SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2017, Vol. 60 092112:1–092112:15
doi: 10.1007/s11432-016-9057-8

Cost-effective testing based fault localization with distance based test-suite reduction

Xingya WANG[1], Shujuan JIANG[1*], Pengfei GAO[1], Xiaolin JU[2,3], Rongcun WANG[1] & Yanmei ZHANG[1,3]

[1]*School of Computer Science and Technology, China University of Mining and Technology, Xuzhou* 221116*, China;*
[2]*School of Computer Science and Technology, Nantong University, Nantong* 226019*, China;*
[3]*Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin* 541004*, China*

Received December 30, 2016; accepted March 29, 2017; published online July 28, 2017

Abstract The aim of testing based fault localization (TBFL) involves improving the efficiency of program debugging by providing developers with a guide of ranked list of suspicious statements. However, collection of testing information of the whole original test-suite is excessively expensive or even infeasible for developers to conduct TBFL. Traditional test-suite reduction (TSR) techniques are utilized to reduce the size of test-suite. However, they entail a time-consuming process of whole testing information collection. In this study, the distance based test-suite reduction (DTSR) technique is proposed. As opposed to the whole testing information, the distances among the test cases are used to guide the process of test-suite reduction in DTSR. Hence, it is only necessary to collect the testing information for a portion of the test cases for TSR and TBFL. The investigation on the Siemens and SIR benchmarks reveals that DTSR can effectively reduce the size of the given test-suite as well as the time cost of TBFL. Additionally, the fault locating effectiveness of DTSR results is close to that when the whole test-suite is used.

Keywords program debugging, fault localization, test-suite reduction, distance estimation, category partition

Citation Wang X Y, Jiang S J, Gao P F, et al. Cost-effective testing based fault localization with distance based test-suite reduction. Sci China Inf Sci, 2017, 60(9): 092112, doi: 10.1007/s11432-016-9057-8

1 Introduction

Debugging is a tedious and time-consuming albeit essential activity during software development. Researchers have proposed several types of approaches to improve the process of fault localization [1] to reduce the effort for manual debugging. Among these approaches, testing based fault localization (TBFL) is promising [2]. It relies on testing information (e.g., code coverage and execution result) obtained from executing the program to automatically identify the most likely faulty parts, and thereby narrows search areas.

Thus, TBFL was demonstrated as an effective method in several studies. However, several issues may affect its performance. A very serious problem corresponds to the time-consuming process of collecting coverage information. The performance of thousands of test cases on industrial software can involve several hours or even days [3]. Therefore, collection of coverage information of all test cases is a slow

* Corresponding author (email: shjjiang@cumt.edu.cn)

process that may not always be practically feasible. Another problem corresponds to the tedious process of executing result inspections [4]. The execution of results is essential when a TBFL technique is applied to statistically locate faulty parts. However, it is not easy to assert the executing result of an execution when a failure is not as obvious as a program crash or invalid output formats [5]. Typically, it requires manual inspection. Therefore, significant manual effort is expended in determining the results of all executions, and this decreases the efficiency of TBFL.

A valid method to address these problems involves reducing test cases wherein tracing and inspection is required. Extant studies proposed a few test-suite reduction (TSR) techniques to reduce the number of test cases. These techniques always share a common framework. First, a few characters (e.g., high-coverage [6], high-partition [4]) that can maintain the effectiveness of TBFL with the original test-suite are summarized, and this is followed by generating a series of test requirements based on these characters. Next, a heuristic algorithm (e.g., greedy algorithm) is used to output a reduced test-suite that can satisfy all test requirements. Finally, it collects coverage information and executing results of the reduced test-suite and provides the same to a TBFL for fault localization. The experimental results indicated that these techniques effectively reduced the size of the original test-suite while not exerting an obvious negative effect on TBFL. However, it is not possible to perform the characters summarization (such as determining the code coverage of the original test-suite) and the following heuristic based test-suite reduction unless all test executions were inspected. Thus, it is not possible for these techniques to decrease the size of test cases wherein tracing and inspection is required. Therefore, TBFL continues to suffer from the low efficiency on coverage information collection, and it is difficult to reduce the time cost of TBFL.

In this study, a distance based test-suite reduction (DTSR) approach was proposed to reduce the time effort required for TBFL. The DTSR selected a subset of test cases that include maximum average pair-wise distance as the output of TSR. It relied on the intuition that test diversity maintains the characters of the original test-suite and that the distances among the test cases are reasonable to measure the test diversity. With respect to the DTSR, only a small portion of test cases were needed to trace and inspect, and the developers could only use the information associated with test executions for the reduced test-suite and the failure-causing test case to locate the fault. Thus, the proposed approach effectively reduced the time cost of testing information collection. The proposed approach was empirically compared to coverage based test-suite reduction (CTSR) [4,6] and random based test-suite reduction (RTSR) on the reduction rate Reduction, the fault localization effect Expense, and the time cost TimeCost. Experimental results indicated that the proposed approach effectively reduced the size of original test-suite as well as the time cost of TBFL while the reduced result did not affect the effectiveness of TBFL.

The rest of this study is organized as follows. Section 2 overviews the corresponding TSR techniques, TBFL techniques, and the test requirements for TBFL. The test-suite reduction approach is described in detail in Section 3. Experimental design and results analysis are presented in Section 4. The threats to validity and related studies are discussed in Section 5. Finally, Section 6 concludes the article and outlines directions for future research.

2 Preliminaries

This section presents an overview of TSR techniques as well as TBFL techniques, and summarizes three test requirements that are applicable to TBFL.

2.1 Test-suite reduction

The aim of TSR [7] involves selecting a subset of original test-suite that maintain a few features similar to the whole test-suite. These features are called test requirements. A set of test requirements is assumed, and the TSR problem can be stated as follows

Given: a set of test requirements $\mathrm{RS} = \{r_i \mid i \in [1, n]\}$ and a test-suite TS with subsets $\mathrm{TS}_{\mathrm{subs}} = \{\mathrm{TS}_i \mid i \in [1, n]\}$, the value of $\mathrm{isMeet}(r_i,\ t)$ corresponds to 1 where $t \in \mathrm{TS}_i$. Function $\mathrm{isMeet}(r,\ t)$ is used to test whether r is satisfied by t. It corresponds to 1 if t satisfies r and otherwise corresponds to 0.

Table 1 Notations widely used in suspiciousness calculation

Notation	Value	Description
n_p	$\sum_{j=1}^{n}(1-r_j)$	The number of passed test cases
n_f	$\sum_{j=1}^{n} r_j$	The number of failed test cases
$n_{cp}(s_i)$	$\sum_{j=1}^{n} c_{i,j}\times(1-r_j)$	The number of passed test cases that can cover the statement s_i
$n_{cf}(s_i)$	$\sum_{j=1}^{n} c_{i,j}\times r_j$	The number of failed test cases that can cover the statement s_i
$n_{up}(s_i)$	$\sum_{j=1}^{n}(1-c_{i,j})\times(1-r_j)$	The number of passed test cases that cannot cover the statement s_i
$n_{uf}(s_i)$	$\sum_{j=1}^{n}(1-c_{i,j})\times r_j$	The number of failed test cases that cannot cover the statement s_i

Problem: determining a representative set of test cases from TS such that executing the reduced test-suite $\mathrm{TS}_{\mathrm{reduced}}$ satisfies all test requirements in RS.

It is necessary to satisfy all test requirements, and thus $\mathrm{TS}_{\mathrm{reduced}}$ must contain at least one test case from each TS_i. It is assumed that one test case may satisfy more than one test requirement, and thus it is intractable to determine a minimum test-suite that satisfies all test requirements. Therefore, TSR must approximate the minimum cardinality [7].

2.2 Testing based fault localization

The intuition behind TBFL is that the statements that are primarily executed by passed test cases are less likely to be faulty when compared with those primarily executed by failed test cases [8]. With respect to each statement, a suspiciousness is assigned to indicate the likelihood that the statement contains a fault. An increase in the suspiciousness increases the likelihood of the statement to contain a fault. Subsequently, the statements with the highest rank are first examined while searching for the fault. If the fault is not found after examining these statements, then developers examine the remaining statements in order of decreasing rank.

It is assumed that a program PG consists of m executable statements and a test-suite TS containing n test cases. The instrumented program is traced to gather coverage information and executing results. A matrix $M_{\mathrm{cover}} = \{c_{i,j} \mid i \in [1,m],\ j \in [1,n]\}$ is used to represent the coverage of m statements collected after running n test cases. If a statement s_i is executed by test case t_j, then $c_{i,j}$ corresponds to 1, and otherwise corresponds to 0. A vector $V_{\mathrm{result}} = \{r_j \mid j \in [1,n]\}$ is used to represent the executing results. If the test case t_j leads to a failed execution, then r_j corresponds to 1, otherwise it corresponds to 0. The TBFL calculates the suspiciousness of each statement based on M_{cover} and V_{result}.

$$\mathrm{Tarantula}(s) = \frac{n_{cf}(s)/n_f}{n_{cf}(s)/n_f + n_{cp}(s)/n_p}, \tag{1}$$

$$\mathrm{HSSF}(s) = \frac{n_{cf}^2(s)}{n} - \frac{n_{cf}(s)\times n_{cp}(s)}{n^2}. \tag{2}$$

Risk evaluation formula is used to compute statement suspiciousness and is the key to TBFL. Eqs. (1) and (2) present two typical TBFL formulas Tarantula [8] and HSSF [9], respectively, which are used in the experiment. Specifically, Tarantula is a representative TBFL approach that is widely used as a comparative technique, and HSSF was theoretically proved as one of the most effective TBFL techniques [1,2,8,9]. Table 1 explains the notations used in Eqs. (1) and (2). For each notation, its value (column 2) and a brief description (column 3) are presented.

2.3 Test requirements for testing based fault localization

Intuitively, an increase in the testing information increases the effect of TBFL [10]. A reduction in the size of test-suite could reduce the testing information and affect the effectiveness of TBFL. Therefore, random based test-suite reduction appears unreliable, and it is necessary for the reduced test-suite to satisfy certain critical test requirements to mitigate the negative impact of less testing information. In this section, three test requirements that are helpful in TBFL are summarized.

2.3.1 *Result requirement*

In TBFL, the risk evaluation formulas are usually constructed by comparing the differences between the failed executions and the passed executions [1]. If the reduced test-suite contains only failed (or passed) test cases, then all statements may be assigned the same suspiciousness. As a result, the ranked list that consists of same suspiciousness statements cannot provide a meaningful guide for fault localization. Therefore, result requirement (RR) must be satisfied to guarantee that the reduced test-suite contains at least one failed test case and one passed test case.

2.3.2 *Coverage requirement*

Coverage requirement (CR) is an often-used TSR requirement in areas of regression testing [11, 12] and fault localization [6, 13]. The aim involves generating a reduced test-suite that covers the same set of statements as the original test-suite. Additionally, cov(TS) is used to represent the number of covered statements under the test-suite TS. An empirical study by Jiang et al. indicated that the reduced test-suite that satisfies CR is more helpful in TBFL [13] when compared with the sampled cases. Therefore, it is necessary to satisfy the coverage requirement to guarantee that the reduced result $\mathrm{TS}_{\mathrm{reduced}}$ covers the same statements as the original test-suite TS, that is, $\mathrm{cov}(\mathrm{TS}_{\mathrm{reduced}}) = \mathrm{cov}(\mathrm{TS})$.

2.3.3 *Partition requirement*

Definition 1 (T-undistinguishable [4])**.** Given a program PG and a test-suite TS, two statements of PG, namely s_1 and s_2 are T-undistinguishable only if each test case in TS: executes both s_1 and s_2, or executes neither s_1 nor s_2.

Evidently, T-undistinguishable is reflexive, symmetric, and transitive. Based on classical set theory, statements indicating a T-undistinguishable relationship belong to one equivalence class [4]. According to the definition 1, statements belonging to the same equivalence class exhibit undistinguishable coverage features. The suspiciousness of each statement is estimated based on its coverage feature, and thus statements in a equivalence class are assigned the same suspiciousness. This may involve the wastage of additional time to break the tie. Furthermore, par(TS) is used to represent the number of equivalence classes under the test-suite TS. In the simplest circumstance where TS consists of only one test case t, the following two equivalence classes are obtained, namely those covered by t and those that are not covered. Thus, par(TS) corresponds to 2. An increase in the value of par(TS) increases the power of TS to distinguish statements [4]. Thus, developers can reduce their efforts with respect to tie breaking. Therefore, it is necessary to satisfy partition requirement (PR) to guarantee that $\mathrm{TS}_{\mathrm{reduced}}$ has the same number of equivalence classes as TS, that is, $\mathrm{par}(\mathrm{TS}_{\mathrm{reduced}}) = \mathrm{par}(\mathrm{TS})$.

3 Our approach

This section overviews the proposed DTSR approach and details two important steps, namely distance estimation and stopping criteria designation.

3.1 Overview

Figure 1 presents the framework of DTSR. As previously mentioned, the basis of DTSR corresponds to maximizing the average pair-wise distance among the test cases in the reduced test-suite. Thus, it requires a measure of distance for pairs of test cases and an optimization algorithm to minimize the output set of test cases with respect to that measure. Therefore, DTSR works with the following two main phases: (1) Distance matrix generation. Each matrix cell represents the distance value for pair-wise test cases evaluated by a certain distance function. (2) Test-suite reduction. Selecting a subset of the original test-suite that is most likely to satisfy all the test requirements. Generally, the problem corresponds to an NP-hard problem [7]. Therefore, a greedy algorithm is adopted to determine an optimal solution.

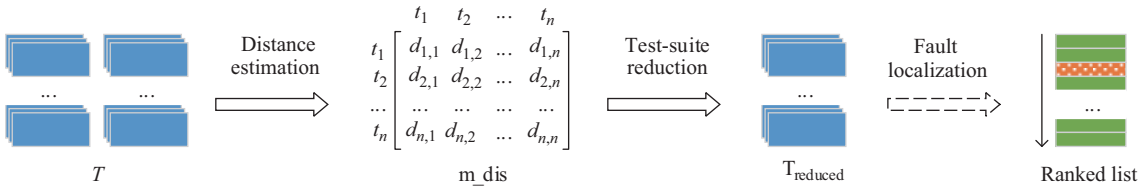

Figure 1 (Color online) Framework of distance based test-suite reduction.

Algorithm 1 outlines the details of the proposed approach (DTSR). It treats the program PG, the original test-suite TS, the first failure detected test case t_{f1}, a distance function DisFunc(), and a stopping criterion stop_cri as inputs. It finally outputs the reduced test-suite $\mathrm{TS}_{\mathrm{reduced}}$.

Algorithm 1 Distance based test-suite reduction

Input: PG, TS, t_{f1}, DisFunc(), stop_cri;
Output: $\mathrm{TS}_{\mathrm{reduced}}$.

1: // **Phase 1: generate the distance matrix** m_dis**.**
2: **for** $i = 1$ to n **do**
3: **for** $j = i$ to n **do**
4: $\mathrm{m\_dis}_{i,j} = \mathrm{m\_dis}_{j,i} = \mathrm{DisFunc}(t_i,\ t_j)$;
5: **end for**
6: **end for**
7: // **Phase 2: reduce the original test-suite TS with a greedy algorithm.**
8: $\mathrm{TS}_{\mathrm{reduced}} = \{t_{f1}\}$, $\mathrm{TS}_{\mathrm{unreduced}} = \mathrm{TS} - \{t_{f1}\}$;
9: **while** $\mathrm{TS}_{\mathrm{educed}}$ cannot satisfy stop_cri **do**
10: $\mathrm{dis}_{\max} = \mathrm{MIN\_VALUE}$, $t_{\mathrm{selected}} = \mathrm{NULL}$;
11: **for all** $t_u \in \mathrm{TS}_{\mathrm{unreduced}}$ **do**
12: $\mathrm{dis}_{\min} = \mathrm{MAX\_VALUE}$;
13: **for all** $t_r \in \mathrm{TS}_{\mathrm{reduced}}$ **do**
14: **if** $\mathrm{dis}_{\min} > \mathrm{m\_dis}_{r,u}$ **then**
15: $\mathrm{dis}_{\min} = \mathrm{m\_dis}_{r,u}$;
16: **end if**
17: **end for**
18: **if** $\mathrm{dis}_{\max} < \mathrm{dis}_{\min}$ **then**
19: $\mathrm{dis}_{\max} = \mathrm{dis}_{\min},\ t_{\mathrm{selected}} = t_u$;
20: **end if**
21: **end for**
22: $\mathrm{TS}_{\mathrm{reduced}} = \mathrm{TS}_{\mathrm{reduced}} \cup t_{\mathrm{selected}},\ \mathrm{TS}_{\mathrm{reduced}} = \mathrm{TS}_{\mathrm{reduced}} - t_{\mathrm{selected}}$;
23: run PG with t_{selected} and collect the testing information;
24: **end while**
25: output $\mathrm{TS}_{\mathrm{reduced}}$.

First, DTSR (Phase 1: lines 1–6) computes the distance values for pairs of test cases and uses an $n \times n$ matrix m_dis to record the distance values. The distance between test case t_i and t_j is equal to the distance between t_j and t_i, and thus it is only necessary for DTSR to compute one of the fore-mentioned values. Next (Phase 2: lines 7–25), based on m_dis and the given stop_cri, the DTSR resorts to a greedy algorithm to select a subset of test cases with maximum average pairwise distances. In each iteration (lines 10–23), the test case t_{selected} with the maximum distance with respect to the selected test cases $\mathrm{TS}_{\mathrm{reduced}}$ is chosen, and the testing information including the coverage information and the executing result is collected. A study by Jiang et al. [13] is followed, and the minimum distance between a test case and any test cases in a test-suite is used as the distance between a test case and a test-suite (lines 12–17). Test-suite reduction stops when the selected ones satisfy stop_cri.

3.2 Distance estimation

This section presents two types of approaches for distance estimation.

3.2.1 *String distance based distance estimation*

A test case can be assumed as a string as well as a vector of characters in a multidimensional space. A numerical encoding approach is used to transfer it into a vector of numbers. This is followed by calculating the distances between the test cases by using a certain distance metric. Several numerical ways are utilized to encode these characters. However, for the purposes of scalability and simplicity, ASCII is employed to encode these characters.

$$\mathrm{dis}_{\mathrm{Hamming}}(\mathrm{vec}_1,\ \mathrm{vec}_2) = \sum_{i=1}^{n} \mathrm{isEqual}(\mathrm{vec}_1[i],\ \mathrm{vec}_2[i]), \tag{3}$$

$$\mathrm{dis}_{\mathrm{Cartesian}}(\mathrm{vec}_1,\ \mathrm{vec}_2) = \sqrt{\sum_{i=1}^{n} (\mathrm{vec}_1[i] - \mathrm{vec}_2[i])^2}, \tag{4}$$

$$\mathrm{dis}_{\mathrm{Manhattan}}(\mathrm{vec}_1,\ \mathrm{vec}_2) = \sum_{i=1}^{n} |\mathrm{vec}_1[i] - \mathrm{vec}_2[i]|. \tag{5}$$

In this study, four distance metrics are selected, namely Hamming [14], Cartesian, Manhattan, and Levenshtein [15], to estimate the distances among the test cases. Specifically, Hamming is used to estimate the bit difference between two strings, and Cartesian and Manhattan indicate the linear distance and the sum of the wheelbase of two points in the standard coordinate system, respectively. Eqs. (3)–(5) present the formal definitions of Hamming, Cartesian, and Manhattan, respectively. Function $\mathrm{isEqual}(c_1,\ c_2)$ is used to test whether c_1 is equal to c_2. If c_1 is equal to c_2, the value of $\mathrm{isEqual}(c_1,\ c_2)$ is 0, otherwise 1.

Hamming, Cartesian and Manhattan are originally proposed to compare two strings of identical length. Thus, the strings of test cases should possess the same length. In this study, extant studies by Ledru et al. [3] are followed to complete the shorter string by a suffix that consists of character NUL in which the ASCII value corresponds to 0. For example, given two strings, str_1 = "abcd" and str_2 = "aac", the shorter string str_2 is first completed with NUL, and then ASCII encoding is performed on the two strings. Finally, the two strings are translated into two numeric vectors, namely vec_1 = (97, 98, 99, 100) and vec_2 = (97, 97, 99, 0), which are suitable for distance estimation. The distances between vec_1 and vec_2 are as follows: $\mathrm{dis}_{\mathrm{Hamming}}(\mathrm{vec}_1,\ \mathrm{vec}_2) = 0 + 1 + 0 + 1 = 2$, $\mathrm{dis}_{\mathrm{Cartesian}}(\mathrm{vec}_1,\ \mathrm{vec}_2) = \sqrt{(97-97)^2 + (98-97)^2 + (99-99)^2 + (100-0)^2} = 100.0049$ and $\mathrm{dis}_{\mathrm{Manhattan}}(\mathrm{vec}_1,\ \mathrm{vec}_2) = |97-97| + |98-97| + |99-99| + |100-0| = 101$.

Levenshtein indicates the minimum number of character edit operations (i.e., substitution, deletion, and insertion) necessary to transform one string into another. This is different from Hamming, Cartesian, and Manhattan because Levenshtein can be applied to strings of arbitrary lengths. For example, the Levenshtein distance between str_1 and str_2 corresponds to 2 since the following two operations change str_1 into str_2, and this cannot be performed with fewer than two operations.

"bcde" ⇒ "bbde"(substitution of "b" for "c"),

"bbde" ⇒ "bbd"(deletion "e" at the end).

3.2.2 *CPM based distance estimation*

Category partition method (CPM) is a black-box (or specification) based test-generation technique that generates test cases without using the knowledge of the internal structure of the program [16]. The use of CPM creates test cases by refining the functional specification into a test specification. The test specification defines categories $C = \{C_i \mid i \in [1, k]\}$ as major properties that affect the behavior of the functional unit under test. Each category is partitioned into subsets known as choices, which are defined as "all the different kinds of values that are possible for the category" [17]. Each choice within a category corresponds to a set of similar values that is assumed with respect to the type of information in the category [18]. The created test cases should cover all possible choices to provide a sufficient verification for the program to be analyzed.

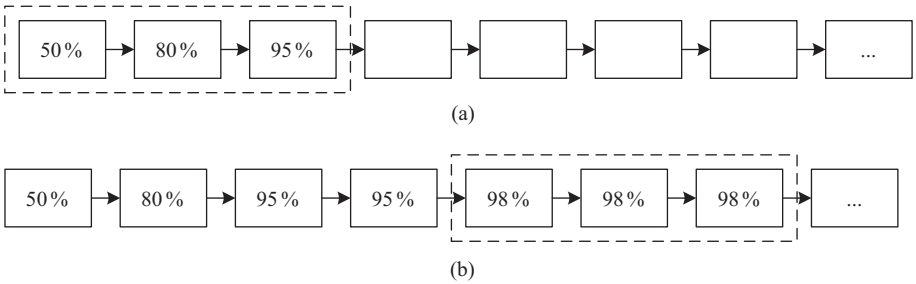

Figure 2 Sliding a 3-width window to a list of coverage rates. (a) Contents are different; (b) contents are similar.

In this study, CPM was applied to analyze the functional differences among the test cases as opposed to generating new test cases. The use of CPM translates a test case into a vector of choices, where each cell represents a functional unit that the test case is expected to test. Subsequently, the functional distances were estimated by comparing the choice vectors. It is difficult to numerical encode a choice, and thus Hamming was used to calculate the distance between two test cases. For example, given two test cases, namely t_1 and t_2, where t_1 is used to test the program under choices $C_{1.1}$, $C_{2.1}$, and $C_{3.1}$, and t_2 is used to test the program under choices $C_{1.1}$ and $C_{3.2}$. First, NUL is used to complete the shorter test case t_2 to ensure that the two test cases possess the same length. Subsequently, CPM is used to translate t_1 and t_2 into two identical vectors, namely $\text{vec}_1 = (C_{1.1}, C_{2.1}, C_{3.1})$ and $\text{vec}_2 = (C_{1.1}, \text{NUL}, C_{3.2})$. Hence, the distance between vec_1 and vec_2 is as follows: $\text{dis}_{\text{CPM}}(\text{vec}_1, \text{vec}_2) = \text{dis}_{\text{Hamming}}(\text{vec}_1, \text{vec}_2) = 0+1+1 = 2$.

3.3 Stopping criteria designation

The stopping criterion should be designed to satisfy or approximately satisfy all TBFL test requirements.

Specifically, RR denotes that $\text{TS}_{\text{reduced}}$ must contain both the failed test cases and the passed ones, i.e., $\text{TS}_{\text{reduced}} \cap \text{TS}_{\text{failed}} \neq \emptyset$ and $\text{TS}_{\text{reduced}} \cap \text{TS}_{\text{passed}} \neq \emptyset$. When compared with $\text{TS}_{\text{passed}}$, $\text{TS}_{\text{failed}}$ is always a small part of the original test-suite, and thus it is easily ignored in the process of test case selection [4]. In order to solve this problem, $\text{TS}_{\text{reduced}}$ is initialized with the failure detected test case. Thus, $\text{TS}_{\text{reduced}}$ contains at least one failed test case.

Additionally, CR denotes that $\text{TS}_{\text{reduced}}$ should cover the same statements with the original test-suite TS, and PR denotes that $\text{TS}_{\text{reduced}}$ should have the same equivalence classes with TS. The runtime information of the whole test cases is unknown, and thus the coverage result, as well as the partition result, of TS are also unknown. Thus, it cannot directly determine whether $\text{TS}_{\text{reduced}}$ satisfies CR or PR.

In this study, the sliding window technique was used to design the stopping criterion while facing CR. This is because selecting another test case will not change the coverage result if a set of selected test cases covers the same statements with the original test-suite. The sliding window technique works through the following two steps: (1) Coverage list generation in which each of the unchecked test cases is selected in turn during the test-suite reduction, and the coverage rate of each new test-suite is calculated and recorded. Thus, a list of coverage rates L_{cov} is obtained. (2) Stopping criterion recognition in which a window win is slid over L_{cov} to verify whether the window contents are similar. If the contents of win are different, then it is considered that $\text{TS}_{\text{reduced}}$ is less likely to satisfy CR. Conversely, if the contents of win are similar, then $\text{TS}_{\text{reduced}}$ is more likely to satisfy CR, and the test-suite reduction should be stopped. It should be noted that an increase in the width of the window increases the satisfaction of CR.

The code coverage list L_{cov} shown in Figure 2 is used to illustrate the working of the sliding window. A 3-width window win_3 is assumed. As shown in Figure 2(a), all nodes (coverage rates) in win_3 are different from each other. Thus, it is assumed that $\text{TS}_{\text{reduced}}$ is unable to satisfy CR. Conversely, as shown in Figure 2(b), all nodes in win_3 have the same code coverage. Thus, there is a high probability that $\text{TS}_{\text{reduced}}$ satisfies CR. In this case, it is unnecessary to select and run a next test case, and test-suite reduction should stop.

Similarly, when the partitioned result of the selected test cases is similar to those of the original test-suite, the partition result will not change when another test case is selected. Therefore, the sliding window

Table 2 Experimental subjects

Program	Description	LOC	#Fault	#Test	#Category	#Choice
JTcas	Altitude separation	181	41	1608	12	40
Tot_info	Information measure	283	23	1079	8	24
Schedule1	Priority scheduler	290	9	1200	7	24
Schedule2	Priority scheduler	317	10	1200	10	34
Print_tokens1	Lexical analyzer	478	7	1200	–	–
Print_tokens2	Lexical analyzer	410	10	1200	–	–
NanoXML	XML parser	7160	32	237	–	–
Siena	Notification middleware	6098	10	567	–	–

technique can also be used to design the stopping criterion while facing PR.

4 Empirical study

This section aims to answer the following research questions.

RQ1: What is the width that should be set to the window while designing the stopping criterion?

RQ2: Can the DTSR technique select a small subset from the original test-suite?

RQ3: Can the DTSR results maintain the effectiveness of TBFL?

RQ4: Can the DTSR technique reduce the time cost of TBFL?

4.1 Setup

4.1.1 *Subjects*

Table 2 summarizes the characteristics of Java programs used in the experiments. For each subject, its name (column 1), a brief description (column 2), the number of code lines (column 3), faulty versions (column 4), test cases (column 5), categories (column 6), and choices (column 7) are described. The former six subjects correspond to Java versions [19] of Siemens that are translated from C versions, and the rest are provided by SIR [20]. Each program consists of a base version, and dozens of faulty versions in which faults were manually seeded. This resulted in 142 faulty versions. However, the given test-suite did not detect any fault in the case of 13 faulty versions. Therefore, they were discarded since TBFL requires failure-causing test cases. A total of 129 faulty versions were used in the experiment.

All subjects provided no information about the categories and the choices and also failed to provide any test specifications. Hence, it was necessary to manually identify the categories and choices. The functional units of Print_token1 and Print_token2 were not clear, and the internal structures of NanoXML and Siena were quite complex. Thus, the CPM was performed by only analyzing the other four subjects.

4.1.2 *Variables*

For the purposes of comparison, CTSR techniques [4,6] and RTSR techniques were applied to the subjects. The CTSR used the greedy algorithm to continue selecting a test case from the unselected test-suite at each iteration until the selected cases satisfied the given test requirements. Specifically, cov denotes the CTSR technique that satisfied both RR and CR, and par denotes the CTSR technique that satisfied both RR and PR. The RTSR randomly selected a test case from the unselected test-suite and traced it at each iteration until the selected ones satisfied the given test requirements. Similarly, the RTSR technique that satisfied both RR and CR is denoted by rand_{cov}, and the RTSR technique that satisfied both RR and PR is denoted by rand_{par}. The TSR results were also applied to two typical TBFL techniques, namely Tarantula [8] and HSSF [9], to evaluate the effectiveness of TBFL. To summarize, the empirical study used nine TSR techniques and two TBFL techniques, thereby resulting in 18 pairs.

$$\text{Reduction} = \text{size}_{T_{\text{selected}}}/\text{size}_T \times 100\%, \tag{6}$$

$$\text{Expense} = \text{num}_{\text{examined}}/\text{num}_{\text{stmts}} \times 100\%, \tag{7}$$

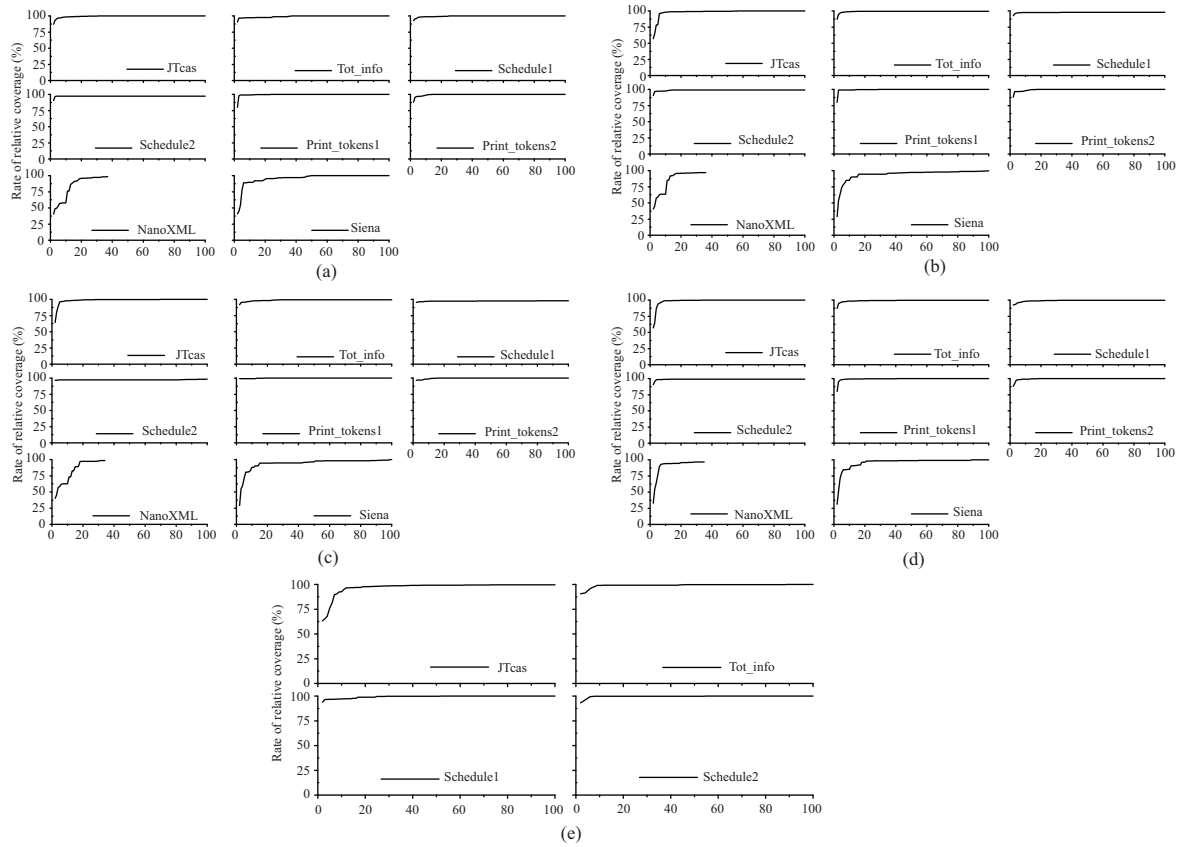

Figure 3 $\text{Relative}_{\text{cov}}$ comparison on all subjects when using different distance measures. (a) Hamming; (b) Cartesian; (c) Manhattan; (d) Levenshtein; (e) CPM.

$$\text{TimeCost} = \text{TimeCost}_{\text{reduction}} + \text{TimeCost}_{\text{collection}} + \text{TimeCost}_{\text{computation}}. \tag{8}$$

For each pair, the following three dependent variables were measured: Reduction, Expense, and TimeCost. Eqs. (6)–(8) present these evaluation metrics. Specifically, Reduction indicates the size of test cases that should be executed and inspected, and this is measured by calculating the ratio of the reduced test-suite $\text{size}_{\text{TS}_{\text{reduced}}}$ to the original test-suite size_{TS}. Furthermore, Expense represents the effectiveness of a TBFL technique that is measured by the percentage of the statements that should be examined to determine the fault. Finally, TimeCost represents time required for a process of fault locating that consists of the time of three main steps in fault localization, namely test-suite reduction, coverage information collection, and suspiciousness computation.

A test-suite and the first failure-detected test case t_{f1} were considered as the input with the aim of locating the fault detected by t_{f1}. Additionally, t_{f1} is unpredictable in practice, and thus the results of TSR as well as TBFL are not unique. In order to reduce the coincidence of t_{f1} on TSR and TBFL, each of the failed test cases of a faulty version ver_f was considered as t_{f1}, and the average results were used as the final results for ver_f.

4.2 Results and analysis

First, an aim involves investigating a width that fits the design of the stopping criterion. Figures 3 and 4 present the results of $\text{Relative}_{\text{cov}}$ and $\text{Relative}_{\text{par}}$, respectively, by comparing all subjects while using different distance measures. The horizontal axis of each sub-figure represents the width of the sliding window in the range of 2 to 100, and the vertical axis denotes $\text{Relative}_{\text{cov}}$ that represents the ratio between $\text{cov}(\text{TS}_{\text{reduced}})$ and $\text{cov}(\text{TS})$, or $\text{Relative}_{\text{par}}$ that represents the ratio between $\text{par}(\text{TS}_{\text{reduced}})$ and $\text{par}(\text{TS})$. From Figure 3, it is observed that a short width window provides a high $\text{Relative}_{\text{cov}}$ for Siemens with each distance metric. However, it is not high for SIRs. For example, $\text{Relative}_{\text{cov}}$ of NanoXML only

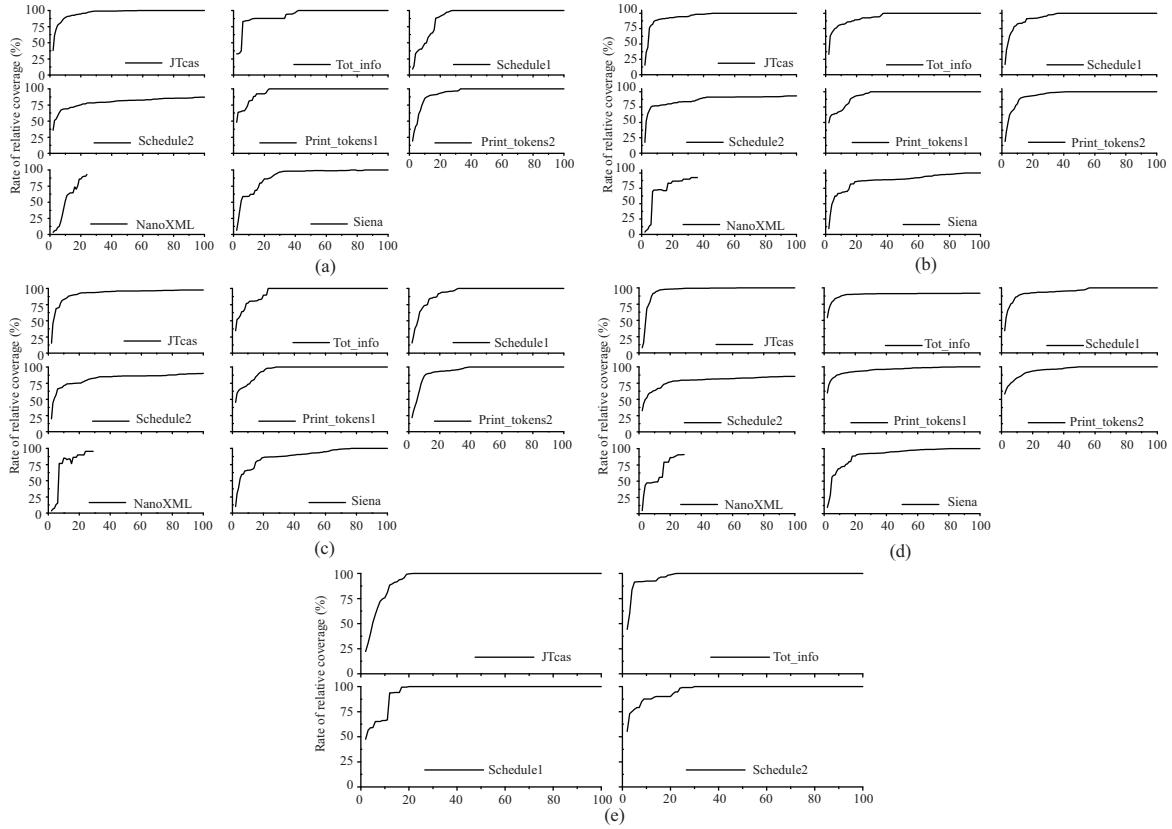

Figure 4 $\text{Relative}_{\text{par}}$ comparison on all subjects when using different distance measures. (a) Hamming; (b) Cartesian; (c) Manhattan; (d) Levenshtein; (e) CPM.

corresponds to 49.40% while using a 5-width window in Hamming. Thus, it is necessary to determine a longer width. The results indicate that a 20-width window provides a high $\text{Relative}_{\text{cov}}$ for all subjects. In this case, the $\text{Relative}_{\text{cov}}$ of all subjects with each distance measure exceed 94% and increasing the window width does not significantly increase $\text{Relative}_{\text{cov}}$. From Figure 4, it is observed that a short width window cannot provide a high $\text{Relative}_{\text{par}}$ for both Siemens and SIRs with each distance metric. Similarly, the results indicate that a 20-width window provides a high $\text{Relative}_{\text{par}}$ for all subjects. In this case, the $\text{Relative}_{\text{par}}$ of all subjects with each distance measure exceeds 85% with the exception of Schedule2 in which $\text{Relative}_{\text{par}}$ exceeds 75% and increasing the window width does not significantly increase $\text{Relative}_{\text{par}}$. Therefore, it is concluded that a 20-width window is fit for designing the stopping criterion.

Table 3 presents the Reduction results of the proposed DTSR techniques (columns 6–10) and the compared CTSR techniques (columns 2–3) and RTSR techniques (columns 4–5) while using a 20-width window. Rows 10–12 show the average Reduction of the four former subjects, namely, the ave_4, the Siemens $\text{ave}_{\text{Siemens}}$, and the SIR subjects ave_{SIR}. They are distinguished because CPM is only used for the former four subjects, and Siemens and SIRs belong to different scales [9]. This indicates that in most cases, CTSR techniques correspond to the minimum Reduction, and RTSR techniques correspond to the maximum Reduction. The DTSR falls in between CTSR and RTSR. An exception corresponds to CTSR and DTSR when they are applied to NanoXML. A small percentage of test cases cannot satisfy the test requirements given that NanoXML contains thousands of statements but a small amount of test cases. Thus, CTSR exhibits a higher Reduction result. Moreover, the results suggest that CTSR is not very different with respect to the DTSR. This indicates that DTSR techniques effectively reduce the size of the original test-suite even without the whole testing information.

Tables 4 and 5 present the Tarantula Expense and the HSSF Expense, respectively, of DTSR techniques (columns 6–10), as well as CTSR (columns 2–3) and RTSR techniques (columns 4–5) while using a 20-

Table 3 Reduction of the reduced test-suite when window width is 20

Program	cov	par	rand_{cov}	rand_{par}	Hamming	Levenshtein	Cartesian	Manhattan	CPM
JTcas	3.27	3.65	4.92	5.56	3.94	4.10	4.08	4.01	4.86
Tot_info	3.85	4.92	6.58	8.18	4.24	4.73	4.50	4.62	6.50
Schedule1	3.71	4.92	6.64	9.10	3.88	4.20	4.03	4.26	6.50
Schedule2	3.71	7.10	5.69	8.70	3.56	3.95	4.84	3.62	6.73
Print_tokens1	3.46	4.58	5.98	6.69	3.71	3.85	4.04	4.01	–
Print_tokens2	4.08	5.50	6.22	8.63	4.56	4.71	4.67	4.59	–
NanoXML	47.52	57.58	90.67	83.19	17.91	18.81	20.41	31.57	–
Siena	8.48	11.62	17.70	21.99	7.23	8.72	7.16	7.19	–
ave_4	3.52	4.47	5.65	6.98	3.99	4.28	4.27	4.18	5.70
$\text{ave}_{\text{Siemens}}$	3.58	4.59	5.74	7.14	4.03	4.29	4.29	4.21	–
ave_{SIR}	37.76	46.09	72.43	67.89	15.24	16.29	17.10	25.48	–

Table 4 Tarantula Expense of the reduced test-suite when window width is 20

Program	cov	par	rand_{cov}	rand_{par}	Hamming	Levenshtein	Cartesian	Manhattan	CPM	Origin
JTcas	17.14	16.77	17.31	17.38	17.16	16.95	17.40	17.08	17.44	16.96
Tot_info	25.72	19.64	24.75	23.83	19.37	21.31	23.82	24.38	21.16	21.20
Schedule1	7.08	6.45	7.42	7.48	7.99	6.79	6.54	8.09	6.46	6.90
Schedule2	33.87	34.34	34.10	35.38	36.26	32.76	33.65	36.43	30.38	33.05
Print_tokens1	27.83	24.01	28.86	29.33	37.39	26.40	29.12	27.73	–	26.05
Print_tokens2	24.18	24.11	31.66	28.78	32.25	29.86	27.26	26.24	–	27.38
NanoXML	7.61	7.54	10.26	6.10	8.15	5.75	6.74	6.63	–	7.44
Siena	4.17	4.55	4.15	4.07	4.79	4.43	4.67	4.67	–	4.20
ave_4	20.11	18.13	19.97	19.86	18.59	18.59	19.61	20.03	18.55	18.61
$\text{ave}_{\text{Siemens}}$	21.13	19.22	21.90	21.53	21.48	20.40	21.15	21.28	–	20.11
ave_{SIR}	6.75	6.79	8.73	5.59	7.31	5.42	6.22	6.14	–	6.63
p_4	0.1326	0.3620	0.0814	0.0406	0.2841	0.2203	0.1420	0.0441	0.2034	–
p_{Siemens}	0.2588	0.0863	0.0140	0.0066	0.0789	0.1812	0.0691	0.0535	–	–
p_{SIR}	0.3056	0.1614	0.2556	0.2192	0.0293	0.2931	0.4382	0.4174	–	–

width window. In order to evaluate the impact of reduced test cases on TBFL, the Tarantula experiment and HSSF experiment were also performed on the original test-suite origin (column 11). Rows ave_4, $\text{ave}_{\text{Siemens}}$, and ave_{SIR} present the average Expense of the four former subjects, the Siemens subjects and the SIR subjects, respectively. It was observed that in most cases, par exhibited the best average Expense. This indicated that partition requirement is necessary in TSR. The results also indicated that origin did not exhibit the best results across all subjects. This indicated that the use of a larger size test-suite may not provide a better fault locating result with Tarantula or HSSF while the redundant test cases may exert a negative influence on suspiciousness estimation (e.g., more test cases of Coincidental Correctness [21]). Additionally, the results show that HSSF is a better risk evaluation formula when compared with Tarantula, and this is consistent with the theoretical analysis [9].

Additionally, in order to verify whether the original results are better than the TSR results, paired Wilcoxon tests were applied between column origin and the rest of nine columns (columns 2–10) in Tables 4 and 5, respectively. The last three rows indicate the p-values of the former four subjects p_4, the Siemens subjects p_{Siemens}, and the SIR subjects p_{SIR}. With respect to the CTSR results and DTSR results, most of the p-values exceeded 0.05 (i.e., $\frac{11}{12}$ in CTSR, $\frac{29}{32}$ in DTSR). This indicates that the hypothesis that the fault locating effectiveness of origin exceeds that of CTSR and DTSR should be rejected in most cases. With respect to the RTSR results, a third of the p-values ($\frac{4}{12}$) were less than 0.05 and all of them corresponded to Siemens subjects. Furthermore, ave_4 and $\text{ave}_{\text{Siemens}}$ of origin always exceeded that of RTSR, and thus RTSR may affect the effectiveness of TBFL. Therefore, it is concluded that the proposed DTSR techniques did not affect the effectiveness of TBFL.

Finally, the TimeCost of TBFL applied to the DTSR techniques and the compared CTSR and RTSR

Table 5 HSSF Expense of the reduced test-suite when window width is 20

Program	cov	par	rand_{cov}	rand_{par}	Hamming	Levenshtein	Cartesian	Manhattan	CPM	Origin
JTcas	14.78	14.55	14.76	14.82	14.45	14.95	14.97	12.54	15.07	13.91
Tot_info	20.32	13.45	19.31	17.04	18.23	17.64	20.28	20.37	13.74	18.49
Schedule1	6.95	6.84	7.47	6.72	7.74	6.58	6.95	8.40	6.33	6.47
Schedule2	37.86	34.57	35.53	40.94	41.97	32.25	37.31	39.29	32.91	33.48
Print_tokens1	21.99	18.04	20.64	19.72	22.27	20.54	26.02	17.94	–	20.76
Print_tokens2	23.85	22.52	32.78	24.33	25.64	20.15	21.35	24.04	–	23.17
NanoXML	3.77	3.55	4.53	3.71	6.93	4.54	5.66	4.47	–	5.40
Siena	4.42	5.06	4.92	4.85	5.00	5.48	4.50	4.48	–	5.41
ave_4	17.69	15.26	17.22	17.02	17.37	16.44	17.72	16.83	15.41	16.26
$\text{ave}_{\text{Siemens}}$	18.67	16.25	19.15	18.01	18.63	17.15	18.74	17.69	–	17.34
ave_{SIR}	3.93	3.93	4.63	4.00	6.45	4.78	5.37	4.47	–	5.40
p_4	0.0600	0.3227	0.0137	0.2132	0.1509	0.3395	0.0458	0.1273	0.2311	–
p_{Siemens}	0.0224	0.1642	0.0847	0.1997	0.0645	0.1369	0.0726	0.2152	–	–
p_{SIR}	0.0763	0.1905	0.0867	0.1481	0.3333	0.2759	0.3386	0.0017	–	–

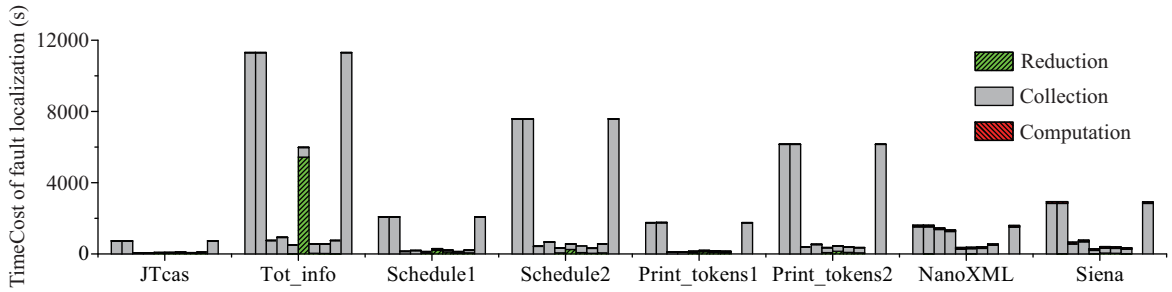

Figure 5 (Color online) TimeCost (s) of the fault localization when window width is 20.

techniques are presented. Figure 5 shows the TimeCost when a 20-width window is used. The horizontal axis represents the subjects, and the vertical axis represents the TimeCost of fault localization. Without loss of generality, the time overhead of Tarantula was measured as the representative of TBFL techniques. The TSR techniques compared in this figure are as follows: two CTSR techniques (cov and par), two RTSR techniques (rand_{cov} and rand_{par}), and five proposed DTSR techniques (Hamming, Levenshtein, Cartesian, Manhattan, and CPM). For further comparison, the TimeCost of applying the original test-suite to Tarantula were provided. Thus, for each subject, ten different TimeCost values were collected and recorded by applying 9 TSR outputs and the original test-suite on Tarantula.

Additionally, TimeCost was measured by considering the time cost of $\text{TimeCost}_{\text{reduction}}$ (green area), $\text{TimeCost}_{\text{collection}}$ (grey area), and $\text{TimeCost}_{\text{computation}}$ (red area). It was observed that TBFL spends maximum time on coverage information collection and spent minimum time on suspiciousness computation. An exception occurred in Levenshtein when it was applied to Tot_info. Levenshtein is a high complexity string distance estimation approach with a cost (in time and memory) that corresponds to $O(m \times n)$ where m and n denote lengths of the two strings. The test-suite of Tot_info was constructed by thousands of long strings, and thus generating the distance matrix of $\text{TS}_{\text{Tot\_info}}$ definitely significantly increased the time cost on TSR. Other subjects do not exhibit this problem. All traces were necessary to conduct a fault localization by using CTSR results as well as the original test-suite, and this entailed more time than those of DTSR and RTSR. However, the time of DTSR was not significantly different from the time of RTSR on Siemens programs. Siemens corresponded to small-scale programs, and thus a small number of test cases satisfied the test requirements. Therefore, both DTSR and RTSR spent shorter time on coverage information collection. The reduced results of DTSR were fewer than those of RTSR. However, they exhibited similar fault locating time. However, DTSR required a slightly lower time when compared with those of other TSR techniques on subjects NanoXML and Siena. This result indicated that the proposed DTSR techniques effectively improve the efficiency of TBFL.

5 Discussion

5.1 Threats to validity

Internal validity is an issue with respect to the proposed approach due to the errors that affect the experimental results without developers's knowledge. The main issues with respect to internal validity are as follows. First, the compared techniques used in this experiment constitute issues. In the experiment, the CTSR techniques are selected and implemented based on the description of Yu et al. [6] and Hao et al. [4]. Second, the TBFL techniques used in the experiment correspond to a potential issue. In order to avoid this, two typical TBFL techniques are selected, namely Tarantula [8] and HSSF [9], which are widely used in TBFL studies. Third, the initial failed test case may impact the results of the TSR and the TBFL. This is reduced by using each of the failed test cases as the initial failed test case. Finally, the manual analysis of the program specification constitutes an issue because the experimental results of CPM technique may be affected by the capacity of the researcher who identifies the categories and choices. This is avoided by involving multiple researchers to perform the category partition. This is further reduced by involving software developers to identify the categories and choices.

There are also threats to external validity of the study wherein the results may not be generalized to all systems. In the experiment, the effects of TSR on TBFL were evaluated by using only 8 programs, and thus it was possible to definitively state that the findings are applicable for programs in general. The study attempted to address a few of the uncertainties by evaluating a variety of programs with diverse sizes. Another external validity threat is that the faults correspond to seeded faults that are carefully designed by Do et al. [20]. Seeded faults are widely used in several previous studies although they do not constitute naturally occurring faults.

Construct validity is also a threat to the proposed approach due to the complexity of the proposed DTSR techniques. When compared with CTSR [4, 6] and RTSR, the DTSR requires additional calculations of the distances among the test cases, and this increases the complexity of fault localization. However, although a program may contain several faulty versions, the program usually contains only one test-suite. Thus, it is only necessary to perform a calculation of distances among the test cases. An increase in the number of faulty versions decreases the average distance calculation time. Conversely, the program should be repeatedly modified and executed during program debugging. Thus, the time for program tracing exceeds that of distance calculation. Therefore, the additional step is not a threat to the proposed approach.

5.2 Related work

The proposed approach is related to TSR because the approach provides strategies on TSR to improve the efficiency of program debugging. Additionally, the goal of TSR in this study involves selecting test cases as inputs to TBFL, and the study is also related to testing based fault localization. Both fore-mentioned studies are briefly reviewed.

5.2.1 *Test-suite reduction*

The aim of TSR involves reducing the size of a given test-suite by identifying and eliminating redundant test cases during software maintenance. The random technique corresponds to a straightforward way to test-suite reduction although it cannot guarantee the quality of reduced test-suite [4, 22]. In addition to this technique, most existing TSR approaches are based on structural coverage such as statement coverage [11], branch coverage [12] and MC/DC [23], or black-box information such as input difference [24], category partition result [25], and service interaction [26]. Rothermel and Harrold [27] surveyed test-suite selection techniques and introduced a framework to evaluate different techniques. Wong et al. [28] suggested that TSR techniques could lower the number of executed test cases without significantly reducing fault detection capabilities of test-suites. However, Rothermel et al. [29] examined costs and benefits of TSR techniques, and their results indicated that TSR could severely compromise fault detection capabilities of test-suites.

The present study corresponds to TSR because the aim of the proposed approach involves reducing the size of the given test-suite. Traditional TSR techniques aim at facilitating fault detection while the aim of the proposed approach involves facilitating fault localization. The proposed approach corresponds to distance based test-suite reduction and does not require collecting coverage information of the whole test-suite. Thus, it can effectively reduce the TSR time cost.

5.2.2 *Testing based fault localization*

Testing based fault localization (TBFL) approaches mainly differ in the granularity of execution collected (e.g., program statement [8,9], predicate [30]), and in the strategies employed for suspicion computation [1, 2]. The present study is different from extant studies on TBFL because the proposed approach focuses on reducing efforts on coverage information collection. The DTSR approach was applied to two typical fault localization approaches [8, 9]. The results indicate that the time needed for TBFL is effectively reduced while the effectiveness of TBFL is maintained.

A few studies focused on evolving test-suite (e.g., test-suite generation [10], test-suite reduction [4,6,31]) to aid in locating faults. Baudry [10] analyzed the type of information needed for TBFL and proposed the TfD criterion (test for diagnosis) to improve the quality of a test-suite. Yu et al. [6] proposed two TSR techniques, namely statement based and vector based techniques, to investigate the impact of test-suite composition on the effectiveness of TBFL. The results revealed that TBFL always exhibited a worse performance if a test-suite was reduced by a statement based technique. In contrast, TBFL always exhibited the same performance if a test-suite was reduced by vector based technique. Hao et al. [4] emphasized the test oracle problem in TBFL. Developers spend a considerable amount of time on inspecting results due to the lack of expected outputs. Thus, they proposed three strategies to reduce the size of test-suite. Vidacs et al. [31] proposed a combined approach to satisfy test requirements of fault detection and fault localization. The results indicated that the combined approach obtained a significant tradeoff between the capability of the fault detection and fault localization. In a manner similar to the above four studies, the aim of the present study involves improving the performance of TBFL by improving the quality of the given test-suite. The currents study is different from extant studies because the proposed approach relies on the distances and categories among the test cases as opposed to the testing information of all test cases. This avoids the expensive process of coverage information collection with respect to the complete test-suite.

6 Conclusion

In this study, the distance based test-suite reduction (DTSR) technique is proposed to enhance the efficiency of testing based fault localization and to verify its effectiveness by performing an empirical study. Given the use of DTSR, it is only necessary to trace and inspect test cases in the reduced test-suite while conducting a testing based fault localization. This aids developers in avoiding the costly process of running the entire information collection. The results of the empirical study indicate that the proposed approach enables developers to select a small subset of test cases while continuing to achieve effective fault locating results. A future study will involve applying the proposed approach to an increased number of subjects and conducting more detailed empirical studies.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61673384, 61502497, 61562015), Guangxi Key Laboratory of Trusted Software (Grant Nos. kx201609, kx201532), Scientific Research Innovation Project for Graduate Students of Jiangsu Province (Grant No. KYLX_1390), Science and Technology Program of Xuzhou (Grant No. KC15SM051), and China Postdoctoral Science Foundation (Grant No. 2015M581887).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Wong W E, Gao R Z, Li Y H, et al. A survey on software fault localization. IEEE Trans Softw Eng, 2016, 42: 707–740
2 Xie X Y, Chen T Y, Kuo F C, et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans Softw Eng Meth, 2013, 22: 402–418
3 Ledru Y, Petrenko A, Boroday S, et al. Prioritizing test cases with string distances. Automat Softw Eng, 2012, 19: 65–95
4 Hao D, Xie T, Zhang L, et al. Test input reduction for result inspection to facilitate fault localization. Automat Softw Eng, 2010, 17: 5–31
5 Gong L, Lo D, Jiang L X, et al. Diversity maximization speedup for fault localization. In: Proceedings of the 27th International Conference on Automated Software Engineering, Essen, 2012. 30–39
6 Yu Y B, Jones J A, Harrold M J. An empirical study of the effects of test-suite reduction on fault localization. In: Proceedings of the 30th International Conference on Software Engineering, Leipzig, 2008. 201–210
7 Shi A, Yung T, Gyori A, et al. Comparing and combining test-suite reduction and regression test selection. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, Bergamo, 2015. 237–247
8 Jones J A, Harrold M J, Stasko J. Visualization of test information to assist fault localization. In: Proceedings of the 24th International Conference on Software Engineering, Orlando, 2002. 467–477
9 Ju X L, Jiang S J, Chen X, et al. HSFal: effective fault localization using hybrid spectrum of full slices and execution slices. J Syst Softw, 2014, 90: 3–17
10 Baudry B. Improving test suites for efficient fault localization. In: Proceedings of the 28th International Conference on Software Engineering, Shanghai, 2006. 82–91
11 Rothermel G, Harrold M J. Empirical studies of a safe regression test selection technique. IEEE Trans Softw Eng, 1998, 24: 401–419
12 Wang R C, Qu B B, Lu Y S. Empirical study of the effects of different profiles on regression test case reduction. IET Softw, 2015, 9: 29–38
13 Jiang B, Zhai K, Chan W K, et al. On the adoption of MC/DC and control-flow adequacy for a tight integration of program testing and statistical fault localization. Inform Softw Tech, 2013, 55: 897–917
14 Hamming R W. Error detecting and error correcting codes. BELL Labs Tech J, 1950, 29: 147–160
15 Levenshtein V I. Binary codes capable of correcting deletions, insertions and reversals. Dokl Akad Nauk Sssr, 1965, 10: 707–710
16 Chen T Y, Poon P L, Tang S F, et al. On the identification of categories and choices for specification-based test case generation. Inform Softw Tech, 2004, 46: 887–898
17 Zhang X Y, Towey D, Chen T Y, et al. Using partition information to prioritize test cases for fault localization. In: Proceedings of the 39th Annual Computer Software and Applications Conference, Taichung, 2002. 121–126
18 Ostrand T J, Balcer M J. The category-partition method for specifying and generating fuctional tests. Commun ACM, 1988, 31: 676–686
19 Santelices R, Jones J A, Yu Y B, et al. Lightweight fault-localization using multiple coverage types. In: Proceedings of the 31st International Conference on Software Engineering, Vancouver, 2009. 56–66
20 Do H, Elbaum S, Rothermel G. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. Empir Softw Eng, 2005, 10: 405–435
21 Masri W, Assi R A. Prevalence of coincidental correctness and mitigation of its impact on fault localization. ACM Trans Softw Eng Meth, 2014, 23: 294–304
22 Zhang Z Y, Chen Z Y, Gao R Z, et al. An empirical study on constraint optimization techniques for test generation. Sci China Inf Sci, 2017, 60: 012105
23 Fang C R, Chen Z Y, Xu B W. Comparing logic coverage criteria on test case prioritization. Sci China Inf Sci, 2012, 55: 2826–2840
24 Jiang B, Zhang Z Y, Chan W K, et al. Adaptive random test case prioritization. In: Proceedings of the 24th International Conference on Automated Software Engineering, Auckland, 2009. 233–244
25 Zhang X F, Xie X Y, Chen T Y. Test case prioritization using adaptive random sequence with category-partition-based distance. In: Proceedings of the 2016 International Conference on Software Quality, Reliability and Security, Vienna, 2016. 374–385
26 Chen L, Wang Z Y, Xu L, et al. Test case prioritization for web service regression testing. In: Proceedings of the 5th International Symposium on Service-Oriented System Engineering, Nanjing, 2010. 173–178
27 Rothermel G, Harrold M J. Analyzing regression test selection techniques. IEEE Trans Softw Eng, 1996, 22: 529–551
28 Wong W E, Horgan J R, London S, et al. Effect of test set minimization on fault detection effectiveness. In: Proceedings of the 17th International Conference on Software Engineering, Washington, 1995. 41–50
29 Rothermel G, Harrold M J, Ronne J V. Empirical studies of test-suite reduction. Softw Test Verif Rel, 2002, 12: 219–249
30 Wang X Y, Jiang S J, Ju X L, et al. Mitigating the dependence confounding effect for effective predicate-based statistical fault localization. In: Proceedings of the 39th Annual Computer Software and Applications Conference, Taichung, 2015. 105–114
31 Vidacs L, Beszedes A, Tengeri D, et al. Test suite reduction for fault detection and localization: a combined approach. In: Proceedings of the 2014 IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering, Antwerp, 2014. 204–213