Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 14, No. 4, 2026, 663-675.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

FSBFL: A Fuzzy Expert System for Improving Spectrum-based Fault Localization

Mohammad Mahdi Estesnaei and Saeed Araban*

Department of engineering, Ferdowsi University of Mashhad, Mashhad, Iran.

Article Info

Article History:
Received 21 April 2026
Revised 06 June 2026
Accepted 08 July 2026

DOI:10.22044/jadm.2026.17502.2890

Keywords:
Software Testing, Spectrum-based Fault Localization, Coincidentally Correctness, Fuzzy Expert System.

Corresponding author: araban@um.ac.ir (S. Araban).

Abstract

Spectrum-based fault localization (SBFL) is a widely used technique that utilizes coverage data and test outcomes to calculate a suspiciousness score for each program statement. The fundamental hypothesis of SBFL is that a statement covered by more failed test cases and fewer passed test cases is more likely to be faulty. However, the effectiveness of SBFL is hindered by coincidental correctness, which occurs when a fault is executed but no failure is detected. Additionally, traditional SBFL methods assign equal weight to all failed tests, despite some failed tests containing more valuable information. This study aims to enhance SBFL performance by employing a fuzzy expert system to address these challenges. Thirteen open-source subject programs were used to evaluate the efficiency of the proposed FSBFL method. Experimental results, assessed using four key metrics, demonstrate that FSBFL outperforms popular spectrum-based fault localization techniques.

1. Introduction

During software debugging, it is essential to detect potential error codes and amend them promptly to mitigate or prevent the risks and financial losses associated with software defects [1]. A significant portion of the debugging process focuses on identifying the source of faults [2]. Fault identification is a laborious, time-consuming, and costly part of the software debugging process [3]. Consequently, numerous software fault localization methods have been developed to replace laborious manual tasks and reduce the time required.

Spectrum-Based Fault Localization (SBFL) is a debugging technique that ranks program elements (such as statements, branches, or methods) by their likelihood of containing a bug, based on which code is executed by passing and failing test cases. Original SBFL methods use risk evaluation formulas (e.g., Ochiai [4], Tarantula [5], Jaccard [6], DStar [7], or Op [8]) to calculate the suspiciousness score of each program element. These formulas analyze the number of passing and failing test cases that execute a program element, enabling the elements to be ranked according to their likelihood of being faulty. The number of passed test cases is the total count of test cases that execute successfully and produce the expected results. The number of failed test cases is the total count of test cases that do not produce the expected results. The effectiveness of SBFL techniques can be influenced by many factors. One such factor is coincidental correctness (CC), which refers to a situation where a fault has been executed, yet no failure is observed. Original SBFL cannot effectively handle CC test cases because it assumes that all passing test cases represent correct program behavior. A substantial body of research focuses on identifying CCs and mitigating their negative impacts [9-12]. In addition to spectrum-based factors, static factors extracted from source code are used in designing fuzzy rules for identifying

CCs [13]. Since these methods model CC test case identification as a binary classification problem, they are susceptible to false positives, where genuinely passing test cases may be incorrectly classified as CC. To address this, approaches such as those presented in [1, 14] introduce weighted methods for identifying CC test cases, with three versions of Ochiai [14] and DStar3 [1] designed to mitigate the negative impact of CC test cases. Original SBFL methods assign equal weight to all failed test cases, although some failed test cases contain more information [15]. Empirical evaluations of fault localization methods indicate that the selection of test cases can significantly influence their effectiveness [16, 17]. Empirical studies have shown that no single SBFL formula consistently outperforms all others [6, 18]. There are numerous SBFL formulas because the relationships between variables are not well understood, so they offer different mathematical models of suspiciousness. A fuzzy approach is well suited to overcoming the limitations of original SBFL because it effectively models uncertainty and partial truth. Rather than treating all failed test cases equally or identifying coincidentally correct test cases through rigid binary classification, fuzzy logic assigns continuous membership degrees and adaptive weights based on the characteristics of each test case. This enables more accurate representation of diagnostic information, reduces negative impact of false positives in CC identification, and improves the effectiveness and robustness of fault localization.

Based on these considerations, this paper identifies several factors and develops a fuzzy expert system that calculates the suspiciousness scores of statements using established fuzzy rules. The primary contributions of this research are outlined as follows:

- We introduce a method that assigns greater weights to more informative failed tests. A failed test is more informative if it covers fewer statements. In this case, the probability of each covered statement being faulty increases.
- A method is proposed to calculate the weights of passed tests, reflecting their likelihood of being CCs. The similarity values of a passed test case with the failed ones are integrated to compute the CC weight of the passed test case.

The remainder of this paper is structured as follows: Section 2 provides an overview of SBFL techniques and coincidental correctness. Section 3 outlines the specifics of our proposed fuzzy spectrum-based fault localization (FSBFL) approach. The design of the experiment and the analysis of the results are presented in Section 4. Section 5 address potential validity threats related to our proposed method. Finally, Section 6 offers concluding remarks.

1. Background and Related Works

In this section, we provide an overview of SBFL techniques and coincidentally correct test cases. Additionally, a motivating example is included to clearly demonstrate how SBFL techniques operate.

2.1. Spectrum-based Fault Localization

Suppose program under test contains n statements, which we call $st_1, st_2, \ldots, st_n$. Additionally, a test suite T contains m test cases $t_1, t_2, \ldots, t_m$. $T = T_F \cup T_P$ is a set of test executions where T_F represents the set of test executions that failed, while T_P represents the set of test executions that passed. Passed test cases are executions of a program whose outputs are expected, whereas failed test cases are executions of a program whose outputs are unexpected. The statement coverage of a test case t, can be represented as an n-dimensional binary vector $E_t=<e_1, e_2, \ldots, e_j, \ldots, e_n>$.

As shown in Figure 1, if the test case t executes the statement st_j, the value of e_j is 1, otherwise, it is 0. The result of the test case is indicated by P and F, which mean passed and failed, respectively. For each statement, four integer parameters (a_{ep}, a_{np}, a_{ef} and a_{nf}) are determined using coverage information and the outcomes of test cases. The initial portion of the subscript shows if the related statement was executed *(e)* or not *(n)*, while the latter portion indicates whether the test passed *(p)* or failed *(f)*. For example, the a_{ef} of a statement refers to the number of failed test cases that have executed that statement.

As shown in Table 1, most SBFL techniques, such as Tarantula [5], Ochiai [4], DStar* [7], and Op [8], apply a formula to determine a suspiciousness score for every program statement. The sample program in Figure 1 has 12 statements, two of which st_3 and st_6 are faulty. The suspiciousness scores and the ranking of statements for the example in Figure 1, calculated using the Ochiai technique, are presented in columns six and seven of Table 2 respectively. The ranking is calculated in the worst case, which means that the faulty statement is assumed to be the last statement checked among statements with the same suspiciousness score.

	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=r*2; //correct: r=r+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:Faild)	P	P	P	P	P	P	P	P	F	F

Figure 1. A sample program with two seeded faults.

1.1. Coincidental Correctness

Based on the PIE[1] model [19], for a failure to occur, the following three conditions must be satisfied: CE = the faulty statement was executed; CI = the program has transitioned into an infectious state; and CP = the infection has propagated to the output [20]. Coincidental correctness occurs when CE is satisfied but either CI is not met or CI is satisfied but CP is not met. Research has demonstrated that coincidental correctness occurs frequently [20–22]. In the example shown in Figure 1, the six passed test cases t3 to t8 are coincidentally correct. For instance, test case t4 covers the faulty statement st3, but it does not result in an infectious state. Therefore, its actual result is equal to the expected result and is considered a passed test case.

Table 1. Four SBFL techniques.

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

Table 2. The suspiciousness scores and ranking of statements for the sample program using the Ochiai method.

st#	a_{ep}	a_{np}	a_{ef}	a_{nf}	*Susp*	*Rank*
st_1	8	0	2	0	0.45	7
st_2	8	0	2	0	0.45	7
st_3	6	2	2	0	0.50	3
st_4	6	2	2	0	0.50	3
st_4	1	7	1	1	0.50	3
st_6	5	3	1	1	0.29	9
st_7	3	5	0	2	0.00	12
st_8	2	6	1	1	0.41	8
st_9	2	6	0	2	0.00	12
st_{10}	8	0	2	0	0.45	7
st_{11}	6	2	1	1	0.27	10
st_{12}	8	0	2	0	0.45	7

1.2. Related Works

Many studies first identify CC test cases and then mitigate their adverse effects using two strategies: cleansing or relabelling. The cleansing strategy involves removing the identified CC test cases from the test suite, while the relabelling strategy involves relabelling those test cases from passed to failed [12,23]. In the literature, clustering algorithms (such as k-means [9, 12, 23–25]), supervised learning algorithms (such as KNN [1, 14] and SVM [10, 11]), and fuzzy methods [13] are employed for identifying CC test cases. Therefore, in most studies (excluding [1, 14]), the identification of CCs has been considered a binary classification problem, and the accuracy of relabelling strategies can be significantly affected by the false positive rate. Such a classification method may involve ambiguity, which a Fuzzy Expert System is capable of managing in uncertain

[1] Propagation-Infection-Execution

situations. Therefore, instead of binary classification, we designed fuzzy rules with the concept of partial membership to reduce the negative impact of CCs.

2. The Proposed Method

The proposed method considers various factors for statements, based on test case factors. Then, a fuzzy expert system is designed to estimate the suspiciousness score of statements according to the defined fuzzy rules. Below, the suggested factors and the framework of the developed fuzzy expert system are explained.

2.1. Test Case Factors

We introduce a similarity measure based on weighted statement coverage to quantify the similarity between two test cases (failed and passed test cases). Then, the similarity values of a passed test case with the failed ones are integrated to calculate the CC weight of the passed test case. The CC weight of a passed test case is in the range [0,1]. Suppose $SP(fm)=<sp_1,sp_2,\dots,sp_j,\dots,sp_n>$ is a real vector whose components are the suspiciousness scores of program statements calculated by the *fm* formula. *fm* is a spectrum-based formula as shown in Table 1, and sp_j is the suspiciousness score of the statement st_j. (1) determines the weight of program statements using the failed test case t_f and the *fm* formula ($W(t_f|fm)$ is an n-dimensional real vector). In this equation, the operator × multiplies each pair of corresponding elements from the two vectors, producing a vector as the result. In the denominator of the fraction, the 1-norm (also called the L1 norm) of the vector is calculated. Finally, the vector components are divided by the 1-norm value. The weight of the statements not covered by the test case t_f is zero, while the total weight of the covered statements is one. Also, statement with a higher suspicious score will be given greater weight. Equation (2) calculates the CC weight of the passed test case t_p using the *fm* formula. The purpose of (2) is to assign more weight to a passed test that is more likely to be a CC.

$$W(t_f \mid fm)=\frac{E_{t_f}\times SP(fm)}{\| E_{t_f}\times SP(fm)\|_1} \tag{1}$$

$$w_{CC}(t_p \mid fm)=\max_{t_f\in T_F}\| W(t_f \mid fm)\times E_{t_p}\|_1 \tag{2}$$

Some failed tests offer more information than others [15]. If a failed test covers only one statement, it allows us to confidently conclude that the covered statement is faulty. Therefore, the information provided by this test is more valuable than that from a test covering multiple statements. Equation (3) calculates the weight of the failed test case t_f within the range of [0,1]. In this equation, n is the number of program statements, and the test weight is inversely proportional to the number of statements it covers.

$$w_F(t_f)=1-\frac{\| E_{t_f}\|_1-1}{n-1} \tag{3}$$

2.2. The Statement Suspiciousness Factors

The original SBFL techniques use the following two hypotheses [14]: a) a statement covered by more failed test cases has a higher likelihood of being faulty; b) a statement covered by more passed test cases has a lower likelihood of being faulty. Accordingly, we calculate two suspicious factors, EF and EP, for each statement according to (4). In this equation, EF is the ratio of failed test cases that executed the statement to the total number of failed test cases. EP is the ratio of passed test cases that executed the statement to the total number of passed test cases. EF and EP are in the range [0,1]. According to SBFL hypotheses, the higher the EF and the lower the EP, the more likely the statement is to be faulty.

$$EF=\frac{a_{ef}}{a_{ef}+a_{nf}},\ EP=\frac{a_{ep}}{a_{ep}+a_{np}} \tag{4}$$

The presence of CC test cases negatively impacts SBFL techniques by undermining two fundamental assumptions of SBFL. To mitigate the negative effect of CCs, we calculate, for each statement, the average CC weight of passed tests (ACCWP) covering that statement. A high ACCWP indicates a greater likelihood that the statement is incorrect, while a low value suggests it is less likely to be faulty. Statements covered by more informative failed tests are more likely to be faulty. Equation (3) assigns greater weight to a failed test that contains more information. Therefore, for each statement, we calculate the average weight of the failed tests (AWF) covering that statement. A high AWF value indicates that the statement is more likely to be faulty, and vice versa.

2.3. Case Study

In this section, we explain the above equations using the examples presented in Table 3. Columns two, three, and four show the statement coverage vectors (as illustrated in Figure 1) for tests t1, t9, and t10, respectively. The SP(Ochiai) column indicates the suspiciousness score of statements based on the Ochiai method. Column six displays the product of the corresponding values in columns three and five. The value in column six is the product of the corresponding values from columns three and five, with its 1-norm calculated in the last

row. By dividing the values in column six by the 1-norm, the values in column seven are obtained. Similarly, the values in the eighth column are calculated. Values in columns nine and ten are obtained by multiplying the values in the specified columns of their first row. According to (2), the maximum value in the last row of columns nine and ten (i.e., 0.6) represents the CC weight of the passed test case t1. Finally, according to (3), the weight of the failed test t9 is equal to 1 - (8-1) / (12-1) = 0.36. EF and EP values are extracted from Table 2. For example, the EF and EP values for statement st11 are 1/2 = 0.5 and 6/8 = 0.75, respectively.

Table 3. An example to explain equations.

St#	E_{t1}	E_{t9}	E_{t10}	SP(Ochiai)	E_{t9}×SP(Ochiai)	$W(t_9\|Ochiai)$	$W(t_{10}\|Ochiai)$	$W(t_9\|Ochiai)\times E_{t1}$	$W(t_{10}\|Ochiai)\times E_{t1}$
st_1	1	1	1	0.45	0.45	0.13	0.13	0.13	0.13
st_2	1	1	1	0.45	0.45	0.13	0.13	0.13	0.13
st_3	0	1	1	0.50	0.5	0.14	0.14	0	0
st_4	0	1	1	0.50	0.5	0.14	0.14	0	0
st_5	0	0	1	0.50	0	0	0.14	0	0
st_6	0	1	0	0.29	0.29	0.08	0	0	0
st_7	0	0	0	0.00	0	0	0	0	0
st_8	0	1	0	0.41	0.41	0.12	0	0	0
st_9	1	0	0	0.00	0	0	0	0	0
st_{10}	1	1	1	0.45	0.45	0.13	0.13	0.13	0.13
st_{11}	1	0	1	0.27	0	0	0.08	0	0.08
st_{12}	1	1	1	0.45	0.45	0.13	0.13	0.13	0.13
1-norm		8	8		3.5			0.52	0.6

2.4. The Proposed Fuzzy Expert System

In Section 2.2, we presented several factors that can be used to estimate the probability that a statement is faulty. A test that passed but is very similar to a failed test is more likely to be linked to a higher chance of CC. However, it is challenging to give a precise definition of what "high similarity" is [13]. This kind of ambiguity and uncertainty can be examined using fuzzy set theory. [26]. The primary subsystems of a fuzzy expert system are fuzzification, the knowledge base, inference, and defuzzification. [27]. Fuzzification converts crisp (numeric) input data into fuzzy values using membership functions. The values of the four inputs of the system, namely *EF*, *EP*, *ACCWP*, and *AWF*, are represented through two linguistic variables (Low *(L)* and High *(H)*) utilizing a Gaussian membership function $\mu_A(x) = e^{\frac{-(x-C)^2}{2\sigma^2}}$, where C denotes the center and σ denotes the spread of the function. We select σ = *0.425* and C = *0* for the *L* membership function, and C = *1* for the H membership function. The membership functions are provided as follows:

$$\mu_L(x) = e^{-\frac{x^2}{0.361}} \quad \mu_H(x) = e^{-\frac{(x-1)^2}{0.361}} \tag{5}$$

Five membership functions were also defined for the output (*SUSP*: Suspiciousness score) representing Very Low (*VL*), Low (*L*) Medium (*M*), High (*H*), and Very High (*VH*) linguistic classes, which are defined using parameters σ = 0.1062 and C = 0, 0.25, 0.5, 0.75, 1 respectively. The membership functions are given in the following:

$$\mu_{VL}(x) = e^{-\frac{y^2}{0.023}} \quad \mu_L(y) = e^{-\frac{(y-0.25)^2}{0.023}}$$
$$\mu_M(x) = e^{-\frac{(y-0.5)^2}{0.023}} \quad \mu_H(x) = e^{-\frac{(y-0.75)^2}{0.023}} \tag{6}$$
$$\mu_{VH}(x) = e^{-\frac{(y-1)^2}{0.023}}$$

In this research, we employ a rule-based knowledge base to illustrate the relationships between various statement suspiciousness factors and the likelihood of statements being faulty. The fuzzy rule base represents the collection of fuzzy IF-THEN rules [28]. In our fuzzy knowledge-based system, there are 11 fuzzy rules, which are presented as follows:

1. if ACCWP is H then SUSP is VH
2. if AWF is H then SUSP is H
3. if EF is H then SUSP is H
4. if EF is L and ACCWP is H then SUSP is M
5. if EF is H and EP is L then SUSP is VH
6. if EF is L and AWF is H then SUSP is M
7. if EF is H and EP is H and ACCWP is H then SUSP is VH
8. if EF is L and EP is L and ACCWP is L then SUSP is L
9. if EF is L and EP is H and ACCWP is L then SUSP is VL
10. if EF is H and EP is H and ACCWP is L then SUSP is H
11. if EF is L and EP is H and ACCW is L and AWF is H then SUSP is L

Fuzzy inference involves converting input variables into output variables through fuzzy logic and includes five steps: fuzzifying the input variables; applying the fuzzy operators (such as AND or OR) in the rule's antecedent; deriving the implication from the antecedent to the consequent; combining the consequents from all the rules; and finally, defuzzifying the result. The Mamdani fuzzy inference system [29, 30] and the Takagi-Sugeno inference system [31] are two widely used types of fuzzy inference systems. We utilized the Mamdani fuzzy inference system, which employs fuzzy sets for both input and output variables. Our system evaluates the mentioned fuzzy rules where the AND operator, IMPLICATION method and AGGREGATION process are $\min(\mu_A(x), \mu_B(x))$, $\min(\mu_A(x), \mu_B(x))$ and $\max(\mu_A(x), \mu_B(x))$ respectively.

Many different defuzzification techniques have been proposed in the literature. Among these methods, the center of gravity (COG) defuzzification technique is widely used [32, 33] and is also the most popular and intuitively appealing defuzzification approach [27]. We employed COG in the defuzzification phase, which provides a crisp value based on the center of gravity of the fuzzy set.

2.5. Computational Cost

As mentioned in section 2.1, n is the number of statements and m is the number of test cases, so the program spectrum is an n,m binary matrix. It is obvious that the time complexity of spectrum-based fault localization is $O(nm)$. Suppose m_1 represents the number of failed test cases, and m_2 represents the number of passed test cases. The time complexity of each phase of the proposed algorithm is as follows:

- *SP(fm)* is calculated once and has a time complexity of $O(nm)$.
- The time complexity of (1) is $O(n)$, and it is computed m_1 times for m_1 failed test cases, resulting in a total complexity of $O(nm_1)$. The result is an $n \times m_1$ table used in (2).
- The time complexity of (2) is $O(nm_1)$ and is calculated m_2 times for m_2 passed test cases, resulting in a total complexity of $O(nm_1m_2)$. The result is an m_2-dimensional real vector used in the *ACCWP* calculation.
- The time complexity of (3) is $O(n)$, and it calculated m_1 times for m_1 failed test cases, resulting in a total complexity of $O(nm_1)$. The result is an m_1-dimensional real vector used in the *AWF* calculation.
- The time complexity of calculating *ACCWP, AWF, EF and EP* for each row of the program spectrum is $O(m)$, where $m = m_1 + m_2$. Therefore, the total time complexity of applying 11 fuzzy rules to all rows of the program spectrum is $O(nm)$. (Since the evaluation of 11 fuzzy rules requires a constant time, we consider it as one step.

Ignoring the lower-order complexities of terms mentioned, the result will be $O(nm_1m_2)$. In the worst case, if we assume the growth rates of m_1 and m_2 are proportional to m, the time complexity will be $O(nm^2)$. However, in most cases, the value of m_1 (the number of failed tests) is small, and the growth rate of m_2 is proportional to m, resulting in a time complexity of $O(nm)$.

3. Experiment and Results Analysis

3.1. Research Questions

The purpose of FSBFL is to provide a better measure for calculating the suspiciousness score of statements compared to the original SBFL. The empirical study is conducted to investigate the following research questions:

- RQ1) Does the FSBFL method improve the performance of SBFL techniques?
- RQ2) Which SBFL formula performs better when using the FSBFL method?
- RQ3) Compared to other methods such as FES (fuzzy expert system) [34] and CBS (consensus-based strategy) [35], is FSBFL more effective in terms of fault localization accuracy?

3.2. Subject Programs

We utilized 13 well-known open-source software applications, seven of which are from Siemens [36] and six from Defects4J [37]. Table 4 provides a summary of these programs along with their corresponding test sets. All of these programs are well-known and have been extensively used to assess fault localization methods [6-8, 13, 36, 38, 39]. The Siemens suite was obtained from the Software-artifact Infrastructure Repository (SIR[2]), which includes seven small programs with seeded faults. Gcov (short for GNU coverage) was used to collect test case statement coverage information. Some versions were excluded due to compiler errors or segmentation faults, as it was not possible to extract the statement coverage of the test cases. We also used the Defects4J[3] suite (one of the largest available datasets of Java defects) [18] to

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

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

better evaluate our method on real faults. Versions without failed tests were removed. The dataset used in the experiment has been published [4].

Table 4. Subject Programs.

Program	Faulty Versions		LOC	#Tests	Fault Type
	All	Used			
printtokens	7	7	565	4130	Seeded
printtokens2	10	9	510	4115	Seeded
schedule	9	5	374	2650	Seeded
schedule2	10	9	307	2710	Seeded
tcas	41	39	173	1608	Seeded
tot_info	23	23	412	1052	Seeded
replace	32	29	563	5542	Seeded
Apache commons-lang	65	60	22K	2245	Real
Apache commons-math	106	104	85K	3602	Real
Mockito framework	38	30	23K	1457	Real
Joda-Time	27	27	28K	4130	Real
JFree Chart	26	24	96K	2205	Real
Closure compiler	133	132	90K	7927	Real

3.3. Evaluation Metrics

We used the following four metrics to compare the performance of FSBFL in improving the effectiveness of SBFL techniques.

- EXAMscore [40-44] represents the percentage of statements in a program that need to be inspected before identifying a faulty statement. A lower EXAMscore indicates more accurate fault localization.
- CNSE (Cumulative Number of Statements Examined) refers to the number of statements that need to be examined across all faulty versions of a program in order to identify the fault [1]. A lower *CNSE* means better fault localization.
- Accuracy (acc@n) counts the number of faults that have been localized within the top n positions of the ranking [45]. This metric is important because numerous studies indicate that developers tend to examine only a limited number of lines of code when identifying faults [46, 47]. A higher acc@n value indicates a better fault localization technique.
- Wilcoxon Signed-Rank test (WSR) is a non-parametric statistical hypothesis test that does not assume normal distribution. The paired *WSR* test uses the sign and the magnitude of the rank of the differences between pairs of measurements *F(x)* and *G(y)*. At the specified significance level σ, both 2-tailed and 1-tailed p-values are available for drawing a conclusion. Hypotheses for the 2-tailed p-value are: a) Null Hypothesis (H_0): *F(x)* and *G(y)* are not significantly different. b) Alternative Hypothesis (H_1): *F(x)* and *G(y)* are significantly different. For 1-tailed p-value, there are two cases, the lower case and the upper case. In the upper case, the hypotheses are: a) Null Hypothesis (H_0): *F(x)* does not significantly tend to be less than the *G(y)*. b) Alternative Hypothesis (H_1): *F(x)* significantly tends to be less than the *G(y)*. if $p \geq \sigma$, the H_0 is accepted; otherwise, H_1 is accepted.

3.4. Methodology

Using the same benchmark set, we evaluated the performance of FSBFL against four well-known SBFL methods (listed in Table 1) as well as two advanced techniques (FES [34] and CBS [35]). Reference [7] states that the DStar* formula achieves the highest fault localization accuracy when the parameter * is set to 3 (i.e., DStar3). To calculate the ACCWP, the FSBFL method uses a spectrum-based formula fm (as shown in (1) and (2)) to determine the suspiciousness scores of program statements. In this experiment, we used Op, DStar3, Ochiai, and Tarantula to compute the suspicion scores of program statements (i.e., SP(fm)) and named the resulting methods FOp, FDStar3, FOchiai, and FTarantula, respectively. Many lines in the program spectrum are duplicated. To improve execution efficiency, duplicate lines are removed before suspiciousness calculation. The suspiciousness score is then computed once for each unique line and propagated to all of its duplicate occurrences.

To answer RQ3, we chose the FES [34] and CBS [35] techniques because both utilize only the program spectrum and, like FSBFL, do not require additional instrumentation or source code analysis. CBS uses a rank aggregation method to combine multiple rank orderings on the same set of candidates (statements) to obtain a "better ordering." In implementing the CBS method, four distinct rankings were generated using four SBFL techniques (Op, DStar3, Ochiai, and Tarantula) and aggregated into a single consensus-based ranking. To enhance the effectiveness of fault localization, FES [34] employs a fuzzy expert system to combine various SBFL methods. We use FES to integrate four SBFL techniques (Op, DStar3, Ochiai, and Tarantula) and compare its results with those of FSBFL.

3.5. Experimental Results and Discussion

The empirical study was carried out on 498 versions of 13 subject programs, as shown in Table 4. To answer RQ1, four fault localization metrics (EXAMscore, CNSE, acc@n, and WSR) were used, and the results are shown in Tables 5, 6, 7, 8, and 9, respectively. Table 5 shows the EXAMscore comparison between the original SBFL and

[4] https://zenodo.org/records/10991894/files/dataset.zip?download=1

FSBFL, where a lower EXAMscore indicates better performance. For example, the value of 28.95 at the intersection of the printtokens row and the DStar3 column in Table 5 is the average EXAMscore of all faulty versions of the subject program printtokens. Bold data indicate that FSBFL performed better than the original SBFL technique. Bold data also show that, except for the FOp technique, FSBFL led to improvement in most subject programs. In the Overall row, the average EXAMscore of all faulty versions across all subject programs is shown. In this row, the best result (value 19.30) is indicated with an underscore and corresponds to FDStar3. The percentage of EXAMscore improvement compared to the original SBFL technique is shown in the Improvement row. The most significant improvement occurred with FTarantula (30.27%), but the best overall result belongs to FDStar3 (19.30). Table 6 shows the CNSE comparison between original SBFL and FSBFL, and a lower CNSE value indicates better performance. For example, consider the intersection of the printtokens row and the Op column, 326 statements must be checked to locate all faults of all faulty versions of printtokens when using the original Op technique. Bold data shows that in most subject programs, improvement is observed in FSBFL methods. The Overall row displays the total number of statements that need to be examined to identify all faults in every faulty version of all subject programs. The lowest value corresponds to the FDStar3 (*431281*), which is indicated by an underscore. As shown in the row *Improvement*, FDStar3 is associated with the most significant improvement, achieving a value of 35.56%. Table 7 presents the percentages of 498 versions in which FSBFL wins, ties, or loses against Original SBFL, FES, and CBS. It is clear that the percentage of wins is always higher than the percentage of losses. acc@n represents the number of faults identified within the top n positions of the ranked list. As shown in Table 8, we utilized acc@1, acc@3, acc@5, acc@10, and acc@30 to tally the respective faults within the subject programs. A higher acc@n value indicates more effective fault localization. The bold data show that the results of FSBFL are equal to or better than those of SBFL, and it is evident that the FSBFL method has led to improvements, except in FOp. The data with underscores represent the best results, and it can be seen that the FDStar3 method was consistently the best. In our experiments, we conducted two paired *WSR* tests, including both 1-tailed (upper) and 2-tailed test, to compare our method with the original SBFL, FES, and CBS methods at the σ level of *0.05*. Translated into our context, as mentioned in section 3.3, *F(x)* is the CNSE list for all subject programs based on our method; while *G(y)* is the CNSE list for all subject programs based on the original SBFL, FES, and CBS methods.

Table 5. The average EXAMscore of SBFL and FSBFL methods.

Program	*Op*	*DStar3*	*Ochiai*	*Tarantula*	*FOp*	*FDStar3*	*FOchiai*	*FTarantula*
Printtokens	24.51	28.95	34.21	44.21	24.51	**24.51**	**24.59**	**24.51**
printtokens2	2.56	6.25	10.83	16.52	2.67	**2.95**	**2.78**	**2.67**
schedule	20.82	19.90	20.97	9.03	21.09	20.69	**20.69**	20.69
schedule2	45.98	51.72	53.64	60.08	**45.63**	**45.63**	**45.63**	**45.63**
tcas	46.37	47.83	48.74	50.43	46.45	**46.17**	**46.37**	**46.49**
tot_info	13.19	18.31	21.53	27.01	**12.83**	**13.01**	**12.83**	**12.87**
replace	7.63	8.81	9.28	12.76	10.79	8.94	**8.94**	**9.31**
Lang	22.29	25.33	25.35	25.45	**15.51**	**14.49**	**14.44**	**15.29**
Math	31.54	36.03	35.94	35.12	**22.12**	**21.22**	**21.93**	**22.19**
Mockito	15.15	18.84	17.96	18.28	16.62	**15.66**	18.45	**17.88**
Time	26.14	38.56	38.34	38.34	**22.32**	**21.10**	**21.29**	**22.67**
Chart	16.18	20.16	19.94	19.34	19.08	**18.34**	**18.35**	**18.42**
Closure	16.91	22.54	22.33	22.51	**15.05**	**15.04**	**15.41**	**15.13**
Overall	22.98	27.50	27.81	28.52	**19.89**	**19.30**	**19.73**	**19.88**
Improvement					13.42%	29.82%	29.06%	30.27%

Table 6. The CNSEscore of SBFL and FSBFL methods.

Program	*Op*	*DStar3*	*Ochiai*	*Tarantula*	*FOp*	*FDStar3*	*FOchiai*	*FTarantula*
printtokens	326	385	455	588	326	**326**	**327**	**326**
printtokens2	46	112	194	296	48	**53**	**50**	**48**
schedule	158	151	159	68	160	157	**157**	157
schedule2	528	594	616	690	**524**	**524**	**524**	**524**
tcas	1175	1212	1235	1278	1177	**1170**	**1175**	**1178**
totinfo	373	518	609	764	**363**	**368**	**363**	**364**
replace	538	621	654	899	760	630	**630**	**656**
Lang	9154	10645	10659	10729	**4996**	**4435**	**4412**	**4886**
Math	93302	103142	102892	101983	**59554**	**57647**	**58759**	**59671**
Mockito	6651	8291	7951	8135	7069	**6587**	7666	**7468**
Time	36752	52785	52461	52454	**31991**	**30694**	**30872**	**32600**
Chart	16368	24029	24033	24896	**15089**	**14844**	**14886**	**14903**
Closure	360343	466759	462695	467330	**312598**	**313846**	**320479**	**313801**
Overall	525714	669244	664613	670110	**434655**	**431281**	**440300**	**436582**
Improvement					17.32%	35.56%	33.75%	34.85%

Table 7. Percentage of versions where FSBFL wins/ties/loses based on EXAMscore metric.

	Original SBFL			***FES***			***CBS***		
	Wins	*Ties*	*Loses*	*Wins*	*Ties*	*Loses*	*Wins*	*Ties*	*Loses*
Fop	29.52%	53.21%	17.27%	38.15%	43.78%	18.07%	34.74%	45.98%	19.28%
FDStar3	37.75%	55.02%	7.23%	35.94%	51.20%	12.85%	30.32%	56.43%	13.25%
FOchiai	45.18%	43.98%	10.84%	40.56%	43.57%	15.86%	35.54%	47.59%	16.87%
FTarantula	49.00%	38.15%	12.85%	40.76%	41.57%	17.67%	36.55%	44.78%	18.67%

Table 8. The acc@n of SBFL and FSBFL methods.

acc@n	***Op***	***DStar3***	***Ochiai***	***Tarantula***	***FOp***	***FDStar3***	***FOchiai***	***FTarantula***
acc@1	16	15	14	14	15	**16**	**16**	**16**
acc@3	55	54	52	47	53	**55**	**55**	**55**
acc@5	82	78	76	69	80	**82**	**82**	**81**
acc@10	123	117	113	107	119	**124**	**122**	**122**
acc@30	205	201	194	190	198	**207**	**204**	**206**

Table 9. Results of the paired Wilcoxon Signed-Rank test.

Method	*WSR*	*Original*	*FES*	*CBS*
Fop	2-tailed	2.43E-03	1.99E-04	2.58E-03
	1-tailed(U)	1.22E-03	9.93E-05	1.29E-03
FDStar3	2-tailed	2.55E-17	1.38E-06	1.28E-04
	1-tailed(U)	1.27E-17	6.91E-07	6.38E-05
FOchiai	2-tailed	2.78E-15	3.62E-06	1.75E-04
	1-tailed(U)	1.39E-15	1.81E-06	8.77E-05
FTarantula	2-tailed	3.21E-15	3.58E-05	5.84E-04
	1-tailed(U)	1.61E-15	1.79E-05	2.92E-04

As shown in Table 9, the p-values for all 2-tailed tests are less than *0.05*. This indicates that the performance of the proposed method and the compared methods are significantly different. Also, the p-values for all 1-tailed (upper) tests are less than *0.05*, implying that the alternative hypotheses (H_1) can be generally accepted, which implies that the proposed method requires the examination of fewer statements to locate faults than the original SBFL, FES and CBS methods. The data underlined in the overall row of Tables 5 and 6 show that the FDStar3 method has performed better overall. Additionally, column FDStar3 in Table 8 indicates superior performance of FDStar3 compared to other methods. Therefore, in the following, we compare the effectiveness of FDStar3 with two approaches: CBS (Consensus-Based Strategy) [35] and FES (Fuzzy Expert ystem) [34], as two of the most advanced methods. Table 10 compares the results of EXAMscore and CNSE metrics for FDStar3, FES and CBS methods. Data with underscores show the best results and the FDStar3 method performs best in most subject programs compared to FES and CBS

methods. In summary, from the Overall row, it can be concluded that FDStar3 performs best (in terms of EXAMscore and CNSE metrics) in improving the effectiveness of SBFL techniques compared to FES and CBS methods.

To finalize our comparisons, we also examined the number of faults identified within the top n positions (n = 1, 3, 5, 10, and 30, called acc@1, acc@3, acc@5, acc@10, and acc@30, respectively) in the rankings of the FDStar3, FES, and CBS methods. In summary, Table 11 demonstrates that FDStar3 achieves the best performance in terms of acc@n, thereby enhancing the effectiveness of SBFL techniques.

According to the above discussion, the following answers can be given to the research questions:

- In addition to the factors used in SBFL, other factors can be extracted from the program spectrum, which can lead to improvements in fault localization.
- Using a fuzzy expert system to integrate additional factors extracted from the program spectrum (along with the factors used in the original SBFL) leads to SBFL improvement.
- The CC identification factor extracted by DStar3 leads to better fault localization (i.e., FDStar3).
- Overall, FSBFL techniques outperform the original SBFL techniques in terms of all evaluation metrics.
- Overall, FDStar3 outperforms two competing methods (FES and CBS) in terms of all evaluation metrics.

Table 10. Comparison of FDStar3, FES and CBS.

Program	*EXAMscore*			*CNSE*		
	FDStar3	*FES*	*CBS*	*FDStar3*	*FES*	*CBS*
printtokens	24.51	26.54	28.80	326	353	383
printtokens2	2.95	5.41	3.17	53	97	57
schedule	20.69	20.95	19.90	157	159	151
schedule2	45.63	48.85	52.94	524	561	608
tcas	46.17	47.71	47.59	1170	1209	1206
totinfo	13.01	18.31	17.67	368	518	500
replace	8.94	8.81	8.17	630	621	576
Lang	14.49	21.97	21.99	4435	9043	9036
Math	21.22	31.38	31.41	57647	92930	93009
Mockito	15.66	14.72	14.68	6587	6504	6459
Time	21.10	25.35	25.44	30694	35573	35700
Chart	18.34	15.31	15.60	14844	15748	15880
Closure	15.04	16.44	16.64	313846	352925	354420
Overall	19.30	23.22	23.27	431281	516241	517985

Table 11. The acc@n of FDStar3, FES and CBS methods.

acc@n	*FDStar3*	*FES*	*CBS*
acc@1	16	15	16
acc@3	55	54	55
acc@5	82	78	81
acc@10	124	116	121
acc@30	207	202	204

4. Threats to Validity

The external validity of the method, which pertains to how well the experimental results can be generalized, is closely related to the choice of subject programs and fault types. As shown in Table 4, the subject programs include 13 popular open-source programs that have been widely used in the literature. The selected subject programs exhibit considerable variation in terms of their size, functionality, the number of faulty versions, and the total number of test cases. All faults in the Siemens suite are hand-seeded. To better evaluate the proposed method, the Defects4J suite is used because it is unclear whether artificial faults accurately represent the characteristics of real bugs [18]. A potential threat to construct validity arises from the metrics employed in our evaluation. We evaluate and compare the performance of FSBFL against other techniques (four popular SBFL techniques, FES, and CBS) using the EXAMscore, CNSE, acc@n, and WSR metrics. The primary threat to the internal validity of our experiments is the possibility of errors during the implementation of the techniques compared in this study. To mitigate this risk, we carefully examined the step-by-step traces manually, comparing our implementation with the manual execution of several small programs. Therefore, we believe this threat is minimal.

5. Conclusion and Future Works

Many efforts have been made to automate test data generation [51] and fault localization. *S*BFL techniques use a risk evaluation formula to convert the program spectrum and test results into statement suspiciousness scores. Coincidental correctness occurs when a test case passes (i.e., produces the correct output), not because the code is correct, but due to a fortunate or accidental alignment of circumstances. Coincidental correctness may adversely affect SBFL techniques, and some test cases are more informative than others. In this paper, in addition to the statement

suspiciousness factors used in the SBFL technique, two other factors (a CC identification factor for the passed test case and a weight factor for the failed test case) were proposed. We build a fuzzy expert system and propose a fault localization method, namely FSBFL. The proposed method uses an original SBFL risk evaluation formula to extract the value of the CC identification factor. We carried out experiments employing four effective metrics to assess FSBFL on 13 well-known open-source subject programs, varying in size from small (Siemens) to large (Defects4J), which include both artificial and real faults. Our experimental results suggest that incorporating additional factors can enhance the original SBFL method.

Table 12. Average improvement of EXAMscore and CNSE metrics for the Defects4J dataset.

Program	*Average Improvement*	
	EXAMscore	*CNSE*
Lang	39.04%	54.21%
Math	36.69%	41.17%
Mockito	1.66%	6.51%
Time	36.31%	33.45%
Chart	0.96%	31.06%
Closure	27.01%	27.40%

Table 12 is derived from the data presented in Table 5 and indicates that stronger gains can be achieved on specific Defects4J projects. For example, a 30.42% improvement in the EXAMscore metric for the *Lang* project, based on the FOp method (compared to the original Op method), can be calculated (Using data from Table 5) as follows:

(22.29–15.51)/22.29*100=30.42%.

Similarly, the improvements for FDStar3, FOchiai, and FTarantula are calculated as 42.80%, 43.04%, and 39.92%, respectively. The average of these four values (30.42, 42.80, 43.04, and 39.92) is shown (i.e., 39.04) at the intersection of row Lang and column EXAMscore in Table 12. Coincidental correctness tends to be more problematic in some types of software than others because certain computations can mask the effects of faults, allowing incorrect internal states to produce correct outputs. In the Defects4J dataset, the Math, Lang, and Time domains are numerical computations, string processing, and date/time processing, respectively. Table 12 shows that identifying CC in these three projects has led to stronger achievements. Fuzzy rules can be learned automatically from data instead of being manually defined by experts. In future work, we plan to investigate the effect of automatically learned fuzzy rules on the accuracy and efficiency of the proposed approach. Slice-based fault localization (FL) [48, 49] is a debugging technique that identifies likely fault locations by analyzing the program slice associated with a failure. Instead of examining all executed statements, it focuses solely on those that could have influenced the incorrect output through data and control dependencies. SBFL is fast and scalable, whereas slice-based FL offers greater precision. To integrate these two methods, SBFL can be applied exclusively to the statements within the slice, or the suspiciousness scores from SBFL and slice-based FL can be combined using a weighted average. Information retrieval (IR)-based fault localization [50] comprises techniques that identify buggy code by treating bug reports (as queries) and source code (as documents). These techniques use information retrieval algorithms to find the source files or methods whose textual content most closely matches the bug report. Several methods can be used to integrate these two approaches. As previously mentioned, the weighted average is a straightforward method for integrating two scores calculated by different methods. Another strategy is hierarchical localization. IR-based fault localization ranks files based on their similarity to the bug report, and then SBFL ranks statements only within these top-ranked files.

References

[1] 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, pp. 20-37, 2019.

[2] J. S. Collofello and L. Cousins, "Towards automatic software fault location through decision-to-decision path analysis," in *Managing Requirements Knowledge, International Workshop on*, 1987: IEEE Computer Society, pp. 539-539.

[3] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering,* vol. 42, no. 8, pp. 707-740, 2016.

[4] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*, 2006: IEEE, pp. 39-46.

[5] J. A. Jones, M. J. Harrold, and J. T. Stasko, "Visualization for fault localization," in *in Proceedings of ICSE 2001 Workshop on Software Visualization*, 2001.

[6] 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, no. 11, pp. 1780-1792, 2009.

[7] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Transactions on Reliability,* vol. 63, no. 1, pp. 290-308, 2013.

[8] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on software engineering and methodology (TOSEM),* vol. 20, no. 3, pp. 1-32, 2011.

[9] Y. Li and C. Liu, "Using cluster analysis to identify coincidental correctness in fault localization," in *2012 Fourth International Conference on Computational and Information Sciences*, 2012: IEEE, pp. 357-360.

[10] X. Xue, Y. Pang, and A. S. Namin, "Trimming test suites with coincidentally correct test cases for enhancing fault localizations," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014: IEEE, pp. 239-244.

[11] 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, no. 2, pp. 119-136, 2018.

[12] 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.

[13] 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, p. 110635, 2020.

[14] Z. Li, M. Li, Y. Liu, and J. Geng, "Identify coincidental correct test cases based on fuzzy classification," in *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*, 2016: IEEE, pp. 72-77.

[15] L. Naish, H. J. Lee, and K. Ramamohanarao, "Spectral debugging with weights and incremental ranking," in *2009 16th Asia-Pacific Software Engineering Conference*, 2009: IEEE, pp. 168-175.

[16] B. Baudry, F. Fleurey, and Y. Le Traon, "Improving test suites for efficient fault localization," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 82-91.

[17] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, "Directed test generation for effective fault localization," in *Proceedings of the 19th international symposium on Software testing and analysis*, 2010, pp. 49-60.

[18] S. Pearson *et al.*, "Evaluating and improving fault localization," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017: IEEE, pp. 609-620.

[19] J. M. Voas, "PIE: A dynamic failure-based technique," *IEEE Transactions on software Engineering,* vol. 18, no. 8, p. 717, 1992.

[20] 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 (TOSEM),* vol. 23, no. 1, pp. 1-28, 2014.

[21] R. Abou Assi, C. Trad, M. Maalouf, and W. Masri, "Coincidental correctness in the Defects4J benchmark," *Software Testing, Verification and Reliability,* vol. 29, no. 3, p. e1696, 2019.

[22] 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 (ISSTA 2009)*, 2009, pp. 1-5.

[23] 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, no. 05, pp. 721-741, 2013.

[24] Y. Li and C. Liu, "Identifying coincidental correctness in fault localization via cluster analysis," *Journal of Software Engineering,* vol. 8, no. 4, pp. 328-344, 2014.

[25] X. Yang, M. Liu, M. Cao, L. Zhao, and L. Wang, "Regression identification of coincidental correctness via weighted clustering," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, 2015, vol. 2: IEEE, pp. 115-120.

[26] L. A. Zadeh, "Fuzzy sets," *Information and control,* vol. 8, no. 3, pp. 338-353, 1965.

[27] T. J. Ross, *Fuzzy logic with engineering applications*. John Wiley & Sons, 2005.

[28] L.-X. Wang, "A course in fuzzy systems," 1999.

[29] I. Iancu, "A Mamdani type fuzzy logic controller," *Fuzzy logic-controls, concepts, theories and applications,* vol. 15, no. 2, pp. 325-350, 2012.

[30] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International journal of man-machine studies,* vol. 7, no. 1, pp. 1-13, 1975.

[31] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE transactions on systems, man, and cybernetics,* no. 1, pp. 116-132, 1985.

[32] P. C. Shill, K. K. Pal, M. F. Amin, and K. Murase, "Genetic algorithm based fully automated and adaptive fuzzy logic controller," in *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, 2011: IEEE, pp. 1572-1579.

[33] P. R. Srivastava, S. Kumar, A. P. Singh, and G. Raghurama, "Software testing effort: An assessment

through fuzzy criteria approach," *Journal of Uncertain Systems,* vol. 5, no. 3, pp. 183-201, 2011.

[34] C.-T. Lin, W.-Y. Chen, and J. Intasara, "A framework for improving fault localization effectiveness based on fuzzy expert system," *IEEE Access,* vol. 9, pp. 82577-82596, 2021.

[35] V. Debroy and W. E. Wong, "A consensus-based strategy to improve the quality of fault localization," *Software: Practice and Experience,* vol. 43, no. 8, pp. 989-1011, 2013.

[36] 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 16th International conference on Software engineering*, 1994: IEEE, pp. 191-200.

[37] 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 the 2014 international symposium on software testing and analysis*, 2014, pp. 437-440.

[38] F. Feyzi and S. Parsa, "Inference: effective fault localization based on information-theoretic analysis and statistical causal inference," *Frontiers of Computer Science,* vol. 13, no. 4, pp. 735-759, 2019.

[39] 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.

[40] W. E. Wong, V. Debroy, and D. Xu, "Towards better fault localization: A crosstab-based statistical approach," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 42, no. 3, pp. 378-396, 2011.

[41] X. Ju, S. Jiang, X. Chen, X. Wang, Y. Zhang, and H. Cao, "HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices," *Journal of Systems and Software,* vol. 90, pp. 3-17, 2014.

[42] 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, no. 2, pp. 188-208, 2010.

[43] W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, "Effective software fault localization using an RBF neural network," *IEEE Transactions on Reliability,* vol. 61, no. 1, pp. 149-169, 2011.

[44] W. E. Wong and Y. Qi, "BP neural network-based effective fault localization," *International Journal of Software Engineering and Knowledge Engineering,* vol. 19, no. 04, pp. 573-597, 2009.

[45] 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.

[46] F. Steimann, M. Frenkel, and R. Abreu, "Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 314-324.

[47] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th international symposium on software testing and analysis*, 2016, pp. 165-176.

[48] F. Tip, "*A survey of program slicing techniques*." *Journal of Programming Languages,* vol. 3, pp. 121-189, 1994.

[49] E. Soremekun, L. Kirschner, M. Böhme, and A. Zeller, "Locating faults with program slicing: an empirical analysis," *Empirical Software Engineering,* vol. 26, no. 3, p. 51, 2021.

[50] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *2012 34th International conference on software engineering (ICSE)*, 2012: IEEE, pp. 14-24.

[51] A. H. Damia, M. Esnaashari, and M. R. Parvizimosaed, "Software Testing using an Adaptive Genetic Algorithm," *Journal of AI and Data Mining,* vol. 9, no. 4, pp. 465-474, 2021

یک سیستم خبره فازی برای بهبود مکان‌یابی خطا مبتنی بر طیف

محمد مهدی استثنائی و سعید عربان*

گروه کامپیوتر، دانشکده مهندسی، دانشگاه فردوسی مشهد، مشهد، ایران

ارسال ۲۰۲۶/۰۴/۲۴؛ بازنگری ۲۰۲۶/۰۶/۰۶؛ پذیرش ۲۰۲۶/۰۷/۰۸

چکیده:

مکان‌یابی خطا مبتنی بر طیف، یکی از روش‌های پرکاربرد اســـت که به آزمونگر نرم‌افزار در پیدا کردن خطاهای برنامه کمک می‌کند. در این روش از پوشش دستورات برنامه و نتایج آزمایه‌ها استفاده شده و دستورات برنامه، به ترتیبی که مظنون به خطا هستند ، رتبه‌بندی می‌شوند. ایده این روش آن است که هرچه دستوری توسط آزمایه‌های ناموفق بیشتر و موفق کمتر، پوشش داده شود، احتمال معیوب بودن آن بیشتر است. موفقیت اتفاقی زمانی رخ می‌دهد که یک دستور معیوب توسط یک آزمایه اجرا شده، اما در عین حال نتیجه درست نیز حاصل شود. چنین آزمایه‌ای را، آزمایه‌ی اتفاقاً موفق می‌نامند. در این حالت، آزمایه‌ی مورد نظر به عنوان یک آزمایه‌ی موفق برچســـب خورده و باعث کاهش کارائی مکان‌یابی خطا مبتنی بر طیف می‌شـــود. همچنین روش‌های ســـنتی مکان‌یابی خطا مبتنی بر طیف، برای همه‌ی آزمایه‌های ناموفق، وزن یکســـانی قائل هســـتند، در حالیکه برخی از آن‌ها حاوی اطلاعات ارزشـــمندتری می‌باشـــند. این پژوهش با اســـتفاده از یک ســـیســـتم خبره فازی و در نظر گرفتن چالش‌های بیان شـــده، ســـعی در بهبود کارائی مکان‌یابی خطا دارد. به منظور ارزیابی روش پیشـــنهادی، ســـیزده برنامه‌ی متن باز به همراه تعداد زیادی آزمایه مورد اســـتفاده قرار گرفت. نتایج حاصـــل از چهار معیار ارزیابی نشان می‌دهد که روش پیشنهادی، کارائی بیشتری نسبت به روش‌های سنتی مکان‌یابی خطا مبتنی بر طیف دارد.

کلمات کلیدی: آزمون نرم‌افزار، مکان‌یابی خطا مبتنی بر طیف، موفقیت اتفاقی، سیستم خبره فازی.