	0.02
	CS	0.06	0.03	0.02	0.02	0.02	0.02
	ED	0.06	0.05	0.05	0.04	0.03	0.03
	GL	0.05	0.04	0.03	0.02	0.02	0.02
	PD	0.03	0.03	0.02	0.03	0.02	0.02
	SS	0.05	0.03	0.03	0.02	0.03	0.02
schedule	JI	0.11	0.09	0.10	0.10	0.06	0.07
	CS	0.09	0.10	0.10	0.08	0.09	0.09
	ED	0.04	0.04	0.01	0.01	0.01	0.01
	GL	0.10	0.11	0.09	0.09	0.09	0.09
	PD	0.05	0.07	0.05	0.07	0.06	0.04
	SS	0.10	0.12	0.09	0.10	0.09	0.09
schedule2	JI	0.13	0.12	0.10	0.09	0.07	0.06
	CS	0.14	0.14	0.11	0.09	0.08	0.05
	ED	0.04	0.04	0.07	0.03	0.05	0.04
	GL	0.13	0.14	0.12	0.09	0.07	0.06
	PD	0.08	0.03	0.06	0.03	0.04	0.05
	SS	0.15	0.11	0.12	0.10	0.08	0.06
print_tokens	JI	0.10	0.09	0.08	0.08	0.06	0.05
	CS	0.10	0.10	0.08	0.06	0.05	0.05
	ED	0.02	0.02	0.01	0.00	0.00	0.00
	GL	0.13	0.09	0.07	0.06	0.06	0.05
	PD	0.03	0.02	0.01	0.00	0.00	0.00
	SS	0.10	0.09	0.07	0.06	0.06	0.05
print_tokens2	JI	0.05	0.04	0.02	0.01	0.01	0.01
	CS	0.06	0.04	0.02	0.01	0.02	0.01
	ED	0.02	0.01	0.01	0.00	0.00	0.00
	GL	0.06	0.04	0.02	0.02	0.02	0.01
	PD	0.03	0.05	0.04	0.02	0.02	0.02
	SS	0.06	0.03	0.03	0.02	0.02	0.01

Table 8: A summary of the statistical significance of differences between each pair of similarity measures over 48 different experimental settings (8 programs × 6 sampling rates based on the global similarity prioritization algorithm).

	CS	ED	GL	JI	PD
ED	(23/16/9)	—	—	—	—
GL	(1/35/12)	(9/11/28)	—	—	—
JI	(2/36/10)	(8/11/29)	(3/45/0)	—	—
PD	(5/7/36)	(4/2/42)	(5/8/35)	(6/9/33)	—
SS	(2/33/13)	(6/14/28)	(4/39/5)	(3/41/4)	(32/10/6)

and the order of test case generated. Similarly, T_8 also performed significantly better than the additional function-coverage prioritization. In general, T_8 and the additional statement coverage prioritization were equal to each other. Likewise, T_8 was comparable to the best coverage-based prioritization algorithm, that is, the additional branch coverage prioritization.

In terms of the standard deviation, T_8 was superior to other prioritization techniques. This means that it can yield more reliable results. In this sense, T_8 is very promising as a candidate for regression test case prioritization techniques. With respect to the additional coverage-based prioritization techniques, we find that the best technique is the additional branch coverage prioritization, followed by the additional statement coverage prioritization and finally the additional function-coverage prioritization. The result suggests that coverage granularity has an important impact on the additional coverage prioritization techniques.

6.5. Discussion. The main motivation for similarity-based techniques is to provide a scalable testing approach [29]. The input of the global similarity-based prioritization algorithm is a set of branches covered which represent test cases. The more the test cases in the original test suite, the larger the scale of the inputs. The input is also related to the number of branches in software under testing. The more the number of branches, the larger the average cost of calculating the distance between pair-wise test cases. In short, the scalability of our proposed approaches increases with the size of test suite and the number of branches. Additionally, the prioritized test case ordering is relevant to the topologies of similarity measures. Jaccard Index and its variants generate the close results as the three measures have similar topologies.

7. 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.

(i) Internal Validity. We empirically compared the effects of six similarity measures on two prioritization algorithms by sampling different rates of test cases. The conclusion of this study could be affected by the sampling rates. Therefore, we took 6 samples for every subject program and reported the mean value and the standard deviation of every sample. Another potential threat to internal validity lies on potential instrumentation inaccuracy. To mitigate this threat, widely used and tested open source tool `gcov` has been adopted. If there is a fault in the original programs, the validity of the collected profiles may be threatened by the fault. To avoid any bias, the subject programs were chosen from a well-managed software repository [36].

(ii) External Validity. Our results only rely on eight subject programs written in the C language including three medium-sized programs and five small-sized programs. Threats to external validity are revolved around the representativeness of the subject programs. To alleviate, we conducted experiments on the eight programs which are from different domains with different characteristics, for example, different number of faults, different line of code, and different sizes of test suites. Further empirical studies will be conducted on larger scale and more representative industrial programs in future work.

(iii) Construct Validity. As previous studies, the measure (i.e., normalized average percentage of fault detection) is adopted to evaluate the effectiveness of different prioritization techniques. This may be insufficient for evaluating the effectiveness of different combinations as different combinations have different execution costs. There may be other metrics which are more relevant to this study. Also, as previous studies, we did not distinguish the severities of different faults. The severities of faults might influence the effectiveness of different combinations.

8. Conclusion and Future Work

Regression testing is usually executed under constrained resources and time. Test case prioritization has been proposed to improve the effectiveness of regression testing. It aims to reschedule an execution order of test cases so that the faults can be detected as early as possible. Previous studies have verified that ART-based test case prioritization algorithm is very promising. However, the algorithm significantly depends on similarity measures. Moreover, the algorithm cannot assure the global diversity of test cases.

In this context, we proposed a global similarity-based test case prioritization algorithm based on the comparison of similarity measures between test cases in a given suite. We empirically evaluated the effects of different similarity measures on the effectiveness of test cases prioritization over 8 programs written in the C language. By ANOVAs, we find that different measures have significant effects on the two test case prioritization algorithms. Particularly, Euclidean distance performs better than other five similarity measures in terms of NAPFD and standard deviation. Therefore, we recommend Euclidean distance. On the contrary, the proportional distance performs worse than other similarity measures we studied. In other words, we do not recommend the proportional distance. The experimental results also show that there is no statistically significant difference between Jaccard Index and its variants.

Table 9: The mean value of NAPFD and the standard deviation. Columns "NV" and "SD" show the NAPFD and the standard deviation, respectively.

Program	TCP	$N = 1$		$N = 2$		$N = 3$		$N = 4$		$N = 5$		$N = 6$	
		NV	SD	NV	SD	NV	SD	NV	SD	NV	SD	NV	SD
gzip	T_1	0.293	0.10	0.419	0.09	0.506	0.09	0.540	0.09	0.596	0.10	0.680	0.08
	T_2	0.426	0.00	0.626	0.00	0.751	0.00	0.813	0.00	0.851	0.00	0.875	0.00
	T_5	0.581	0.05	0.674	0.04	0.701	0.04	0.735	0.04	0.764	0.04	0.787	0.04
	T_6	0.553	0.06	0.639	0.04	0.685	0.04	0.720	0.04	0.753	0.05	0.773	0.04
	T_7	0.537	0.04	0.635	0.04	0.682	0.04	0.712	0.04	0.741	0.04	0.760	0.04
	T_8	0.432	0.04	0.505	0.02	0.527	0.02	0.565	0.01	0.615	0.01	0.669	0.01
sed	T_1	0.607	0.07	0.686	0.06	0.720	0.06	0.806	0.03	0.824	0.03	0.860	0.03
	T_2	0.423	0.00	0.631	0.00	0.707	0.00	0.761	0.00	0.798	0.00	0.823	0.00
	T_5	0.646	0.05	0.697	0.04	0.755	0.04	0.803	0.04	0.846	0.03	0.859	0.04
	T_6	0.641	0.06	0.702	0.02	0.781	0.02	0.833	0.02	0.877	0.02	0.883	0.03
	T_7	0.655	0.01	0.718	0.02	0.794	0.02	0.856	0.02	0.881	0.02	0.904	0.02
	T_8	0.647	0.03	0.721	0.02	0.780	0.01	0.829	0.01	0.869	0.01	0.888	0.01
make	T_1	0.726	0.23	0.801	0.16	0.871	0.12	0.904	0.08	0.924	0.07	0.942	0.05
	T_2	0.139	0.00	0.143	0.00	0.144	0.00	0.145	0.00	0.145	0.00	0.146	0.00
	T_5	0.678	0.26	0.827	0.17	0.880	0.09	0.929	0.06	0.943	0.05	0.942	0.06
	T_6	0.920	0.03	0.980	0.00	0.993	0.00	0.996	0.00	0.997	0.00	0.998	0.00
	T_7	0.923	0.04	0.980	0.00	0.993	0.00	0.996	0.00	0.997	0.00	0.998	0.00
	T_8	0.917	0.03	0.971	0.02	0.984	0.01	0.980	0.00	0.990	0.00	0.986	0.00
totinfo	T_1	0.601	0.09	0.702	0.05	0.741	0.04	0.769	0.04	0.796	0.03	0.814	0.03
	T_2	0.758	0.00	0.827	0.00	0.855	0.00	0.870	0.00	0.878	0.00	0.884	0.00
	T_5	0.634	0.07	0.712	0.05	0.749	0.04	0.774	0.03	0.815	0.04	0.829	0.03
	T_6	0.653	0.04	0.734	0.03	0.771	0.03	0.789	0.03	0.853	0.03	0.882	0.03
	T_7	0.660	0.05	0.749	0.04	0.784	0.03	0.802	0.03	0.875	0.03	0.908	0.03
	T_8	0.649	0.06	0.725	0.05	0.774	0.05	0.813	0.04	0.825	0.03	0.845	0.03
schedule	T_1	0.444	0.10	0.593	0.10	0.670	0.12	0.707	0.10	0.732	0.10	0.747	0.09
	T_2	0.355	0.00	0.400	0.00	0.414	0.00	0.422	0.00	0.427	0.00	0.430	0.00
	T_5	0.455	0.11	0.595	0.10	0.680	0.10	0.717	0.10	0.748	0.10	0.765	0.09
	T_6	0.546	0.13	0.672	0.12	0.719	0.12	0.746	0.09	0.791	0.10	0.817	0.09
	T_7	0.582	0.12	0.689	0.12	0.733	0.11	0.782	0.11	0.841	0.10	0.882	0.08
	T_8	0.612	0.04	0.693	0.04	0.724	0.01	0.737	0.01	0.746	0.01	0.751	0.01
schedule2	T_1	0.417	0.18	0.615	0.13	0.695	0.13	0.752	0.11	0.780	0.08	0.810	0.06
	T_2	0.186	0.00	0.193	0.00	0.195	0.00	0.197	0.00	0.197	0.00	0.198	0.00
	T_5	0.429	0.17	0.587	0.15	0.721	0.12	0.776	0.09	0.797	0.09	0.822	0.07
	T_6	0.515	0.15	0.698	0.12	0.758	0.10	0.807	0.09	0.813	0.05	0.848	0.06
	T_7	0.506	0.15	0.675	0.13	0.767	0.12	0.818	0.09	0.830	0.07	0.876	0.06
	T_8	0.576	0.04	0.699	0.04	0.749	0.07	0.789	0.03	0.807	0.05	0.827	0.04
print_tokens	T_1	0.552	0.12	0.690	0.10	0.788	0.08	0.818	0.07	0.852	0.06	0.864	0.05
	T_2	0.182	0.00	0.280	0.00	0.329	0.00	0.354	0.00	0.398	0.00	0.427	0.00
	T_5	0.589	0.12	0.733	0.09	0.795	0.08	0.821	0.07	0.857	0.06	0.879	0.05
	T_6	0.716	0.08	0.803	0.07	0.840	0.07	0.867	0.06	0.888	0.05	0.906	0.06
	T_7	0.695	0.06	0.813	0.08	0.834	0.07	0.856	0.06	0.889	0.05	0.900	0.05
	T_8	0.785	0.02	0.877	0.02	0.915	0.01	0.940	0.00	0.950	0.00	0.952	0.00
print_tokens2	T_1	0.857	0.06	0.919	0.03	0.961	0.02	0.963	0.02	0.967	0.01	0.976	0.01
	T_2	0.000	0.00	0.000	0.00	0.000	0.00	0.000	0.00	0.000	0.00	0.030	0.00
	T_5	0.852	0.06	0.920	0.03	0.968	0.02	0.969	0.02	0.971	0.01	0.977	0.01
	T_6	0.926	0.03	0.965	0.02	0.986	0.01	0.987	0.01	0.991	0.01	0.992	0.01
	T_7	0.933	0.03	0.969	0.01	0.986	0.01	0.988	0.01	0.990	0.01	0.992	0.01
	T_8	0.924	0.02	0.962	0.01	0.982	0.01	0.983	0.00	0.985	0.00	0.990	0.00

Table 10: A summary of the statistical significance of differences between each pair of test case prioritization algorithms over 48 different experimental settings (8 programs × 6 sampling rates based on 6 prioritization algorithms).

	T_1	T_2	T_5	T_6	T_7
T_2	(12/1/35)	—	—	—	—
T_5	(23/24/1)	(38/0/10)	—	—	—
T_6	(48/0/0)	(37/3/8)	(40/4/4)	—	—
T_7	(48/0/0)	(37/3/8)	(41/1/6)	(17/30/1)	—
T_8	(48/0/0)	(36/3/9)	(36/6/6)	(11/27/10)	(9/21/18)

The ART-based prioritization algorithm was empirically compared with the global similarity-based prioritization algorithm. By conducting pair-wise t-test, we find that the global similarity-based prioritization algorithm outperforms the ART-based prioritization algorithm when both of them use Euclidean distance for the calculation of the distance between pair-wise test cases. The result suggests that test case diversity can aid in detecting faults as early as possible. This study is aimed at providing practical guides for picking the approximate combination of test case prioritization algorithms and similarity measures.

For future work, our study will be extended to large scale industrial projects to validate the effectiveness of our approach. Furthermore, much more similarity measures will be empirically evaluated. Last but not least, we will collect more coverage information test cases exercised and construct different structural profiles, for example, function execution sequence, function call sequence, and function call tree. The effects of different structural profiles on test case prioritization will also be empirically evaluated in regression testing.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The work was partly supported by the National Natural Science Foundation of China (Grant no. 61502497), China Postdoctoral Science Foundation (Grant no. 2015M581887), and Science and Technology Program of Xuzhou, China (Grant no. KC15SM051).

References

[1] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 2, pp. 173–210, 1997.

[2] M. J. Harrold and A. Orso, "Retesting software during development and maintenance," in *Proceedings of the Frontiers of Software Maintenance (FoSM '08)*, pp. 99–108, IEEE, Beijing, China, October 2008.

[3] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529–551, 1996.

[4] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[5] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Transactions on Software Engineering and Methodology*, vol. 2, no. 3, pp. 270–285, 1993.

[6] J. A. Jones and M. J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 195–209, 2003.

[7] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: an empirical study," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99)*, pp. 179–188, Oxford, UK, September 1999.

[8] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, "Study of effective regression testing in practice," in *Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE '97)*, pp. 264–274, November 1997.

[9] P. Tonella, P. Avesani, and A. Susi, "Using the case-based ranking methodology for test case prioritization," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pp. 123–132, Philadelphia, Pa, USA, September 2006.

[10] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing Verification & Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[11] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[12] I. Gupta, A. Leitner, M. Oriol, and B. Meyer, "Experimental assessment of random testing for object-oriented software," in *Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA '07)*, pp. 84–94, London, UK, July 2007.

[13] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos, "Time-aware test suite prioritization," in *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '06)*, pp. 1–12, 2006.

[14] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225–237, 2007.

[15] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing Verification & Reliability*, vol. 12, no. 4, pp. 219–249, 2002.

[16] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the ACM 36th International Conference on Software Engineering (ICSE '14)*, pp. 435–445, Hyderabad, India, June 2014.

[17] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov, "Comparing non-adequate test suites using coverage criteria," in *Proceedings of the 22nd International Symposium on Software Testing and Analysis (ISSTA '13)*, pp. 302–313, Lugano, Switzerland, July 2013.

[18] W. Dickinson, D. Leon, and A. Podgurski, "Finding failures by cluster analysis of execution profiles," in *Proceedings of the 23rd IEEE International Conference on Software Engineering (ICSE '01)*, pp. 339–348, Toronto, Canada, May 2001.

[19] W. Masri, A. Podgurski, and D. Leon, "An empirical study of test case filtering techniques based on exercising information flows," *IEEE Transactions on Software Engineering*, vol. 33, no. 7, pp. 454–477, 2007.

[20] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: an industrial case study," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM '11)*, pp. 382–391, IEEE, Williamsburg, VA, USA, September 2011.

[21] B. Jiang, Z. Y. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE '09)*, pp. 233–244, Auckland, New Zealand, November 2009.

[22] X. Zhang, T. Y. Chen, and L. Huai, "An application of adaptive random sequence in test case prioritization," in *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering*, pp. 126–131, 2014.

[23] R. Wang, S. Jiang, and D. Chen, "Similarity-based test case prioritization," in *Proceedings of the 27th International Conference on Software Engineering and Knowledge (SEKE '15)*, pp. 358–363, Pittsburgh, Pa, USA, July 2015.

[24] Y. Ledru, A. Petrenko, and S. Borody, "Prioritizing test cases with string distances," *Automation Software Engineering*, vol. 19, no. 1, pp. 5–95, 2012.

[25] C. Fang, Z. Chen, K. Wu, and Z. Zhao, "Similarity-based test case prioritization using ordered sequences of program entities," *Software Quality Journal*, vol. 22, no. 2, pp. 335–361, 2014.

[26] C. Zhang, Z. Y. Chen, Z. H. Zhao, S. L. Yan, J. Y. Zhang, and B. W. Xu, "An improved regression test selection technique by clustering execution profiles," in *Proceedings of the 10th International Conference on Quality Software*, pp. 171–179, IEEE, Zhangjiajie, China, July 2010.

[27] S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng, "Using semi-supervised clustering to improve regression test selection techniques," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification, and Validation (ICST '11)*, pp. 1–10, Berlin, Germany, March 2011.

[28] R. Xu and D. Wunsch II, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[29] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 1, article 6, 2013.

[30] H. Hemmati and L. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE '10)*, pp. 141–150, San Jose, Calif, USA, November 2010.

[31] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis (ISSTA '09)*, pp. 201–212, 2009.

[32] M. K. Ramanathan, M. Koyuturk, A. Grama, and S. Jagannathan, "PHALANX: a graph-theoretic framework for test case prioritization," in *Proceedings of the 23rd ACM Symposium on Applied Computing (SAC '08)*, pp. 667–673, ACM, Ceará, Brazil, March 2008.

[33] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: the ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[34] R. Wang, B. Qu, and Y. Lu, "Empirical study of the effects of different profiles on regression test case reduction," *IET Software*, vol. 9, no. 2, pp. 29–38, 2015.

[35] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, Elsevier, 3rd edition, 2012.

[36] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[37] G. Rothermel, R. H. Untcn, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[38] A. M. Smith, J. Geiger, G. M. Kapfhammer, and M. L. Soffa, "Test suite reduction and prioritization with call trees," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*, pp. 539–540, Atlanta, Ga, USA, November 2007.

[39] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: a study of test case generation and prioritization," in *Proceedings of theIEEE International Conference on Software Maintenance*, pp. 255–264, IEEE, Paris, France, October 2007.

[40] H. Scheffe, *The Analysis of Variance*, John Wiley & Sons, New York, NY, USA, 3rd edition, 1993.

[41] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[42] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*, pp. 1–10, Honolulu, Hawaii, USA, May 2011.