JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 40, XXX-XXX (2024)
DOI: 10.6688/JISE.202411_40(6).0009

A Similarity-Based Approach to Identify and Manipulate Coincidental Correct Test Cases for Fault Localization

MOHAMMAD MAHDI ESTESNAEI, SAEED ARABAN+
AND AHAD HARATI
Department of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Khorasan Razavi, Iran
E-mail: estesnaei@mail.um.ac.ir; {araban; a.harati}@um.ac.ir

Spectrum-based fault localization (SBFL) is one of the most popular fault localization techniques that uses coverage information and test results to calculate a suspicious score for every program statement. The effectiveness of SBFL suffers from the occurrences of coincidental correctness, which occurs when a fault is executed but no failure is detected. Identifying coincidental correct (CC) test cases can be modeled as a classification problem. Except in exceptional cases, proven identification of CC tests is not possible, so instead of using 0/1 results, we propose a similarity-based approach to identify CC test cases. A strategy is suggested to manipulate CC test cases for SBFL. In the first step, a low-cost computational method is proposed to identify CC test cases based on the similarity of the passed executions to the failed ones. Then, we proposed new similarity measures based on the original ones (such as *Jaccard* similarity and *Euclidean* distance) and presented a method to identify proven CC. Finally, a weighted CC test case manipulation strategy is proposed to mitigate the negative impact of CC test cases in SBFL. We evaluated the proposed method by conducting extensive experiments on 443 faulty versions of 13 popular subject programs, containing artificial and real faults. The results show that the proposed method can improve the accuracy of SBFL techniques with a very low computational cost.

Keywords: software debugging, spectrum-based fault localization, coincidental correct test cases, similarity measures

1. INTRODUCTION

Software testing is the most popular method used for enhancing the quality of software [1], and the most crucial step in it is software debugging. This step is much more complicated and costly, especially for large and complex software [2]. Testing reveals the presence of faults, while debugging locates and corrects faults in the program. When software testing uncovers the faults, the software developer needs to localize the faulty portions of code and correct them. Fault localization is the most expensive activity in program debugging [3].

Received July 31, 2023; revised November 1 & December 21, 2023; accepted January 1, 2024.
Communicated by Yu Chin Cheng.
+ Corresponding author.

One promising approach towards fault localization is Spectrum-Based Fault Localization (SBFL) [4]. In SBFL, coverage information and test case results are used to calculate the suspiciousness score of each program statement. The more suspicious a statement is, the more likely it is to be faulty. A ranking list containing all ranked statements sorted by their suspiciousness score is used for top-down checking by developers to locate faulty statements. When the rank of the faulty statement is higher, the effort to locate it is less.

A coincidental correct (CC) test case executes the faulty elements of the program under test (PUT) but does not reveal any failure. Such a test case reduces the effectiveness of SBFL techniques because it is considered a passed test case. CCs are prevalent [5], and when they are present, the faulty statement is likely to be ranked as less suspicious than when they are not [6].

Several methods have been proposed to identify CCs based on different algorithms, such as clustering algorithms, like *k*-means [6-13], and classification algorithms like KNN [2, 14] and SVM [15, 16].

The insight behind the clustering method is that tests in the same cluster have similar behaviors, and a passed test in a cluster with many failed tests is highly possible to be CC, because it has the potential to execute faulty elements as those failed ones do [12]. To identify CCs using the clustering method, it is common to use the execution profile of each test as an object and the *Euclidean* distance as the distance function [2, 6-8, 14, 17, 18], but determining the appropriate number of clusters is challenging. A larger number of clusters yield a higher rate of false negatives but a lower rate of false positives [12]. The main idea behind the clustering method is that the execution profile of a CC test is similar to the execution profile of a failed test. Therefore, the CC probability (CC weight) for a passed test can be considered as the degree of similarity between its execution profile and the execution profile of failed test cases. By assigning weight to failed tests, they can be more influential in ranking [19]. In this paper, we apply this method to passed test cases and introduce a weighted CC test case manipulation strategy to mitigate the negative impact of CC test cases.

The main contributions of this paper are as follows:

1. A low-cost similarity-based CC identification method is proposed.
2. Several new similarity measures are introduced based on the original ones.
3. We developed the method to identify proven CC test cases.
4. A new CC test case manipulation strategy (*Weighted* strategy) is introduced to improve SBFL, and experimental results show that the effectiveness of SBFL is enhanced.

The rest of this paper is organized as follows. Section 2 describes the background and related works on SBFL and CC test case identification. Section 3 details the CC test case identification and manipulation strategy. The experiment design and analysis of results are shown in Section 4. Section 5 discusses threats to validity. Finally, Section 6 concludes the paper and outlines future directions.

2. BACKGROUND AND RELATED WORK

In this section, we first mention the background of SBFL techniques. Then, we review the proposed methods to reduce the negative impact of CC on SBFL.

Table 1. a_{ef}, a_{ep}, a_{nf} **and** a_{np} **values for the statement** st_j**.**

Values
$a_{ef} = \sum_{t \in T_F} e_t^j$
$a_{ep} = \sum_{t \in T_P} e_t^j$
$a_{nf} = \sum_{t \in T_F} (1 - e_t^j)$
$a_{np} = \sum_{t \in T_P} (1 - e_t^j)$

Table 2. Four SBFL formulas.

Technique	Formula
Tarantula	$Susp(s_i) = \frac{a_{ef}/(a_{ef}+a_{nf})}{a_{ef}/(a_{ef}+a_{nf}) + a_{ep}/(a_{ep}+a_{np})}$
Ochiai	$Susp(s_i) = \frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf}) \times (a_{ef}+a_{ep})}}$
DStar*	$Susp(s_i) = \frac{a_{ef}^*}{a_{ep}+a_{nf}}$
Op	$Susp(s_i) = a_{ef} - \frac{a_{ef}}{a_{ep}+a_{np}+1}$

2.1 Spectrum-Based Fault Localization

SBFL uses test case coverage, commonly known as Spectrum. Suppose PUT contains n statements, which we call $st_1, st_2, \ldots, st_n$. Also, we have a test suite T that comprises m test cases $t_1, t_2, ..., t_m$, where each test case consists of input parameters and expected results. A test case is executed on the PUT and its output, known as the actual result, is compared with the expected result. If the actual result is the same as the expected result, the test case is labeled as a passed test case (P). On the other hand, if the actual result is different from the expected result, the test case is labeled as a failed test case (F). Therefore, the set of test cases in T is divided into two groups: T_P for passed test cases and T_F for failed test cases. As shown in Fig. 1, the statement coverage of each test case t_i can be represented as an n-dimensional binary vector $e_{t_i} = < e_{t_i}^1, e_{t_i}^2, \ldots, e_{t_i}^j, \ldots, e_{t_i}^n >$ which is called statement coverage vector (SCV) of t_i. If the test case t_i covers the jth statement, the value of $e_{t_i}^j$ is 1, otherwise, it is 0. In SBFL, four coefficients are used to calculate the suspiciousness score of program statements. These coefficients are a_{ep}, a_{np}, a_{ef} and a_{nf}. The first part of the subscript indicates whether the statement is executed (e) or not (n), and the second one indicates whether the test case is passed (p) or failed (f). These values are listed in Table 1.

Besides statement coverage, there are different types of elements to record test case coverage information, such as call sequences [20], branches, du-pairs [21], statement frequency [22], and the time of function calls [23].

There are different SBFL techniques in the literature. For instance, the suspiciousness score calculation formulas of four popular SFBL techniques, namely *Tarantula* [24], *Ochiai* [25], *DStar** [26], and *Op* [27] are shown in Table 2. Ajibode *et al*. [28] defined four new metrics from the program spectrum and then they combined these metrics to propose a new heuristic SBFL formula (MECO).

To better illustrate how SBFL techniques work, consider the example shown in Fig. 1, which includes two seeded bugs in statements st_3 and st_6. The test suite contains 10 test cases named t_1 to t_{10}. For each test case, inputs, expected result, actual result, statement coverage vector, and test result are shown. Test results are labeled with P and F, which

	Name of test case	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
		-2	-2	2	2	2	2	2	2	20	5
	Inputs	10	15	-3	13	-5	-3	-4	-3	-3	3
		4	4	4	4	1	8	15	14	12	4
	Expected Result	13	17	3	108	-4	7	-53	-35	-26	68
	Actual Result	13	17	3	108	-4	7	-53	-35	24	71
	TestProg(int a, int b, int c)										
st_1	{ int r=0;	1	1	1	1	1	1	1	1	1	1
st_2	if (a>0)	1	1	1	1	1	1	1	1	1	1
st_3	**{r=a*2; //correct: r=a+2**	0	0	1	1	1	1	1	1	1	1
st_4	if(b>0)	0	0	1	1	1	1	1	1	1	1
st_5	$r = r + a * b * c$;	0	0	0	1	0	0	0	0	0	1
st_6	**else if (c<10) // correct: c<13**	0	0	1	0	1	1	1	1	1	0
st_7	$r = r + a * b + c$;	0	0	1	0	1	1	0	0	0	0
st_8	else $r = r + a + b * c$;	0	0	0	0	0	0	1	1	1	0
st_9	}else $r = a + b + c$;	1	1	0	0	0	0	0	0	0	0
st_{10}	if ($a + b$<10)	1	1	1	1	1	1	1	1	1	1
st_{11}	r++;	1	0	1	0	1	1	1	1	0	1
st_{12}	printf("%d",r);}	1	1	1	1	1	1	1	1	1	1
	Test Results (P: Passed , F:Failed , P*: Proven CC)	P	P	P	P	P	P	P*	P*	F	F

Fig. 1. A sample program with two seeded faults and a set of ten test cases.

Table 3. The suspiciousness and rank list of statements for the example in Fig. 1.

s#	a_{ep}	a_{np}	a_{ep}	a_{nf}	*Ochiai*		*Tarantula*		*Dstar*^{3}	
					Susp	Rank	Susp	Rank	Susp	Rank
st_1	8	0	2	0	0.45	4	0.50	5	1.00	3
st_2	8	0	2	0	0.45	4	0.50	5	1.00	3
st_3	6	2	2	0	0.50	1	0.57	3	1.33	1
st_4	6	2	2	0	0.50	1	0.57	3	1.33	1
st_5	1	7	1	1	0.50	1	0.80	1	0.50	7
st_6	5	3	1	1	0.29	9	0.44	9	0.17	9
st_7	3	5	0	2	0.00	11	0.00	11	0.00	11
st_8	2	6	1	1	0.41	8	0.67	2	0.33	8
st_9	2	6	0	2	0.00	11	0.00	11	0.00	11
st_{10}	8	0	2	0	0.45	4	0.50	5	1.00	3
st_{11}	6	2	1	1	0.27	10	0.40	10	0.14	10
st_{12}	8	0	2	0	0.45	4	0.50	5	1.00	3

means passed and failed, respectively. A proven CC test case is labeled with P*, which refers to a passed test case that can be proven to be CC. More details on proven CC are provided in Section 3.2.

The suspiciousness and rank of each statement calculated by the *Ochiai*, *Tarantula*, and *DStar*^{3} formulas, for the example shown in Fig. 1, are presented in Table 3. The fault localization accuracy of the *DStar** is the best when parameter * is set to 3 [26]. In Table 3, the Rank columns show the maximum number of statements that would have to be examined if that statement were the first statement of that particular suspiciousness level chosen for examination. In this case, for the faulty statement st_3, *Ochiai* and *DStar*^{3} have better fault localization effectiveness than *Tarantula* and the faulty statement st_6 is ranked ninth by all three SBFL formulas.

2.2 Coincidental Correct Test Case

The Propagation-Infection-Execution (PIE) model proposed by Voas [29] emphasizes that the occurrence of a failure must satisfy three conditions: (a) the faulty statement was

executed, (b) the program has transitioned to an infectious state, and (c) the infection state propagated to program output. Coincidental correctness occurs when condition (a) is met, but condition (c) is not. There are two types of coincidental correctness: weak and strong. In weak CC, execution of the faulty statement is met, whereas the infected state might or might not met, whereas, in strong CC, both conditions are met [6]. Strong and weak CC are prevalent [5, 30]. They have a destructive effect on fault localization [5, 29, 31].

Many researchers suggest different approaches to identify CC test cases and then try to eliminate their adverse effects with cleansing or relabeling strategies. The cleansing strategy is to remove identified CC test cases from the test set, and the relabeling strategy is to label those test cases from passed to failed [9, 12]. Both the cleansing strategy and the relabeling strategy are based on the distinct result of CC classification. In the fuzzy-based classification method, the possibility of CC test cases (CC weight) is given rather than 1 or 0. Therefore, Liu *et al*. [2, 14] proposed three fuzzy-based strategies to manipulate CC test cases for SBFL. They redefined *Ochiai* [14] and $DStar^3$ [2] with three fuzzy-based strategies $F-cleansing$, $F-relabelling$ and $F-exchanging$ based on the coincidental correct probability of each passed test case.

To identify CC test cases, several techniques, including unsupervised learning like clustering [6, 7, 9, 10, 12, 13, 32] and supervised learning like support vector machines (SVM) [15, 16] and K-nearest neighbors (KNN) [2, 14] are used to apply the similarity metric to the CC identification problem. In clustering-based techniques, test cases are grouped into different clusters using their structural execution profile, and K-means is mainly employed as the clustering algorithm [6--13]. The insight behind cluster analysis is that tests in the same cluster have similar behaviors. Thus, a passed test in a cluster with many failed tests is highly possible to be coincidentally correct because it has the potential to execute the faulty elements as those failed ones do [12]. Clustering-based techniques suffer from the challenging task of selecting the number of clusters, and some researchers set the number of clusters according to the size of the test set [9, 12]. *Euclidean* distance has been widely used as the distance function for the clustering of test case execution profile [6-8, 17].

Some researchers proposed different approaches to improve the effectiveness of SBFL techniques in the presence of CC test cases. Bandyopadhyay and Ghosh [33] present an approach to improve the effectiveness of SBFL by incorporating the relative importance of different test cases in calculating suspiciousness scores. The importance of a passing test case is proportional to its average proximity to the failing test cases. Bandyopadhyay [34] assigns weights to passing test cases such that the test cases that are likely to be coincidentally correct obtain low weights and finally defines the weighted *Ochiai* formula. Zhou *et al*. [35] estimate the probability that coincidental correctness (CCP) happens for each program execution using dynamic data-flow analysis and control-flow analysis. Then, by changing the calculation of the values of some variables in the *Tarantula* formula using CCP, they defined a new suspicious metric. Wang *et al*. [36] refine code coverage of test runs using control- and data flow patterns prescribed by different fault types. They conjecture that this extra information can strengthen the correlations between program failures and the coverage of faulty program entities.

Metallaxis [37] is a fault localization approach based on mutation analysis that tries to reduce the negative impact of CC on SBFL indirectly. The idea behind this method is that mutants that are killed mainly through failing tests provide a good indication of the

location of a fault. In order to reduce the overhead of mutation analysis, Papadakis and Traon [38] proposed mutation-based fault localization, which uses a sufficient mutant set to locate faulty statements efficiently. FCCI [18] models the CC identification process as a decision-making system by constructing a fuzzy expert system and uses the *Euclidean* similarity of passed executions to the failed ones as a factor to identify CC test cases.

Given the above discussion, clustering-based CC identification methods need to measure the similarity between the execution profile of the passed and failed runs, and some methods also need to assign weights to passed test cases based on the CC probability. The SCV of a test case as its execution profile can be easily extracted with low computational cost. Therefore, research on measuring the similarity between SCV of passed and failed test cases seems necessary.

3. CC TEST CASE IDENTIFICATION AND MANIPULATION

3.1 Similarity-Based CC Test Case Identification

In data science, the similarity measure is often expressed as a number between zero and one and gets a higher value when the data samples are more alike. Sometimes, the measure of similarity is expressed as a distance (like *Euclidean* distance), and a large distance means a low degree of similarity and vice versa.

We propose a similarity-based method to identify CC test cases and the SCV of each test is regarded as an object. The insight behind our method is that a passed test whose SCV is very similar to the SCV of a failed test is highly possible to be CC. Therefore, we assign weight to a passed test (CC weight) based on its similarity (s method) to a failed test. In Eq. (1), the CC weight of passed test case t_p is calculated based on the failed test case t_f using *Sim* similarity measure. Using Eq. (2), we integrate the CC weights calculated for test t_p based on all failed test cases to calculate the CC weight of the passed test case t_p.

$$wcc_s\left(t_p|t_f\right) = Sim(e_{t_p}, e_{t_f}) \tag{1}$$

$$wcc_s(t_p) = \max_{t_f \in T_F} wcc_s(t_p|t_f) \tag{2}$$

Table 4 shows some similarity measures and some of them have been modified to be useful for our method. *Cosine* similarity [39] measures the cosine of the angle between two vectors projected in a multi-dimensional space. We use the original formula to calculate the similarity between the two SCVs. The *Euclidean* distance [40] between two points in *Euclidean* space is the length of a line segment between the two points. The original formula has been modified to calculate a value in the interval [0,1]. A higher value means more similarity.

Jaccard similarity [41] can be used to find the similarity between two sets. The original formula is shown in Eq. (3), where A and B are two sets. We have modified the original formula to be useful for calculating the similarity between two real vectors. For this purpose, t-norm and s-norm operators related to fuzzy logic were used to calculate the intersection and union of two vectors, respectively. In *JaccardMM*, Minimum and Maximum for t-norm and s-norm are used, respectively. In *JaccardAP*, Algebraic Product and Probabilistic SUM are used for t-norm and s-norm, respectively.

Table 4. The similarity measures.

Name	Method	Explanation
Cosine	$Sim(x,y) = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$	Original
Euclidean	$Sim(x,y) = \frac{1}{1+\sqrt{(x_i-y_i)^2}}$	Modified to calculate similarity
JaccardMM	$Sim(x,y) = \frac{\sum min(x_i,y_i)}{\sum max(x_i,y_i)}$	Modified for real vector comparison
JaccardAP	$Sim(x,y) = \frac{\sum x_i y_i}{\sum x_i+y_i-x_i y_i}$	Modified for real vector comparison

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3}$$

3.2 Weighted and Proven CC Test Case Identification

In [14], the KNN algorithm is used to identify CC tests, and weighted *Euclidean* distance is used as similarity measure, in which the weight value of each statement is calculated by the *Tarantula* formula. We use this idea and introduce Eq. (4), which calculate the CC weight of the passed test case t_p based on the failed test case t_f using weighted similarity (*ws* method). In this Equation, $w = \langle w^1, w^2, ..., w^k, ..., w^n \rangle$ is a real vector and the value of w^k is the normalized suspicious score of statement st_k, which is calculated using a SBFL formula. To calculate the normalized w^k, we divide the suspicious score of st_k by the maximum suspicious score. Eq. (6) shows the product of two real vectors used in Eqs. (4) and (5).

$$wcc_{ws}\left(t_p|t_f\right) = Sim(e_{t_p} \times w, e_{t_f} \times w) \tag{4}$$

$$wcc_{wps}\left(t_p|t_f\right) = Sim(e_{t_p} \times e_{t_f} \times w, e_{t_f} \times w) \tag{5}$$

$$x \times y = \langle x^1 \times y^1, x^2 \times y^2, ..., x^n \times y^n \rangle \tag{6}$$

Each failed test case covers at least one faulty statement. Therefore, if a passed test case covers all statements covered by a failed test case, such a passed test case must cover the faulty statement, then it can be identified as a proven CC test case. As shown in Fig. 1, the statements covered by the passed test case t_7 or t_8 include those covered by the failed test case t_9. Therefore, t_7 and t_8 are proven CC test cases. It is desirable to calculate the CC weight of a proven CC equal to 1, but Eq. (4) does not necessarily calculate the value of 1 for a proven CC. We modify Eq. (4) and introduce Eq. (5), which is based on weighted similarity and can identify proven CC (*wps* method). If the statements covered by test case t_p include all the statements covered by test t_f, then the SCV of test case t_f will act like a mask and the relation $e_{t_p} \times e_{t_f} = e_{t_f}$ will be established. As a result, Eq. (5) calculates the maximum value of CC weight for the passed test case t_p based on the failed test case t_f. Therefore, this equation has the ability to distinguish a proven CC test case.

Table 5. CC weight calculation for passed test cases of program shown in Fig. 1.

Method	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
s	0.33	0.31	0.37	0.50	0.37	0.37	0.50	0.50
ws	0.41	0.41	0.79	0.90	0.79	0.79	0.90	0.90
wps	0.41	0.41	0.80	0.90	0.80	0.80	1.00	1.00

Table 6. Modified inputs of the spectrum-based formula based on each manipulation strategy.

	$F-cleansing$	$F-relabeling$	$F-exchanging$	$F-deleting$
a_{ep}	$a_{ep}-cc_e$	$a_{ep}-cc_e$	$a_{ep}-cc_e$	$a_{ep}-cc_e$
a_{np}			$a_{np}-cc_n$	$a_{np}-cc_n$
a_{ef}		$a_{ef}+cc_e$	$a_{ef}+cc_e$	
a_{nf}			$a_{nf}+cc_n$	

The CC weights of passed test cases for the sample program in Fig. 1 are shown in Table 5. In this table the modified *Euclidean* similarity is used. The *DStar*3 formula is used to calculate the normalized vector w. Normalized means that the suspicious score is divided by the maximum suspicious score. According to the values of column *DStar*3 in Table 3, the vector w is $< 0.75, 0.75, 1, 1, 0.375, 0.125, 0, 0.25, 0, 0.75, 0.11, 0.75 >$. All three proposed methods have calculated more weight for CC test cases $t_3, t_4, t_5, t_6, t_7, t_8$. Based on the *wps* method, the weights of proven CC tests t_7 and t_8 are calculated as 1.

3.3 Weighted CC Test Case Manipulation Strategy

CC weights of passed test cases must be integrated into calculating statement suspiciousness in SBFL. Miao *et al*. [12] proposed two strategies for CC test case manipulation: cleansing, and relabeling. The cleansing strategy is based on removing the identified CC test cases from the test set, and relabeling strategy, flips these test cases from passed to failed. Both strategies are based on the distinct results of CC classification.

To integrate CC weights of passed test cases in SBFL, Li *et al*. [14] introduced three fuzzy-based strategies: $F-cleansing$, $F-relabeling$, and $F-exchanging$. Then, they introduced three modified versions of the *Ochiai* formula based on these three strategies. Li *et al*. [2] proposed a Fuzzy Weighted KNN algorithm ($FW-KNN$) to calculate the CC probability (CC weight) of a passed test case and presented three modified versions of the $DStar^3$ formula based on three strategies $F-cleansing$, $F-relabeling$, and $F-exchanging$. We summarize their three strategies in Table 6 based on the modified coefficients of the SBFL formula (No need to change the SBFL formula), and similarly, a fourth strategy $F-deleting$ can be introduced. The values of cc_e, and cc_n, are shown in Table 7. The cc_e is the sum of the CC weights of passed test cases that covered the statement st_j, and the cc_n is the sum of the CC weights of passed test cases that did not cover the statement st_j. As shown in Table 6, for example, in the $F-deleting$ strategy, we replace the a_{ep} value with $a_{ep}-cc_e$ and the a_{np} value with $a_{np}-cc_n$. The values of a_{ef} and a_{nf} are calculated as usual.

In this paper, we introduce a new strategy for manipulating CC test cases based on assigning weights (*Weighted* strategy) to passed test cases. The weight value depends on the CC weight of the passed test case. First, using Eq. (7), we define a weight $w(t_p)$,

Table 7. The values of cc_e and cc_n for statement st_j.

Values
$cc_e = \sum_{t_p \in T_P} e^j_{t_p} \times wcc(t_p)$
$cc_n = \sum_{t_p \in T_P} (1 - e^j_{t_p}) \times wcc(t_p)$

for each passed test case t_p, which is inversely proportional to the CC weight of t_p. This weight is used in calculating the inputs of the SBFL formula. $wcc(t_p)$ is the CC weight of this test case. $wcc(t)$ is the CC weight of test case t, and $|T_P|$ is the number of passed test cases. To calculate $wcc(t)$ and $wcc(t_p)$, we can use any of the three proposed methods *s*, *ws*, or *wps*.

Based on the *Weighted* strategy, to compute coefficient a_{ep} for a statement st_j, we calculate the sum of weights of passed tests in which it was executed, and to compute coefficient a_{np}, we calculate the sum of weights of passed tests in which it was not executed. We call these modified coefficients a^w_{ep} and a^w_{np}, whose calculation methods are shown in Eqs. (8) and (9), respectively.

$$w(t_p) = \frac{(1 - wcc(t_p)) \times |T_P|}{\sum_{t \in T_P}(1 - wcc(t))} \tag{7}$$

$$a^w_{ep} = \sum_{t \in T_P} e^j_t \times w(t) \tag{8}$$

$$a^w_{np} = \sum_{t \in T_P} (1 - e^j_t) \times w(t) \tag{9}$$

For example, with the data of Table 5, using the *wps* method, the weights for tests $t_1 - t_8$ are 2.52, 2.52, 0.85, 0.41, 0.85, 0.85, 0, and 0 respectively. The a^w_{ep} values for statements $st_1 - st_{12}$ of Fig. 1 are 8, 8, 2.97, 2.97, 0.41, 2.56, 2.56, 0, 5.03, 8, 5.07, and 8 respectively. The greater weights of t_1 and t_2 compared to the weights of t_7 and t_8 lead the a^w_{ep} for the statement st_9 being greater than the a^w_{ep} for the statement st_8, and this is desirable because t_7 and t_8 are CC tests. The a^w_{np} values can be calculated using Eq. (9), or we can simply use $|T_P| - a^w_{ep}$, since the total number of weights is equal to $|T_P|$ (According to Eq. (7)). Table 8 shows the statement suspiciousness and ranking of *Tarantula* for the sample program in Fig. 1, before and after identifying CC test cases using the combination of *wps* method and *Euclidean* similarity measure. As can be seen, the *wps* method has led to assigning the rank of 5 to the faulty statement st_6, which performs better compared to the rank of 9 in the original *Tarantula*.

4. EXPERIMENT AND RESULTS ANALYSIS

4.1 Research Questions

We hope that *s*, *ws*, and *wps* will provide a useful measure for assigning CC weights to the passed test cases to improve SBFL. The empirical study is conducted to address the following research questions:

Table 8. The suspiciousness and rank list of statements for the example in Fig. 1 using: Weighting strategy, *wps* Method with *Euclidean* similarity and comparison with the original Tarantula.

s#	a_{ep}^{w}	a_{np}^{w}	a_{ef}	a_{nf}	*Tarantula*		*Tarantula* (*wps*)	
					Susp	Rank	Susp	Rank
st_1	8.00	0.00	2	0	0.50	5	0.50	6
st_2	8.00	0.00	2	0	0.50	5	0.50	6
st_3	2.97	5.03	2	0	0.57	3	0.73	3
st_4	2.97	5.03	2	0	0.57	3	0.73	3
st_5	0.41	7.59	1	1	0.80	1	0.91	2
st_6	2.56	5.44	1	1	0.44	9	0.61	5
st_7	2.56	5.44	0	2	0.00	11	0.00	11
st_8	0.00	8.00	1	1	0.67	2	1.00	1
st_9	5.03	2.97	0	2	0.00	11	0.00	11
st_{10}	8.00	0.00	2	0	0.50	5	0.50	6
st_{11}	5.07	2.93	1	1	0.40	10	0.44	10
st_{12}	8.00	0.00	2	0	0.50	5	0.50	6

- RQ1: Compared to the state-of-the-art weighted fuzzy classification approach ($FW-KNN$), are our proposed methods better in terms of classification metrics?
- RQ2: Does similarity-based CC test case identification improve the performance of SBFL techniques to locate faults?
- RQ3: Which method, similarity measure and strategy have better results in terms of fault localization metrics?
- RQ4: Compared to $FW-KNN$, which combination is better in terms of fault localization metrics?

  To implement $FW-KNN$, weighted *Euclidean* is used as the similarity measure, where the suspicious value of a statement is considered as the weight of this statement. The suspicious value of each statement is calculated with $DStar^3$ as in the original paper. To compare our method with $FW-KNN$, we used *Ochiai* to calculate the statement weights in *ws* and *wps* methods.

4.2 Subject Programs

The empirical study is conducted on 13 open source and popular programs, including seven programs from Siemens [42] and six programs from Defects4j suite [43] as the subject programs. They have been widely used to evaluate fault localization techniques [18,26,27,42,44--46] and CC identification approaches [2,6,8,11,12,14,15,18,47]. Table 9 shows the subject programs and related versions. The Siemens suite is downloaded from the Software-artifact Infrastructure Repository (SIR[1]), which contains seven small-sized programs with seeded faults. Gcov (GNU call-coverage profiler) is used to collect the statement coverage information of test cases. Some versions were discarded due to compiler error or segment error because it is not possible to extract the statement coverage

[1]http://sir.unl.edu: University of Nebraska.

Table 9. Subject programs.

Program	Faulty versions		LOC	#Tests	#Passed Testes		#CC Testes		Fault type
	All	Used			Max	Ave	Max	Ave	
printtokens	7	6	565	4130	4124	4060	4042	2502	Seeded
printtokens2	10	7	510	4115	4082	3891	3606	1413	Seeded
schedule	9	5	374	2650	2643	2561	2627	1641	Seeded
schedule2	10	9	307	2710	2710	2680	2636	2500	Seeded
tcas	41	38	173	1608	1607	1568	1577	1043	Seeded
tot_info	23	20	412	1052	1050	964	1047	680	Seeded
replace	32	29	563	5542	5542	5437	5211	2272	Seeded
Apache commons-lang	65	50	22K	2245	2240	1745	14	9	Real
Apache commons-math	106	75	85K	3602	3599	2476	57	15	Real
Mockito framework	38	33	23K	1457	1450	1104	218	90	Real
Joda-Time	27	27	28K	4130	4084	3816	242	92	Real
JFree Chart	26	19	96K	2205	2186	1790	43	19	Real
Closure compiler	133	125	90K	7927	7876	6602	4124	608	Real

		Predicted	
		Positive	Negative
Actual	Positive	TP:True Positive	FN: False Negative
	Negative	FP: False Positive	TN: True Negative

Fig. 2. Confusion matrix.

of test cases. We also used Defects4J (one of the largest available datasets of Java defects) for which coverage information (spectrum) is available [48][2]. In this dataset, we excluded versions that did not have a failed test or CC.

4.3 CC Test Case Identification Results

Some metrics that are commonly used to evaluate the performance of the binary classification algorithm, which are calculated based on the confusion matrix as shown in Fig. 2. For CC test case identification problem, True Positive (TP) indicates the number of CC test cases classified accurately, False Positive (FP) is the number of actual passed test cases classified as CC, False Negative (FN) means the number of actual CC test cases classified as passed test cases and True Negative (TN) shows the number of actual passed test cases classified accurately.

Instead of binary classification, we assign a CC weight to a passed test case. Table 10 shows how to calculate TP, FP, FN and TN values, where T_P is the set of all passed test cases, T_{CC} is the set of actual CC test cases, which is a subset of T_P, and $wcc(t)$ is the CC weight calculated for passed test case t. It should be noted that $wcc(t)$ is in the range of [0,1], and a higher value indicates that the passed test is more likely to be CC. To evaluate the effectiveness of s, ws and, wps methods in terms of identifying CC test cases, we use the following measurement metrics:

$$Precision = \frac{TP}{TP+FP} \tag{10}$$

[2]https://bitbucket.org/rjust/fault-localization-data

Table 10. TP, FP, FN and TN values for weight-based evaluation.

Values
$TP = \sum_{t \in T_{CC}} wcc(t)$
$FP = \sum_{t \in T_P - T_{CC}} wcc(t)$
$FN = \sum_{t \in T_{CC}} (1 - wcc(t))$
$TN = \sum_{t \in T_P - T_{CC}} (1 - wcc(t))$

Table 11. Comparison of classification evaluation metrics.

CC weighting approach		*Precision*	*Recall*	$F-Measure$	*Accuracy*	*AUC*
Similarity	Method					
Cosine	*s*	0.3752	**0.8431**	0.5193	0.4752	0.7354
	ws	0.4491	0.7527	0.5626	**0.6064**	0.7964
	wps	**0.4493**	0.7655	**0.5662**	0.6056	**0.7980**
Euclidean	*s*	**0.5985**	0.2035	0.3037	**0.6863**	0.7178
	ws	0.4421	0.3826	0.4102	0.6301	0.7014
	wps	0.4774	**0.4498**	**0.4631**	0.6494	**0.7216**
JaccardMM	*s*	0.4019	0.7432	0.5216	0.5417	0.7329
	ws	**0.4290**	0.7612	0.5488	**0.5791**	0.7827
	wps	0.4290	**0.7853**	**0.5549**	0.5763	**0.7883**
JaccardAP	*s*	0.4019	**0.7432**	**0.5216**	0.5417	0.7329
	ws	0.6414	0.2142	0.3212	0.6955	0.7645
	wps	**0.6447**	0.2226	0.3310	**0.6974**	**0.7655**
$FW-KNN$		0.9639	0.0121	0.0240	0.0240	0.6858

$$Recall = \frac{TP}{TP+FN} \tag{11}$$

$$F-mesure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{12}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{13}$$

The ROC curve (receiver operating characteristic curve) is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. TPR is also known as *Recall* and FPR is the proportion of negative examples predicted incorrectly, both of them have a range of 0 to 1. Below are the formulas:

$$TPR = \frac{TP}{TP+FN} \tag{14}$$

$$FPR = \frac{FP}{FP+TN} \tag{15}$$

AUC (area under the curve) is the area under the ROC curve, which is an evaluation metric for measuring the performance of any classification model. The higher the *AUC*, the better the performance of the model at distinguishing between the two classes. The ideal *AUC* value is 1. Higher values of *Precision*, *Recall*, $F-measure$, *Accuracy*, and *AUC* indicate a better classification approach, and values range from 0 to 1.

Table 12. The *CNSE* of *Op* for different CC weighting approaches (the *CNSE* of original *Op* = 498459).

CC weighting approach		*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Similarity	Method					
Cosine	*s*	**480228**	**480228**	**481432**	1046768	1046503
	ws	**484860**	**484860**	**484136**	1017113	1016621
	wps	**483783**	**483781**	**483162**	1018375	1017881
Euclidean	*s*	**494051**	**494051**	**494050**	835673	835342
	ws	**492171**	**492170**	**492174**	982319	980898
	wps	**492282**	**492279**	**492294**	982271	980724
JaccardMM	*s*	**482058**	**482058**	**483533**	1014830	1014589
	ws	**483625**	**483625**	**483246**	1029240	1029037
	wps	**482309**	**482305**	**481862**	1032040	1031792
JaccardAP	*s*	**482058**	**482058**	**483533**	1014830	1014589
	ws	**493978**	**493978**	**493979**	829239	830110
	wps	**493980**	**493980**	**493983**	830595	831530
FW − KNN		499645	**497418**	**497408**	**480783**	**480785**

The empirical study is conducted on 443 versions of 13 subject programs listed in Table 9. To answer RQ1, three CC test case identification methods (*s*, *ws* and *wps*) were applied, and the CC weight of each passed test case was calculated. Table 11 presents the results of the classification evaluation metrics for CC test case identification, and bold data indicate the best results. The results are very close to each other. However, in all the similarity measures, the *wps* method is always the best in the three metrics. For example, in *Cosine* similarity, the *wps* method has the highest value in three measures: *Precision*, $F-Measure$ and, *AUC*. The last row of Table 11 presents the results for the $FW-KNN$ method [2]. The *Precision* value of $FW-KNN$ method is higher than our proposed methods. $FW-KNN$ calculates the most K similar test cases as k-nearest neighbors and uses the ratio of failing of k-nearest neighbors to compute the CC weight of a passed test case. Any passed test cases in k-nearest neighbors that are not proven CC, are considered passed. Therefore, the $FW-KNN$ method is very cautious in identifying the CC test, and as a result, the *Precision* value is too high and the *Recall* value is too low. Other evaluation metrics are also lower in the $FW-KNN$ method compared to the proposed methods.

4.4 Fault Localization Results

To evaluate the performance of a fault localization technique, we use three metrics: *CNSE*, *EXAM* and, *Accuracy* $(acc@n)$.

The Cumulative Number of Statements Examined (*CNSE* metric) [26] is calculated by Eq. (16). In this equation, *SP* is the set of all subject programs, $FV(pr)$ is the set of all faulty versions of subject program *pr* and $FL(v)$ is the number of statements that must be examined to locate all faults in the faulty version *v*. A lower *CNSE* value means a better fault localization technique.

$$CNSE = \sum_{pr \in SP} \sum_{v \in FV(pr)} FL(v) \tag{16}$$

The *EXAM* [49] is the percentage of code that has to be examined before a given bug

Table 13. The *CNSE* of *Ochiai* for different CC weighting approaches (The *CNSE* of original *Ochiai* = 603724).

CC weighting approach		*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Similarity	Method					
Cosine	*s*	604517	604802	604802	**479597**	1010265
	ws	**603386**	605575	605575	**419936**	957121
	wps	604904	606924	606924	**410017**	959225
Euclidean	*s*	**603477**	**603471**	**603471**	**385173**	637612
	ws	**602847**	**603011**	**603011**	**407058**	893008
	wps	**603312**	**603453**	**603453**	**405027**	888574
JaccardMM	*s*	604753	604756	604756	**464947**	961601
	ws	604515	606800	606800	**456835**	986622
	wps	604648	606737	606737	**442615**	991657
JaccardAP	*s*	604753	604756	604756	**464947**	961601
	ws	**603314**	**603502**	**603502**	**384942**	647555
	wps	**603322**	**603509**	**603509**	**385748**	652592
FW − KNN		606138	604061	604061	**565973**	**565886**

Table 14. The *CNSE* of *Tarantula* for different CC weighting approaches (The *CNSE* of original *Tarantula* = 609834).

CC weighting approach		*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Similarity	Method					
Cosine	*s*	**607967**	**607967**	**609686**	907988	**460957**
	ws	**608473**	**608473**	**609247**	935047	**455698**
	wps	**607809**	**607821**	**609126**	930190	**425754**
Euclidean	*s*	**609505**	**609505**	**609601**	773912	**358142**
	ws	**609284**	**609284**	**608138**	909577	**405591**
	wps	**609556**	**609556**	**608345**	912740	**370251**
JaccardMM	*s*	**608239**	**608239**	**609200**	905381	**459221**
	ws	**608437**	**608437**	**609068**	924791	**480421**
	wps	**607895**	**607895**	**609204**	920581	**455250**
JaccardAP	*s*	**608239**	**608239**	**609200**	905381	**459221**
	ws	**609492**	**609492**	**609642**	781365	**374567**
	wps	**609345**	**609345**	**609663**	788621	**372740**
FW − KNN		612452	612452	**609826**	**572253**	**573978**

Table 15. The *CNSE* of $DStar^3$ for different CC weighting approaches (The *CNSE* of original $DStar^3$ = 607459).

CC weighting approach		*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Similarity	Method					
Cosine	*s*	**606311**	610212	610212	918744	1045397
	ws	610466	613331	613331	902625	1014880
	wps	628961	631459	631459	906587	1016114
Euclidean	*s*	**607157**	**607192**	**607192**	734333	794864
	ws	**606762**	**606899**	**606899**	915440	968651
	wps	**606839**	**606980**	**606980**	908422	968391
JaccardMM	*s*	**606006**	608861	608861	887750	1011938
	ws	**606359**	609509	609509	914147	1027415
	wps	**606583**	609711	609711	911022	1030187
JaccardAP	*s*	**606006**	608861	608861	887750	1011938
	ws	**607212**	608738	608738	732952	790425
	wps	**607223**	608739	608739	735049	795939
FW − KNN		609951	607716	607716	**571346**	**572974**

is detected and is computed based on the ranking of all statements. The *EXAM* is defined in Eq. (17), and a lower EXAM value means a better fault localization technique. The average EXAM (AvgEXAM) of all faulty versions of programs in the dataset is calculated by Eq. (18). In this equation, $|pr|$ is the number of faulty versions of program pr.

$$EXAM = \frac{\#\,of\ statements\ examined\ to\ find\ all\ faults}{\#\,of\ all\ program\ statements} \times 100\% \tag{17}$$

$$AvgEXAM = \frac{\sum_{pr \in SP} \sum_{v \in FV(pr)} EXAM(v)}{\sum_{pr \in SP} |pr|} \tag{18}$$

Accuracy ($acc@n$) counts the number of successfully localized faults within the top-n positions of the ranked lists [50]. A higher $acc@n$ value means a better fault localization technique.

To answer RQ2 and RQ3, the *CNSE* results for proposed methods with different strategies, based on different similarity measures, for *Op*, *Ochiai*, *Tarantula* and *DStar*^{3} formulas are shown in Tables 12, 13, 14, and 15, respectively. Bold data show that according to the *CNSE* metric, the fault localization method performed better than the original formula. We can get the following observations for the four SBFL formulas:

1. *Op* formula: *Weighted*, $F-deleting$, and $F-cleansing$ strategies always improve SBFL.

2. *Ochiai* formula: $F-relabeling$ strategy always improves SBFL. *Euclidean* similarity always improves SBFL, except in the $F-exchanging$ strategy. Combining *JacardAP* similarity with *ws* or *wps* methods always improves SBFL except in the $F-exchanging$ strategy.

3. *Tarantula* formula: Except for the $F-relabeling$ strategy, an improvement is always observed in SBFL.

4. *DStar*^{3} formula: *Euclidean* similarity always improves SBFL except in the $F-relabeling$ and $F-exchanging$ strategies. *Weighting* strategy always improves SBFL except in the *Cosine* Similarity.

The $FW-KNN$ always outperforms the original SBFL formula in strategies $F-relabeling$ and $F-exchanging$.

To better answer RQ3, we summarize the data in Tables 12, 13, 14, and 15. Table 16 shows the percentage of times that combining a method and a similarity measure improves the original SBFL. For example, consider the value 45% at the intersection of the *s* row and the *Cosine* column in Table 16. The first rows of Tables 12, 13, 14, and 15 show the *CNSE* values for combining method *s* with five strategies using *Cosine* similarity. There are 20 values in these four tables and, 9 values are bold. Therefore, the percentage of times that combining the *s* method with the *Cosine* similarity improves the original SBFL is 45%. In Table 16, the values in the *Euclidean* column are the largest values. The results show that regardless of the method, *Euclidean* similarity performs better than other similarity measures. *JaccardAP* is also in second place. In each column, the maximum value can be seen in the *ws* row, which shows that this method has the best performance.

Table 16. The percentage of times that SBFL is improved by combining different methods with different similarity measures.

Method	*Cosine*	*Euclidean*	*JaccardMM*	*JaccardAP*
s	45%	70%	45%	45%
ws	45%	70%	45%	60%
wps	40%	70%	45%	60%

Table 17. The percentage of times that SBFL is improved by combining different methods with different strategies.

Method	*Weighted*	$F-deleting$	$F-cleansing$	$F-relabeling$	$F-exchanging$
s	81.25%	62.50%	62.50%	25.00%	25.00%
ws	87.50%	68.75%	68.75%	25.00%	25.00%
wps	81.25%	68.75%	68.75%	25.00%	25.00%

Table 17 shows the percentage of times that combining a method and a strategy improves the original SBFL. In Tables 12, 13, 14, and 15, *CNSE* values for the combination of *wps* method and *Weighted* strategy are shown at the intersection of *wps* rows and *Weighted* columns. There are 16 values in these four tables and 13 values are bold. Therefore, the percentage of times that combining the *wps* method with the *Weighted* strategy improves the original SBFL is 81.25%. In Table 17, this value is located at the intersection of the *wps* row and the *Weighted* column. In each row, the largest value belongs to the *Weighted* column, which means that the *Weighted* strategy performs better than the other strategies. In each column, the maximum value can be seen in the *ws* row, which shows that this method has the best performance. *wps* is also in second place.

Table 18 shows the percentage of times combining a similarity measure and a strategy improves the original SBFL. The largest value in each row belongs to the *Weighted* column. The largest value in each column belongs to the *Euclidean* row, and *JaccardAP* ranks second.

The following results can be summarized from the above discussion:

1. It seems that the *ws* and *wps* methods perform better than the *s* method, and the best result belongs to the *ws* method.
2. Overall, the *Weighted* strategy shows better results than the other four strategies.
3. It seems that *Euclidean* and *JacardAP* are better similarity measures to identify CC test cases.

Table 18. The percentage of times that SBFL is improved by combining different similarity measures with different strategies.

Similarity	*Weighting*	$F-deleting$	$F-cleansing$	$F-relabeling$	$F-exchanging$
Cosine	66.67%	50%	50%	25%	25%
Euclidean	100%	100%	100%	25%	25%
JaccardMM	75%	50.00%	50%	25%	25%
JaccardAP	91.67%	66.67%	66.67%	25%	25%

Table 19. Comparison of *Ews* method with *FW − KNN* based on *CNSE* metric.

Formula	Method	*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Op	*Ews*	**492171**	**492170**	**492174**	982319	980898
	FW − KNN	499645	497418	497408	**480783**	**480785**
Ochiai	*Ews*	**602847**	**603011**	**603011**	**407058**	893008
	FW − KNN	606138	604061	604061	565973	**565886**
Tarantula	*Ews*	**609284**	**609284**	**608138**	909577	**405591**
	FW − KNN	612452	612452	609826	**572253**	573978
DStar^3	*Ews*	**606762**	**606899**	**606899**	915440	968651
	FW − KNN	609951	607716	607716	**571346**	**572974**

Table 20. Comparison of *Ews* method with *FW − KNN* based on *AvgEXAM* metric.

Formula	Method	*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Op	*Ews*	**10.22%**	**10.22%**	**10.22%**	32.45%	32.42%
	FW − KNN	10.50%	10.31%	10.30%	**11.60%**	**11.60%**
Ochiai	*Ews*	**11.09%**	**11.05%**	**11.05%**	11.58%	27.68%
	FW − KNN	11.48%	11.29%	11.29%	**11.28%**	**11.16%**
Tarantula	*Ews*	**12.39%**	**12.39%**	12.79%	26.38%	13.90%
	FW − KNN	12.66%	12.66%	**12.47%**	**12.90%**	**12.67%**
DStar^3	*Ews*	**10.63%**	**10.63%**	**10.63%**	26.94%	30.18%
	FW − KNN	10.94%	10.75%	10.75%	**10.65%**	**10.69%**

Table 21. Comparison of *Ews* method with *FW − KNN* based on *acc*@1 metric.

Formula	Method	*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Op	*Ews*	9	9	9	0	0
	FW − KNN	9	9	9	**7**	**7**
Ochiai	*Ews*	7	7	7	**8**	0
	FW − KNN	7	7	7	7	**8**
Tarantula	*Ews*	8	8	8	8	8
	FW − KNN	8	8	8	8	8
DStar^3	*Ews*	8	8	8	2	0
	FW − KNN	8	8	8	**9**	**9**

Table 22. Comparison of *Ews* method with *FW − KNN* based on *acc*@3 metric.

Formula	Method	*Weighted*	*F − deleting*	*F − cleansing*	*F − relabeling*	*F − exchanging*
Op	*Ews*	**40**	**40**	**40**	0	0
	FW − KNN	37	37	38	**29**	**29**
Ochiai	*Ews*	**35**	**35**	**35**	**36**	5
	FW − KNN	32	32	32	30	**31**
Tarantula	*Ews*	**32**	**32**	**31**	21	**31**
	FW − KNN	29	29	30	**28**	28
DStar^3	*Ews*	**37**	**37**	**37**	9	0
	FW − KNN	34	34	34	**30**	**30**

Table 23. Comparison of Ews method with $FW-KNN$ based on $acc@5$ metric.

Formula	Method	*Weighted*	$F-deleting$	$F-cleansing$	$F-relabeling$	$F-exchanging$
Op	*Ews*	**62**	**62**	**62**	3	3
	$FW-KNN$	61	61	61	**45**	**45**
Ochiai	*Ews*	**56**	**56**	**56**	51	10
	$FW-KNN$	54	54	54	**52**	**53**
Tarantula	*Ews*	**51**	**51**	**51**	29	42
	$FW-KNN$	48	48	48	**43**	**44**
$DStar^3$	*Ews*	**57**	**57**	**57**	15	5
	$FW-KNN$	56	56	56	**54**	**54**

Table 24. Comparison of Ews method with $FW-KNN$ based on $acc@15$ metric.

Formula	Method	*Weighted*	$F-deleting$	$F-cleansing$	$F-relabeling$	$F-exchanging$
Op	*Ews*	**120**	**120**	**120**	17	17
	$FW-KNN$	119	119	119	**104**	**104**
Ochiai	*Ews*	**108**	**108**	**108**	100	31
	$FW-KNN$	107	107	107	**105**	**104**
Tarantula	*Ews*	**102**	**102**	100	56	89
	$FW-KNN$	101	101	100	**89**	**95**
$DStar^3$	*Ews*	**111**	**111**	**111**	34	26
	$FW-KNN$	109	109	109	**105**	**104**

According to the above discussion, the combination of *Euclidean* similarity with the *ws* method (*Ews*) calculates a better CC weight for a passed test case. To answer RQ4, an empirical study is conducted to compare CC test case identification effectiveness between *Ews* and $FW-KNN$. The results are shown in Tables 19, 20, 21, 22 23, and 24. Table 19 shows the comparison of the *CNSE* metric between *Ews* and $FW-KNN$. Better results are shown in bold. In all four formulas, *Ews* has better results than $FW-KNN$ in three strategies: *Weighted*, $F-deleting$, and $F-cleansing$. There are 20 comparisons in this table, 13 of which are bold in the *Ews* rows. This result shows that in 65% of the cases, *Ews* performed better than $FW-KNN$.

Table 20 shows the comparison of the *AvgEXAM* metric between *Ews* and $FW-KNN$. Better results are shown in bold. There are 20 comparisons in this table, 11 of which are bold in *Ews* rows. This result shows that in 55% of the cases, *Ews* performed better than $FW-KNN$.

Tables 21, 22, 23 and 24 present the results of $acc@1$, $acc@3$, $acc@5$ and $acc@15$ metrics, respectively. A higher value indicates a better fault localization technique. Better results are shown in bold. The results of $acc@1$ metric are slightly different from other results. Table 21 shows that our method and $FW-KNN$ have the same results in three strategies *Weighted*, $F-deleting$, and $F-cleansing$. In most cases, the $FW-KNN$ method performs better than our method in $F-relabeling$ and $F-ecchanging$ strategies. The results of Tables 22, 23, and 24 show that in 65%, 60% and 55% of cases, *Ews* performed better than $FW-KNN$ based on $acc@3$, $acc@5$ and $acc@15$ metrics, respectively.

The results of Tables 19, 20, 22, 23, and 24 have significant similarities. These results show that in terms of different evaluation metrics, the *Ews* method performs better with

strategies *Weighted*, $F-deleting$, and $F-cleansing$. Also, two strategies $F-relabeling$ and $F-exchanging$ give better results with $FW-KNN$.

5. THREATS TO VALIDITY

The main threats to the external validity of our approach, which address the ability to generalize our results and empirical findings, are related to the choice of subject programs and fault types. As explained in Section 4.2, we chose a set of 13 open source and popular subject programs widely used in the literature. These programs differ significantly in terms of size, functionality, number of faulty versions, types of faults, and number of test cases. All of the faults in the Siemens suite's programs are hand-seeded. Artificial faults may not represent the characteristics of real faults. Therefore, we used 6 programs from Defects4J, which contained real world open source programs with real faults. All this allows us better to generalize the findings and results of this paper. However, conducting experiments on more applications can better evaluate the effectiveness of our methods.

The threat to construct validity is related to the measurements of the results. We predict a continuous value as the CC weight of a passed test case, so we treat the CC test case identification problem as a regression problem. In terms of CC test case identification accuracy, we use five metrics to evaluate the effectiveness of the proposed methods. The critical question is whether improving which of these metrics has a more significant effect on improving the accuracy of fault localization. We intend to answer this question in our future work. In terms of fault localization accuracy, we use the $CNSE$, $EXAM$, and $acc@n$ metrics to compare the proposed methods with $FW-KNN$. $CNSE$, $EXAM$, and $acc@n$ may not be the best metrics to compare the effectiveness of fault localization techniques. In future work, we plan to use more metrics to measure the experimental results.

The threats to the internal validity of our approach are as following:

- A test oracle is available, and test case execution results can be marked as passed or failed.
- The faults must be deterministic, meaning that the execution results (success or failure) of the test cases are not affected by the run-time environment.

These two assumptions have been accepted in many studies and the threats appear to be limited.

6. CONCLUSION AND FUTURE WORKS

Spectrum-based fault localization (SBFL) techniques use test results and the coverage information of test executions to identify the faulty elements of a program. Coincidentally correct test cases are those that execute faulty statements but do not cause failures. Such test cases reduce the effectiveness of SBFL techniques. It is essential to identify CC test cases and eliminate their destructive effects. A proven CC test case is a passed test case that can be proved as CC, but other passed test cases have the possibility to be CC.

To identify the CC test case, we assign a weight to each passed test case and hope that a greater weight is calculated for the actual CC. The weight is calculated by comparing the

coverage information of a failed and passed test case using similarity measures. We also proposed two new similarity measures (*JaccardMM* and *JaccardAP*). Then we improved the method by calculating weights for program elements and using the similarity measures in a way that can identify proven CC. Finally, to reduce the negative impact of CCs, we proposed the *Weighted* strategy to manipulate the CC weights of passed test cases in SBFL.

A comprehensive empirical study is conducted on 443 faulty versions of 13 subject programs, and the results show that: 1) In most cases, combining similarity-based CC test case identification and the *Weighted* strategy can improve SBFL. 2) Proposed similarity measure (*JacardAP*) and *Euclidean* similarity are two proper similarity measures to identify CC test cases. 3) The proposed CC test case manipulation strategy (*Weighted* strategy) seems to give better results compared to other strategies.

Research on test case generation and selection techniques has shown that using certain test cases can lead to more effective fault localization than others [33, 51]. Therefore, assigning weights to passed test cases can improve SBFL. In our future work, we are going to investigate the effect of integrating our proposed CC test case weighting methods with other passed test case weighting methods on the accuracy of the SBFL.

REFERENCES

1. P. Cao, Z. Dong, K. Liu, and K.-Y. Cai, ''Quantitative effects of software testing on reliability improvement in the presence of imperfect debugging,'' *Information Sciences*, Vol. 218, 2013, pp. 119-132.
2. Y. Liu, M. Li, Y. Wu, and Z. Li, ''A weighted fuzzy classification approach to identify and manipulate coincidental correct test cases for fault localization,'' *Journal of Systems and Software*, Vol. 151, 2019, pp. 20-37.
3. W. E. Wong, T. Wei, Y. Qi, and L. Zhao, ''A crosstab-based statistical method for effective fault localization,'' in *Proceedings of the 1st International Conference on Software Testing, Verification and Validation*, 2008, pp. 42-51.
4. X. Xie, T. Y. Chen, F. C. Kuo, and B. Xu, ''A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization,'' *ACM Transactions on Software Engineering and Methodology*, Vol. 22, 2013, pp. 1-40.
5. W. Masri, R. Abou-Assi, M. El-Ghali, and N. Al-Fatairi, ''An empirical study of the factors that reduce the effectiveness of coverage-based fault localization,'' in *Proceedings of the 2nd International Workshop on Defects in Large Software Systems, Held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2009, pp. 1-5.
6. W. Masri and R. A. Assi, ''Prevalence of coincidental correctness and mitigation of its impact on fault localization,'' *ACM Transactions on Software Engineering and Methodology*, Vol. 23, 2014, pp. 1-28.
7. Y. Li and C. Liu, ''Identifying coincidental correctness in fault localization via cluster analysis,'' *Journal of Software Engineering*, Vol. 8, 2014, pp. 328-344.
8. F. Feyzi and S. Parsa, ''A program slicing-based method for effective detection of coincidentally correct test cases,'' *Computing*, Vol. 100, 2018, pp. 927-969.

9. Y. Miao, Z. Chen, S. Li, Z. Zhao, and Y. Zhou, ''Identifying coincidental correctness for fault localization by clustering test cases,'' in *Seke*, 2012, pp. 267-272.
10. X. Yang, M. Liu, M. Cao, L. Zhao, and L. Wang, ''Regression identification of coincidental correctness via weighted clustering,'' in *Proceedings of IEEE 39th Annual Computer Software and Applications Conference*, 2015, pp. 115-120.
11. F. Feyzi and S. Parsa, ''Fpa-fl: Incorporating static fault-proneness analysis into statistical fault localization,'' *Journal of Systems and Software*, Vol. 136, 2018, pp. 39-58.
12. Y. Miao, Z. Chen, S. Li, Z. Zhao, and Y. Zhou, ''A clustering-based strategy to identify coincidental correctness in fault localization,'' *International Journal of Software Engineering and Knowledge Engineering*, Vol. 23, 2013, pp. 721-741.
13. Y. Li and C. Liu, ''Using cluster analysis to identify coincidental correctness in fault localization,'' in *Proceedings of the 4th International Conference on Computational and Information Sciences*, 2012, pp. 357-360.
14. Z. Li, M. Li, Y. Liu, and J. Geng, ''Identify coincidental correct test cases based on fuzzy classification,'' in *Proceedings of International Conference on Software Analysis, Testing and Evolution*, 2016, pp. 72-77.
15. F. Feyzi and S. Parsa, ''Kernel-based detection of coincidentally correct test cases to improve fault localisation effectiveness,'' *International Journal of Applied Pattern Recognition*, Vol. 5, 2018, pp. 119-136.
16. X. Xue, Y. Pang, and A. S. Namin, ''Trimming test suites with coincidentally correct test cases for enhancing fault localizations,'' in *Proceedings of IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 239-244.
17. L. Weishi and X. Mao, ''Alleviating the impact of coincidental correctness on the effectiveness of sfl by clustering test cases,'' in *Proceedings of International Symposium on Theoretical Aspects of Software Engineering*, 2014, pp. 66-69.
18. A. Sabbaghi, M. R. Keyvanpour, and S. Parsa, ''FCCI: A fuzzy expert system for identifying coincidental correct test cases,'' *Journal of Systems and Software*, Vol. 168, 2020, p. 110635.
19. L. Naish, H. J. Lee, and K. Ramamohanarao, ''Spectral debugging with weights and incremental ranking,'' in *Proceedings of the 16th Asia-Pacific Software Engineering Conference*, 2009, pp. 168-175.
20. V. Dallmeier, C. Lindig, and A. Zeller, ''Lightweight bug localization with ample,'' in *Proceedings of the 6th International Symposium on Automated Analysis-Driven Debugging*, 2005, pp. 99-104.
21. W. Masri, ''Fault localization based on information flow coverage,'' *Software Testing, Verification and Reliability*, Vol. 20, 2009, pp. 121-147.
22. T. Shu, T. Ye, Z. Ding, and J. Xia, ''Fault localization based on statement frequency,'' *Information Sciences*, Vol. 360, 2016, pp. 43-56.
23. C. Yilmaz, A. Paradkar, and C. Williams, ''Time will tell: fault localization using time spectra,'' in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 81-90.
24. J. Jones, M. Harrold, and J. Stasko, ''Visualization for fault localization,'' in *issu body-- --*, 2001, pp. 71-75.

25. R. Abreu, P. Zoeteweij, and A. J. van Gemund, ''An evaluation of similarity coefficients for software fault localization,'' in *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, 2006, pp. 39-46.
26. W. E. Wong, V. Debroy, R. Gao, and Y. Li, ''The dstar method for effective software fault localization,'' *IEEE Transactions on Reliability*, Vol. 63, 2014, pp. 290-308.
27. L. Naish, H. J. Lee, and K. Ramamohanarao, ''A model for spectra-based software diagnosis,'' *ACM Transactions on Software Engineering and Methodology*, Vol. 20, 2011, pp. 1-32.
28. A. Ajibode, T. Shu, K. Said, and Z. Ding, ''A fault localization method based on metrics combination,'' *Mathematics*, Vol. 10, 2022, p. 2425.
29. J. Voas, ''Pie: a dynamic failure-based technique,'' *IEEE Transactions on Software Engineering*, Vol. 18, 1992, pp. 717-727.
30. R. Abou Assi, C. Trad, M. Maalouf, and W. Masri, ''Coincidental correctness in the defects4j benchmark,'' *Software Testing Verification and Reliability*, Vol. 29, 2019, p. e1696.
31. R. M. Hierons, ''Avoiding coincidental correctness in boundary value analysis,'' *ACM Transactions on Software Engineering and Methodology*, Vol. 15, 2006, pp. 227-241.
32. W. Masri and R. A. Assi, ''Cleansing test suites from coincidental correctness to enhance fault-localization,'' in *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation*, 2010, pp. 165-174.
33. A. Bandyopadhyay and S. Ghosh, ''Proximity based weighting of test cases to improve spectrum based fault localization,'' in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 420-423.
34. A. Bandyopadhyay, ''Mitigating the effect of coincidental correctness in spectrum based fault localization,'' in *Proceedings of IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 479-482.
35. X. Zhou, H. Wang, and J. Zhao, ''A fault-localization approach based on the coincidental correctness probability,'' in *Proceedings of IEEE International Conference on Software Quality, Reliability and Security, QRS*, 2015, pp. 292-297.
36. X. Wang, S. Cheung, W. Chan, and Z. Zhang, ''Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization,'' in *Proceedings of IEEE 31st International Conference on Software Engineering*, 2009, pp. 45-55.
37. M. Papadakis and Y. Le Traon, ''Metallaxis-fl: mutation-based fault localization,'' *Software Testing, Verification and Reliability*, Vol. 25, 2015, pp. 605-628.
38. M. Papadakis and Y. Le Traon, ''Effective fault localization via mutation analysis,'' in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 1293-1300.
39. A. Singhal *et al.*, ''Modern information retrieval: A brief overview,'' *IEEE Data Engineering Bulletin*, Vol. 24, 2001, pp. 35-43.
40. M. M. Deza and E. Deza, *Encyclopedia of Distances*, Springer Berlin Heidelberg, 2013.
41. P. Jaccard, ''The distribution of the flora in the alpine zone. 1,'' *New Phytologist*, Vol. 11, 1912, pp. 37-50.
42. M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, ''Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria,'' in *Proceedings of the 16th International Conference on Software Engineering*, 1994, pp. 191-200.

43. R. Just, D. Jalali, and M. D. Ernst, ''Defects4j: a database of existing faults to enable controlled testing studies for java programs,'' in *Proceedings of International Symposium on Software Testing and Analysis*, 2014, pp. 437-440.
44. F. Feyzi and S. Parsa, ''Inforence: effective fault localization based on information-theoretic analysis and statistical causal inference,'' *Frontiers of Computer Science*, Vol. 13, 2019, pp. 735-759.
45. R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. van Gemund, ''A practical evaluation of spectrum-based fault localization,'' *Journal of Systems and Software*, Vol. 82, 2009, pp. 1780-1792.
46. J. A. Jones and M. J. Harrold, ''Empirical evaluation of the tarantula automatic fault-localization technique,'' in *Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering*, 2005, pp. 273-282.
47. Z. Hui, ''Utilization of dependence and weight to improve fault localization method of regression test cases,'' *International Journal of Software Engineering and Knowledge Engineering*, Vol. 27, 2017, pp. 423-447.
48. S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, ''Evaluating and improving fault localization,'' in *Proceedings of IEEE/ACM 39th International Conference on Software Engineering*, 2017, pp. 609-620.
49. W. E. Wong, V. Debroy, and B. Choi, ''A family of code coverage-based heuristics for effective fault localization,'' *Journal of Systems and Software*, Vol. 83, 2010, pp. 188-208.
50. J. Sohn and S. Yoo, ''Fluccs: using code and change metrics to improve fault localization,'' in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017, pp. 273-283.
51. A. Bandyopadhyay, ''Improving spectrum-based fault localization using proximity-based weighting of test cases,'' in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 660-664.

Mohammad Mahdi Estesnaei received his BSc and MSc in Software Engineering from Ferdowsi University of Mashhad. Currently, he is pursuing Ph.D. in Software Engineering at Ferdowsi University of Mashhad, Iran. His research interests include software engineering, software testing, and software debugging.

Saeed Araban received his Ph.D. in Software Engineering from the University of Melbourne, Australia. He is currently an Assistant Professor in the Department of Computer Engineering at Ferdowsi University of Mashhad (FUM), Iran. His research interests include software quality assurance, empirical software engineering and service-oriented enterprise architecture.

Ahad Harati was born in Mashhad, Iran. He received his BSc. in Computer Engineering from Amirkabir University of Technology, Tehran, Iran in 2000. He obtained his MSc. and Ph.D. degrees in Artificial Intelligence and Robotics from University of Tehran, Iran (2002) and Swiss Federal Institute of Technology in Zurich (ETHZ) in 2008, respectively. He is currently an Associate Professor in Computer Engineering Department of Ferdowsi University of Mashhad, Iran. His research interests include robot perception, machine learning and autonomous vision-based navigation .