Leveraging Evidence Theory to Improve Fault Localization: An Exploratory Study

Yueke Zhang
Vanderbilt University
yueke.zhang@vanderbilt.edu

Kevin Leach
Vanderbilt University
kevin.leach@vanderbilt.edu

Yu Huang
Vanderbilt University
yu.huang@vanderbilt.edu

***Abstract*—Background: Fault localization in software maintenance and debugging can be a costly process. Spectrum-Based Fault Localization (SBFL) is a widely-used method for fault localization. It assigns suspicion scores to code elements based on tests, indicating the likelihood of defects in specific code lines. However, the effectiveness of SBFL approaches varies depending on the subject code.**

Aims: In this paper, our aim is to present an approach that combines multiple SBFL formulae using evidence theory.

Method: We first introduce a taxonomy of SBFL techniques. Then, we describe how we fuse suspiciousness scores obtained from a set of SBFL formulae. We also introduce a concept of fuzzy windows, and describe how they can enhance localization accuracy and how they can be tuned to further refine results.

Results: We present an empirical evaluation of our approach using the Defects4J dataset. Our results demonstrate improvements in fault localization accuracy over existing statement-level SBFL techniques. Specifically, by fusing three SBFL methods, our approach reduces code inspection effort by up to 34.5% with a size-4 window and increases the hit rate for the top 10% most suspicious lines by 27.9% using a size-7 window. Moreover, in multi-line bug scenarios, our approach reduces code inspection effort by up to 35.6% and achieves a maximum increase of 43.2% in the hit rate of the top 10% most suspicious lines. Additionally, our approach outperforms state-of-the-art machine learning-based method-level fusion approaches in terms of top rank fault localization accuracy.

Conclusions: Our study highlights the applicability of evidence theory in addressing fault localization as an uncertain and ambiguous information fusion problem involving multiple SBFL techniques. The combination of SBFL formulae using evidence theory, along with the use of fuzzy windows, shows promise in enhancing fault localization accuracy.

***Index Terms*—evidence theory, information fusion, uncertainty, fault localization**

I. Introduction

Software debugging is expensive and time-consuming as the scale and complexity of software increases in modern software engineering. In the United States alone, software bugs and defects lead to billions of dollars of loss every year [1]. In the software industry, testing and debugging takes average costs of 48% in the software development process [2], [3]. As software becomes more complicated, software systems also become more difficult for humans to debug efficiently and effectively [4]. Furthermore, the accuracy of locating bugs can substantially affect the performance of automated bug repair techniques [5]–[7].

Researchers have invested significant effort in creating automated solutions to ease software debugging, especially locating bugs, as an important first step in software maintenance. Spectrum-based fault localization (SBFL) [8]–[10] is one of the most important and popular approaches, where coverage information of program elements (e.g., line coverage) is collected for passing and failing test cases, culminating in a suspiciousness score that indicates the likelihood that each element (e.g., line of code) contains a defect.

Lucia et al. [11] applied the concept of association measures with fault localization and compared 40 measures for fault localization in terms of how suspiciousness scores are presented. Similarly, Lo et al. [12] combined multiple SBFL techniques and used straightforward statistical measures to improve fault localization. More importantly, their analysis shows that there is no best single measure for all cases in fault localization. Other combination approaches, such as Savant [13], Multric [14], and FLUCCS [15] use machine learning algorithms like learning-to-rank on program spectra or bug reports to extract features, then predict the composition of high performance SBFL formulae.

Li et al. proposed a deep learning approach for higher fault localization accuracy at the function level using program features [16]. However, in these ML-based methods, extracting features like line age and churn may be difficult to interpret, limiting their broad applicability across all software projects. These approaches also require significant computational resources and training data, and have not shown accuracy for fine-grained program elements. Thus in this paper, we focus on fine-grained statement-level SBFL, but still compare our approach with AI-based models on the method-level fault localization to provide insights for the community on two different lines of approaches.

In general, fault localization approaches are based on heuristics, each of which emphasizes different aspects of programs and test cases. As a result, each method entails some degree of *uncertainty* — there does not exist a *best* solution, and different methods can perform better in different scenarios, as demonstrated by Lucia et al. [11]. Moreover, uncertainty is inevitable and inherent in software engineering, both in planning and enacting tests [17]. The practical constraints of finite test suites and coverable program elements introduce a degree of uncertainty for whether all defects have been detected.

Meanwhile, existing work in evidence theory (also referred to as Dempster-Shafer Theory) has been used to fuse information to make robust decisions by combining evidence or observations from multiple *sources* of information or evidence [18], [19]. Such approaches are applied in scenarios where each source of evidence about some proposition is imperfect — that no one source is a perfect reflection of all information, but taken in aggregate, all sources provide a more robust view of evidence. Following this insight, we can treat results from different FL formulae as multiple independent uncertain sources of information about which elements of the program are most suspicious. That is, we can view an FL formula as providing a single source of evidence about the proposition that a particular program element is suspicious or defective. Thus, we can apply evidence theory to combine output from multiple disparate FL formulae that hold close or conflicting views over which program elements are suspicious. We hypothesize that this notion can help to improve the overall quality of predicted suspiciousness scores associated with each program element.

In addition, we can also consider that individual FL formulae are ambiguous in identifying specific suspicious program elements — for example, that an FL formulae that identifies one line as suspicious may report nearby lines as suspicious as well. From this insight, we develop a notion of *fuzzy information fusion* over suspicious program elements by smoothing suspiciousness over multiple nearby elements. Fuzzy fusion remains underutilized in the field of Software Engineering. Existing work primarily focuses on its application in Usage-Based Reading to enhance the effectiveness of reading software documentation [20]. However, we note that fuzzy fusion has found wider application in other computing domains. For example, fuzzy fusion has been used to reduce semantic ambiguity in Natural Language Processing [21], to detect phishing attacks in security research [22], and to enhance template matching [23].

In this paper, we present an approach to fuse together multiple fault localization techniques using evidence theory. We leverage the insight that SBFL techniques can be treated as imperfect sources of information about defective lines of code or program elements. Multiple uncertain and ambiguous sources of SBFL information can be combined naturally using extant evidence theory fusion techniques. We assume (1) Each SBFL approach is reasonable (i.e., not a random guess) and (2) Each SBFL score for a line of code is *ambiguous*, within a "window" of error. Defects on line X may indicate high SBFL scores within lines $X \pm \epsilon$, which we call *fuzzy windows*. By applying evidence theory along with imperfect SBFL scores, our approach can improve fault localization by combining findings from multiple SBFL formulae. We apply our approach on the widely-used Defects4J dataset [24]. We also include 9 indicative SBFL formulae in the study and explored the effect of fuzzy window sizes on the localization accuracy.

Our results show that directly fusing the predictions made by diverse SBFL formulae improves the FL accuracy by 31.8% on average. By modeling only the ambiguity of suspiciousness with fuzzy windows of program elements, the FL accuracy increases by 34.5%. Specifically, the FL accuracy of multi-line bug scenarios can be improved by as much as 35.6%. In addition, our method locates 43.2% more bugs than average among the top 10% most suspicious lines. Though our approach focuses on statement-level FL, we still compare our approach against the state-of-the-art ML-based fusion method, Multric, on method-level FL. Our approach surpasses Multric on the accuracy of the top ranked buggy methods. Furthermore, our approach does not introduce additional testing load, but requires only linear mathematical computation associated with each SBFL formulae considered.

The main contributions of this paper are as follows:

- We propose a new categorization of traditional SBFL methods by modeling them as individual sources of evidence, each of which develops suspiciousness scores according to different program characteristics. This categorization distinguishes the suspiciousness evaluation corresponding to specific test traces of a FL task and allows more efficient information fusion.
- We propose a fuzzy window approach to model the ambiguity of suspiciousness evaluation in each single SBFL method, and demonstrate that the smoothed suspiciousness measure improves FL accuracy.
- We provide a framework to model the fault localization process as a fusion task. Our evaluation presents the efficiency of such fusion-based approaches for FL tasks.
- We present a systematic evaluation of our framework and demonstrate the feasibility and benefits of modeling the uncertainty of FL using evidence theory. This work can lead to further exploration and improvement in other automated software engineering processes.
- We openly share our dataset and evaluation scripts to encourage further research in this area.[1].

II. Background and Related Work

We discuss preliminaries to understand our work and place it in context with existing research. Specifically, we discuss (1) evidence theory as a basis for quantifying evidence and fusing information sources, (2) spectrum-based fault localization, (3) research that combines fault localization techniques, and (4) fault localization for defects that span multiple lines or program elements.

A. Evidence Theory, Uncertainty, and Fusion

Evidence theory, also referred to as *theory of belief functions* or *Dempster-Shafer theory (DS-theory or DST)*, along with fuzzy set theory, are essential in fuzzy information fusion tasks [18]. In particular, we use *Dempster's Combination Rule (DCR)*, to aggregate the suspiciousness measured by different spectrum fault localization techniques. Evidence theory was proposed by Dempster and later developed by Shafer [25],

[1]https://anonymous.4open.science/r/ESEM-fuzzy-FL-70CA

[26], and provides a formal mathematical approach to interpreting and combining distinct sources of evidence that support or reject various propositions. Informally, following foundational probability theory, evidence theory considers a set of propositions and a list of probabilities from 0 to 1 that capture each source's confidence in supporting each proposition. Thus, if each source consists of a list of such probabilities, then DCR allows combining these lists together to form a more robust representation of support for each proposition. If multiple sources agree on supporting a particular proposition, then fusing those sources will increase support for that proposition and reduce support for others. In this paper, we treat suspiciousness scores produced by individual fault localization methods to reflect the confidence each formula has that a given program element (i.e., line) is suspicious or defective.

We say that such propositions are *uncertain* in that they are imperfect and may differ between sources of information — for example, two different fault localization techniques might each report that different lines are more suspicious than others, even if only one line is truly defective. Evidence theory has been applied to computing to minimize ambiguities in specific tasks [27]. Uncertain fusion models have been used to resolve ambiguous languages in NLP area, to detect phishing attacks, and to improve template matching [21]–[23], [28]. It also has enjoyed wide application in computer vision, recommendation systems, and sentiment analysis [29]–[32]. In addition, it has been applied in risk evaluation, supply chain management, and commercial security in finance and management areas [33]–[35]. Evidence theory is a mature and well-tested set of techniques — however, it has not yet been adopted by the software engineering community. In our work, we apply to fault localization to improve software maintenance.

1) Formal Basis of Evidence Theory: Evidence theory develops support for a set of propositions — in our context, each line of code has an associated proposition that it in fact includes a bug or not. These propositions, referred to as a *frame of discernment*, are mutually exclusive (e.g., a line of code can either include a bug or not) and collectively exhaustive. We take the frame of discernment, called Ω, to be a finite set, $\Omega = \Theta_1, \Theta_2, \Theta_3 ... \Theta_n$, consisting of n mutually exclusive propositions (e.g., $n = 2$ for FL, each line of code has a bug or not).

Next, consider a function, $m(\Omega)$, called the Basic Probability Assignment (BPA), sometimes referred to as a *basic belief assignment* or *mass function*, on 2^n, the power set of Ω.

The BPA function m must satisfy:

$$m(\phi) = 0 \tag{1}$$

where ϕ is an empty set

$$\sum_{\Theta_i \subset 2^\Omega} m(\Theta_i) = 1 \tag{2}$$

In this arrangement, $m(X)$ measures the support that is directly assigned to proposition $X \subset \Omega$. Because $m(X)$ is between 0 and 1, it can capture varying degrees of support as a probability for a particular proposition. In this way, it also means that uncertainty among propositions is captured in $m(\Omega)$.

Evidence theory provides an approach for fusing sources to reduce uncertainty. In this context, fusing two sources consists of combining m_1 and m_2, the BPAs for each input source. Fusing these two masses results in redistribution of mass among the propositions such that more belief is applied to propositions that the sources agree upon. In particular, Dempster's Combination Rule is denoted by $m = m_1 \oplus m_2$ and computed in the following manner:

$$m(\phi) = 0 \tag{3}$$

$$m_{1,2}(\alpha) = \frac{1}{1-K} \sum_{\beta \cap \gamma = A \neq \phi} m_1(\beta) m_2(\gamma) \tag{4}$$

$$K = \sum_{\beta \cap \gamma = \phi} m_1(\beta) m_2(\gamma) \tag{5}$$

where β and γ are propostions of BPA m_1 and m_2 called the focal elements and K is the conflict coefficients. When focal elements in two BPAs do not overlap, K measures the degree of uncertainty between two belief functions. To combine n BPAs:

$$m = m_1 \oplus m_2 \oplus m_3 ... \oplus m_n \tag{6}$$

DCR's mathematical properties allow the combinations of the beliefs from multiple sources. There are other combination rules available with different conflict coefficients that are useful in different contexts. For example, Yager, Dubois and Murphy also proposed a combination rule [36]–[38]. We use DCR because it is commutative and associative, which facilitates rapid and parallel computation [23].

In this paper, we treat SBFL techniques as individual sources of information, where each line's suspiciousness is a proposition, and each line's suspiciousness score s_i contributes to a Dempster mass $m(Line_i)$. We use DCR to fuse together the suspiciousnes scores reported by SBFL techniques to improve overall fault localization.

B. Spectrum based fault localization

Spectrum-Based Fault Localization (SBFL) is one of the most popular and widely-researched fault localization methods [8] in part because of its relatively low execution time and reasonably high accuracy. SBFL uses passing and failing test cases over elements in a given code snippet (typically individual lines of code), then computes a suspiciousness score for each element. The suspiciousness score indicates the likelihood that the program element in question is responsible for defective behavior (typically, whether the line is implicated by the failing test cases). Finally, SBFL ranks all elements based on the score. Many suspiciousness evaluation formulae have also been proposed, including *ochiai* and *Tarantula* among others [39]–[41].

SBFL suspiciousness formulae vary in how they treat passed and failed test cases, whether a particular program element is

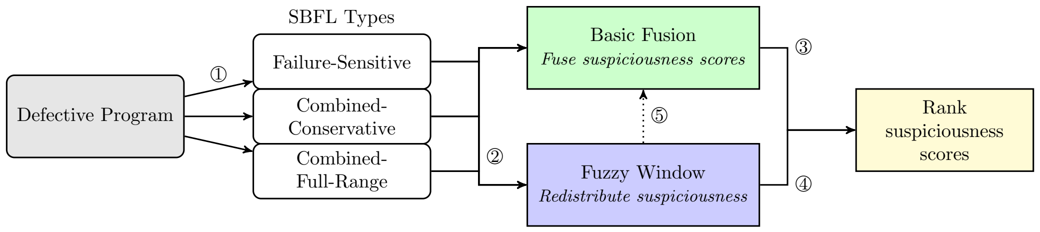

Fig. 1. Fusion and fuzzy window approach. First, we begin by feeding a defective program and test cases to multiple SBFL techniques (1). Next, we feed suspiciousness scores to our windowing and fusion approaches (2). In particular, we consider three scenarios. We consider fusing the raw suspiciousness scores produced by each category of SBFL technique (3). We also consider a fuzzy windowing technique in which suspiciousness scores are redistributed with nearby lines of source code in the original program for each SBFL technique (4). Finally, we combine windowed suspiciousness scores with fusion to produce an accurate suspiciousness score that leverages multiple SBFL techniques (5). These suspiciousness scores are all ranked and returned to the user.

covered during each test case, and the number of test cases overall. As a result, different formulae will more accurately localize defects than others depending on the input program.

C. Combining fault localization methods

Even though many fault localization techniques have been studied, there is no single winner than outperforms all others, both in terms of accuracy and computation cost [42]–[44]. Therefore, researchers have studied combinations of different methods, which may aggregate the advantages of each individual technique. Recently, Jiang et al. proposed a combination of SBFL and statistical debugging SD, which achieves a high improvement compared to the original SD method [45]. Other attempts have fused SBFL with mutation-based fault localization and slice-hitting-sets [46], [47].

In addition, combinations of SBFL have been widely-researched as well. Lucia has examined 40 different SBFL equations and their performance both on single-line bugs and multi-line bugs [11]. Lo et al. adapted all 40 equations and combined half of them based on ranking similarity [12]. Xuan et al. employed a learning-to-rank training on 25 suspiciousness evaluations and combined them based on weight [14]. In this paper, we develop a new approach to fusing SBFL techniques to improve overall fault localization accuracy.

D. Localizing multiple faults

There have been many approaches on fault localization [4], [8], [48], however they focus on single-fault programs. Bug interaction is common in programs, and localizing several bugs usually leads to more complexity, computation, and effort [49]. Existing approaches include one-bug-at-a-time (OBA), parallel, and multi-bug-at-a-time (MBA) [50]–[52]. Among these techniques, OBA is popular as it is fairly direct: the test suite is executed until a single bug is found. SBFL has been used in OBA [53], [54] as well. In this paper, we explore how SBFL techniques perform on a dataset of Java code, including multiple defects at a time.

III. Approach

In this section, we introduce our framework for exploring and modeling uncertainty of fault localization as a fuzzy information fusion problem. Figure 1 illustrates our approach. First, we taxonomize families of Spectrum-Based Fault Localization (SBFL) suspiciousness techniques according to what program information they emphasize and benefit from. Second, we use these SBFL techniques to report suspiciousness scores for a given defective program. Third, we develop and apply a *fuzzy window* technique in which we redistribute suspiciousness scores associated with each line of source code to nearby lines in the original program. Forth, we fuse these fuzzy window scores together to further improve localization. Finally, we combine fusion with our fuzzy window approach to provide accurate fault localization. These suspiciousness scores obtained from each approach are ranked so that a developer can more efficiently locate the defect.

A. Information-based Categorization of SBFL

There are more than 40 different formulae for producing suspiciousness scores in SBFL and there is no single best performer [11]. The state-of-the-art SBFL formulae vary according to the datasets to which they are applied. Table I shows an example of several lines of code and associated suspiciousness scores as reported by *ochiai* and *ample*, two different SBFL techniques. The red lines in the table indicate true bugs, but the lines with the highest suspiciousness according to *ochiai* are not buggy, and only one line is identified as buggy by *ample*. In this example, *ochiai* fails because there is no test case that fails specifically on lines 4 and 5. However, *ample* accounts for both passed and failed test cases, resulting in nonzero suspiciousness for lines 4 and 5. This leads us to two insights. First, we can categorize SBFL formulae according to the information they emphasize (e.g., how much they account for failed test cases, passed test cases, and lines covered by each). The SBFL taxonomy is established on the basis of suspiciousness coverage and the range of suspicious scores allocated. Each category is suitable for different types of code snippets, taking into account factors such as test coverage and fault frequency. Second, we can potentially increase fault localization accuracy by fusing together suspiciousness scores reported by SBFL techniques in each category. *Taxonomy of SBFL Formulae*: Some research has investigated the granularity of program elements used by SBFL formulae such as lines and functions [55], [56]. We focus on the line (statement) level for measuring suspiciousness because it is straightforward to instrument and because it is used by many techniques [45].

These SBFL formulae require the total number of failed and passed test cases for a code snippet, as well as the number of failed and passed test cases that cover a specific line. In our study, we categorize SBFL methods into three groups based on values of suspiciousness scores and how passing and failing test cases are reflected.

- **Failure-Sensitive** (FS). Only failed test cases are used in computing a suspiciousness score for each line.
- **Combined-Conservative** (CC). This category conservatively estimates suspiciousness considering both passing and failing test cases for each line, and produces only a non-negative number. This category produces a suspiciousness of 0 only when a line is never executed.
- **Combined-Full-Rate** (CFR). This category considers both passing and failing test cases, but allows for negative numbers when they build evidence that a line is *not* suspicious.

In this paper, we also use these nine SBFLs listed in Table Table II for exploratory analysis on the proposed evidence theory-based approach. We categorize the 40 available SBFL formulas into three groups and select nine formulas for this paper based on their categories. In the Failure-Sensitive (FS) category, our five formulas perform the best in defects4j. However, in the Combined-Conservative (CC) and Combined-Full-Rate (CFR) categories, there are only 2 or 3 formulas. Hence, we evaluate all of them and select the top 2 formulas from each category.

B. Fusion for SBFL

We model each SBFL category (cf. Section III-A) as evidence for fault localization. SBFL techniques provide imperfect and uncertain information regarding defective lines of code or program elements. Each SBFL source may offer a unique set of suspiciousness scores that can be fused together. Recall from Section II-A that we treat each SBFL technique's suspiciousness as an evidence theory mass function that we can fuse with Dempster's Combination Rule. We fuse all sources of information by modeling each source's uncertainty to generate a final suspicious score for each line.

1) Normalization: The range of suspiciousness evaluations in different SBFL can vary substantially. For example, Zoltar generates small suspiciousness scores that are usually less than 10^{-5}, which complicates fusing the raw data. Thus, we adjust suspiciousness scores so they vary from 0 to 1. Function S maps each element e to a suspiciousness score. We compute S_{max} and S_{min} to be the highest and lowest suspiciousness score seen among all elements for a single SBFL method. For each program element e, the post-normalization score for a SBFL can be calculated as

$$S_{norm}(e) = \frac{S(e) - S_{min}}{S_{max} - S_{min}} \tag{7}$$

Once all scores are normalized, they can be fused together.

2) Selecting SBFL methods as sources of evidence: Based on our three SBFL categories, we choose one formula from each group for fusion. Lo et al. have pointed out some SBFL formulae sometimes generate identical rankings for all lines [12]. Thus, combining these methods would be inefficient as no information would be gained by fusing them. Instead, by selecting one formula from each of our three categories, we avoid redundant suspiciousness rankings. The range of non-zero suspiciousness is different between categories, which assures the suspcisiousness computation process for code snippets varies among formulae.

3) Uncertainty Measure: Recall that we use evidence theory to represent suspiciousness scores from SBFLs as sources of evidence that can be fused together to obtain robust fault localization results. Doing so reduces uncertainty induced by individual SBFLs, which has not been discussed in prior work.

The suspicious scores that different formulae assign to each line is similar to multi-criteria information fusion [57]. We leverage the insight that each test case for a code snippet can be treated as a criterion for which an SBFL formula can provide information. When different information from SBFLs have conflicting or close outcomes, the final result is a comparison among all SBFL formulae.

After normalizing suspicious scores, we build a Belief Probability Assignment (BPA) for each line (see Section II-A1). Now, there are two conditions in fault localization: a line could be either buggy or nonbuggy. In this context, our BPA function m assigns a suspciousness score to each line that reflects the degree to which each SBFL formula concludes the line is buggy. We can directly use the suspiciousness score for line e in the mass function for potentially buggy lines when the suspiciousness is nonzero (i.e., $m(e) = S(e) = \textit{suspiciousness}$).

In our setup, we have three BPAs, m_1, m_2, and m_3, corresponding to one SBFL from each of our three categories (see Section III-A and Table II). Next, we use the Dempster Combination Rule to combine suspiciousness for each line. After fusing results for each line, we compute a ranking based on the fused suspiciousness scores.

C. Fuzzy windows for suspiciousness evaluation

SBFL is based on passing and failing test cases. Indeed, we see many examples throughout the Defects4J dataset in which other lines near a highly suspicious line are also marked as suspicious, even if those nearby lines are not actually implicated in a failing test case. Table III shows suspiciousness scores from Math11 in Defects4J. Column 2 shows raw suspicious scores, where lines 3–7 have suspiciousness 0.58 even though only line 5 is buggy. As a result, we consider localization information to be *ambiguous* among multiple nearby lines.

We adopt a window-based technique: we first pick a window size, w, and for each line in a given code snippet, we set the suspiciousness of that line to be the average suspiciousness of the next w and previous w lines in the code snippet. In Table III, we show the suspiciousness scores using $w = 2$. As we can see, line 5 has the highest suspiciousness, and thus would be considered first in typical SBFL applications that rank according to suspiciousness. We apply this fuzzy window technique in our experiments.

TABLE I
EXAMPLE DEFECTIVE CODE AND CORRESPONDING SUSPICIOUSNESS SCORES ACCORDING TO *ochiai* AND *ample*.

	Code line	**Ochiai**	**Ample**
1	public Weight(double[] weight) {	0.32	0.78
2	final int dim = weight.length;	0.32	0.78
3	weightMatrix = org.apache.commons.math3.linear.MatrixUtils.createRealMatrix(dim, dim);	0.32	0.78
4	for (int i = 0; i <dim; i++) {	0	0.24
5	weightMatrix.setEntry(i, i, weight[i]);}}	0	0.24

TABLE II
TAXONOMY OF POPULAR SBFL FORMULAE.

categories	**SBFL Formula**
Failure-Sensitive (FS)	$ochiai(e) = \frac{failed(e)}{totalfailed \times (failed(e)) + passed(e)}$
	$tarantula(e) = \frac{\frac{failed(e)}{totalfailed}}{\frac{failed(e)}{totalfailed} + \frac{passed(e)}{totalpassed}}$
	$dstar(e) = \frac{\sqrt{failed(e)}}{passed(e) + totalfailed - failed(e)}$
	$zoltar(e) = \frac{failed(e)}{passed(e) + totalpassed + \frac{1000 passed(e)(totalpassed - failed(e))}{failed(e)}}$
	$jaccard(e) = \frac{failed(e)}{totalfailed + passed(e)}$
Combined-Conservative (CC)	$ample(e) = \left\lvert \frac{failed(e)}{totalfailed - failed(e)} - \frac{passed(e)}{totalpassed - passed(e)} \right\rvert$
	$gp02(e) = 2 \times (falied(e) + \sqrt{totalpassed - passed(e)} + \sqrt{passed(e)})$
Combined-Full-Rate (CFR)	$muse(e) = failed(e) - \frac{totalfailed * passed(e)}{totalpassed}$
	$Piatetsky\_Shapiro(e) = failed(e) - totalfailed \times (passed(e) + failed(e))$

Consider S to be a function that returns the suspicious score for a line e. $S_{win}(e)$ computes suspiciousness for e using a fuzzy window of size w:

$$S_{win}(e) = \frac{S(e_{i-w}) + S(e_{i-w+1})... + S(e_{i+w})}{w} \tag{8}$$

Here, i refers to the line number where e occurs — that is, we assign $S_{win}(e)$ to be the average suspiciousness of the nearby $\pm$win lines.

For lines at the beginning or end of the program, the window may begin or end prematurely. In this case, the window includes w lines at the beginning or end of the snippet.

D. *Fusion of fuzzy window suspiciousness*

Finally, we also consider combining fuzzy windows with fusion of the suspiciousness scores from three SBFL techniques together for a given code snippet.

First, we normalize SBFL suspiciousness scores obtained from one of each of our three categories of SBFL techniques. As in Subsection III-B, we can use our fuzzy window suspiciousness scores as Dempster mass functions (i.e., $m_{win}(e) = S(e) = suspiciousness$). We demonstrate the efficacy of this approach in our evaluation.

IV. EVALUATION

In this study, we aim to address these research questions:

- RQ1. Can modeling the uncertainty in FL with evidence theory accurately and robustly localize defects?
- RQ2. Can we account for ambiguity of suspiciousness scores produced by SBFL and improve the accuracy of fault localization?
- RQ3. Can fusing both uncertain and ambiguous SBFL formulae improve FL accuracy?
- RQ4. Can our fusion approach aid in the localization of multi-line bugs?

In our experiments, we apply our framework on the widely-used dataset for fault localization and automated program repair, Defects4J [24], which is a collection of bug scenarios in open source Java projects [16], [58]. Defects4J contains six open-source projects, including Commons Math (Math), Commons Lang (Lang), JFreeChart (Chart), Joda Time (Time), Closure Compiler (Closure) and Mockito, which results in 395 bug scenarios, 366K line of code, and 22,224 test cases. We also include nine individual SBFL shown in Table II, and state-of-the-art fusion fault localizers for evaluation.

Metrics. We use *Exam Score* and *Hit Rate Per n* for defects to evaluate the performance of each method at the line level. Furthermore, in the method-level evaluation, we apply the *Top-n*, *MFR*, and *MAR* metrics.

Exam Score. The Exam Score is a metric that ranges from 0 to 1 that represents the percentage of code that, in the worst case, must be inspected after ranking suspiciousness scores of a given code snippet. For example, if the Exam Score is 0.5, it means we must inspect half of the entire codebase to localize the buggy line. Thus, an effective SBFL technique produces a low Exam Score because it is more likely to rate the truly defective line as highly suspicious.

Hit Rate Per n. Hit Rate Per n (abbreviated as PERn) shows how many faults can be found in top n percent of code after ranking. For our experiments, we measure Hit Rate Per $n = 5, 10, 20$. Given an SBFL technique, a higher Hit Rate

TABLE III
AN EXAMPLE ON FUZZY-WINDOW BASED SUSPICIOUSNESS SCORE (w=2).

	Code line	sus-score	sus-score (fuzzy)
1	public double density(final double[] vals) throws DimensionMismatchException {	0	0.19
2	final int dim = getDimension();	0	0.29
3	if (vals.length != dim) {	0.58	0.35
4	throw new DimensionMismatchException(vals.length, dim);}	0.58	0.46
5	return FastMath.pow(2 * FastMath.PI, -0.5 * dim) *	0.58	0.58
6	FastMath.pow(covarianceMatrixDeterminant, -0.5) *	0.58	0.46
7	getExponentTerm(vals);}	0.58	0.35
8	public double[] getStandardDeviations() {	0	0.29
9	final int dim = getDimension();	0	0.19

is better because it means the technique can rank a higher density of truly-defective lines as being suspicious [14], [45]. In this paper, we take the mean rank for all lines implicated by the defect.

Top n. We apply Top n, the widely used primary metric for evaluating method-level fault localization, to compare our approach with ML-based method-level approaches. Top n refers to the number of successful identification of buggy methods that are ranked among the first n using the method-level FL. Because our approach focuses on fine-grained statement-level FL, to allow the comparison with method-level FL approaches, we identified the method FL by localizing the methods where the most suspicious line of code resides.

MFR and MAR. Mean First Rank (MFR) is the average rank of the first identified buggy method. The Mean Average Rank (MAR) represents the average rank of all buggy methods.

Computational Resources. In our study, all experiments are conducted on a machine with an AMD Ryzen5 4600H 3.00 GHz CPU and 6GB RAM running Ubuntu 18.04. Since the Dempster Combination Rule is a linear formula, its contribution to computation time is negligible compared to the time spent executing test cases, collecting coverage information, and computing suspiciousness scores. Specifically, DCR took 0.026s to fuse three SBFL formulae for 1,000 lines.

A. RQ1: Modeling FL as a fusion task

Recall from Sections I and III that we apply evidence theory to fault localization by treating each SBFL technique as a source of evidence about a defect for a given snippet of code. In RQ1, we explore whether applying evidence theory with uncertainty can improve the accuracy of the final prediction of bug locations. Specifically, we select one SBFL formula from each of our three categories (Section III-A). Then, as described in Section III-B, we use each SBFL formula to compute suspiciousness scores for each line in each Defects4J project. We fuse the three resulting suspiciousness scores for each line to mitigate uncertainty. Then, based on the fused suspiciousness scores, we compare the accuracy of fault localization between our fusiong technique and each individual SBFL formula using the Exam Score. Table IV shows all 20 trials covering all SBFL combinations from the three categories of SBFL (see Section III-A). The *TrialID* represents the first letter of an SBFL formula in its category as shown in Table II. For example, Trial TAM is the trial that includes the

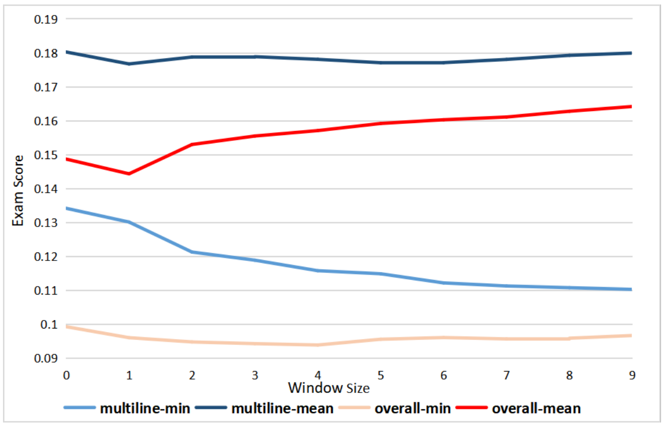

Fig. 2. Exam Score computed for all SBFL methods with window size w from 0–9 applied to Defects4J. *min* refers to the minimum (i.e., best) performance among our 9 SBFL formulae. *mean* refers to the average performance among our 9 SBFL formulae.

individual SBFLs Tarantula, Ample, Muse from each category (i.e., those that produced the lowest exam scores).

From Table IV, our fusion approach obtains the best FL performance when fusing with the best individual SBFLs. However, in general, while fusion provides better FL accuracy than average SBFL, it is not always the best compared to every possible individual SBFL. This observation is similar for the multi-line bug scenarios. That said, we will show in subsequent subsections that resolving ambiguity in suspicious scores among nearby lines can substantially improve fusion-based SBFL (c.f. Section IV-B).

B. RQ2: Fuzzy Windows for SBFL

Following the approach described in Section III-C, we model the ambiguity of suspiciousness evaluation for every line of code. We apply a window of a fixed size w to the raw suspiciousness scores computed for each code snippet for each project. We *smooth* each suspiciousness score by averaging each line's suspiciousness score and the surrounding +/- w lines. After smoothing the suspiciousness scores, we re-evaluate the Hit Rate Per n.

In our experiments, we explore the effect of window size w ranging from 0 to 9. In Figure 2, we show average Exam Scores of overall and multi-line bug scenarios in Defects4J. Due to similar trends between all methods, we present the

TABLE IV

WE COMPARE FL ACCURACY BETWEEN OUR FUSION-BASED METHOD (*Fusion*) AND INDIVIDUAL SBFL METHODS FROM THE THREE CATEGORIES. *TrialID* INDICATES THE SELECTED SBFL METHODS USING THE FIRST LETTER OF EACH TECHNIQUE'S NAME. FOR EXAMPLE, TRIAL OAM SELECTS *ochiai*, *ample*, AND *muse*. *average* REPRESENTS THE AVERAGE PERFORMANCE AMONG THE INDIVIDUAL SBFL METHODS IN EACH TRIAL. *Overall Performance* REFERS TO THE AVERAGE ACCURACY ACROSS ALL BUG SCENARIOS, WHILE *Multiline Performance* REPRESENTS THE AVERAGE ACCURACY IN MULTI-LINE BUG SCENARIOS.

	Overall Performance					Multi-Line Performance				
TrialID	fusion	FS	CC	CFR	average	fusion	FS	CC	CFR	average
OAM	0.0999	**0.0994**	0.1036	0.1105	0.1045	0.1312	**0.1308**	0.1352	0.1462	0.1374
OAP	0.2669	**0.0994**	0.1036	0.3573	0.1868	0.2910	**0.1308**	0.1352	0.3639	0.2100
OGM	0.1270	**0.0994**	0.2073	0.1105	0.1391	0.1522	**0.1308**	0.2279	0.1462	0.1683
OGP	0.3078	**0.0994**	0.2073	0.3573	0.2213	0.3259	**0.1308**	0.2279	0.3639	0.2409
TAM	**0.0978**	0.0995	0.1036	0.1105	0.1045	**0.1259**	0.1305	0.1352	0.1462	0.1373
TAP	0.2519	**0.0995**	0.1036	0.3573	0.1868	0.2734	**0.1305**	0.1352	0.3639	0.2099
TGM	0.1206	**0.0995**	0.2073	0.1105	0.1391	0.1445	**0.1305**	0.2279	0.1462	0.1682
TGP	0.2744	**0.0995**	0.2073	0.3573	0.2214	0.2911	**0.1305**	0.2279	0.3639	0.2408
DAM	0.1041	0.1064	**0.1036**	0.1105	0.1068	0.1370	0.1423	**0.1352**	0.1462	0.1413
DAP	0.2515	0.1064	**0.1036**	0.3573	0.1891	0.2766	0.1423	**0.1352**	0.3639	0.2138
DGM	0.1263	**0.1064**	0.2073	0.1105	0.1414	0.1545	**0.1423**	0.2279	0.1462	0.1722
DGP	0.2783	**0.1064**	0.2073	0.3573	0.2237	0.3037	**0.1423**	0.2279	0.3639	0.2447
ZAM	0.1013	**0.0993**	0.1036	0.1105	0.1045	0.1338	**0.1304**	0.1352	0.1462	0.1373
ZAP	0.2727	**0.0993**	0.1036	0.3573	0.1867	0.2953	**0.1304**	0.1352	0.3639	0.2098
ZGM	0.1300	**0.0993**	0.2073	0.1105	0.1300	0.1568	**0.1304**	0.2279	0.1462	0.1682
ZGP	0.3252	**0.0993**	0.2073	0.3573	0.2213	0.3437	**0.1304**	0.2279	0.3639	0.2407
JAM	**0.1036**	0.1059	0.1036	0.1105	0.1067	0.1366	0.1418	**0.1352**	0.1462	0.1411
JAP	0.2502	0.1059	**0.1036**	0.3573	0.1890	0.2745	0.1418	**0.1352**	0.3639	0.2136
JGM	0.1256	**0.1059**	0.2073	0.1105	0.1412	0.1534	**0.1418**	0.2279	0.1462	0.1720
JGP	0.2762	**0.1059**	0.2073	0.3573	0.2235	0.3002	**0.1418**	0.2279	0.3639	0.2445

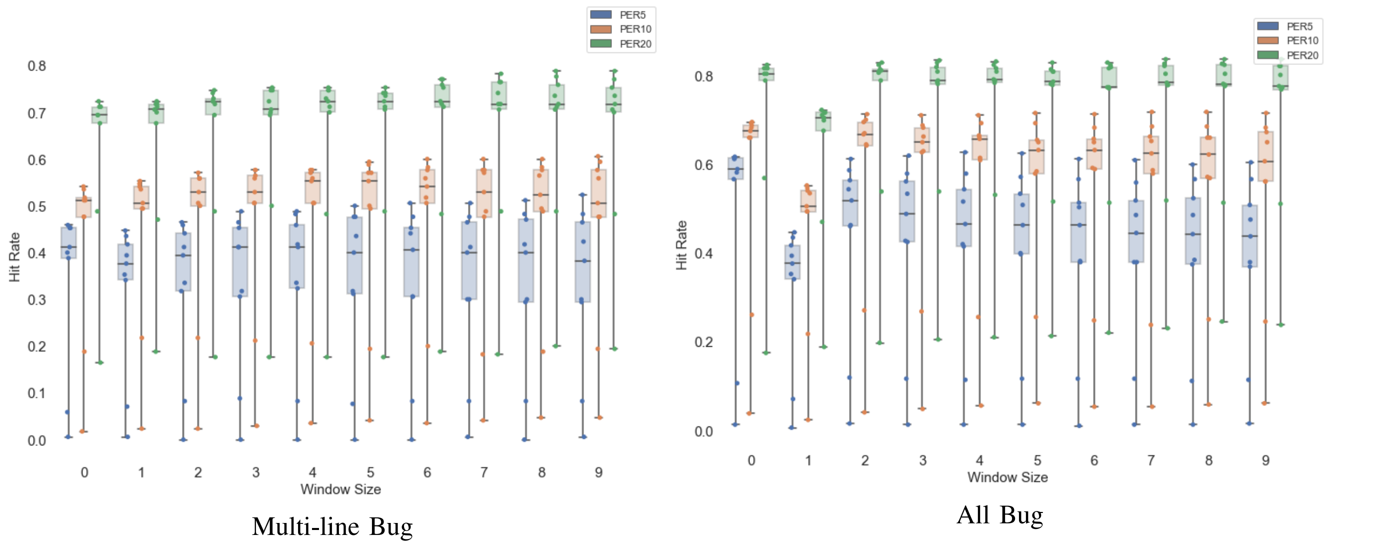

Fig. 3. Hit Rate Per n for all SBFL methods with window size w from 0–9 applied to Defects4J.

comparison of FL accuracy between the maximum, mean, and minimum value among 9 SBFL methods (because it represents the best average FL performance) across different window sizes. All the other evaluation data are available in data link. We see that all the best fuzzy-window approaches outperform *ochiai* (recall that lower is better for Exam Score). The best overall performance is obtained with window size $w = 4$, which yields a decrease of exam score by **5.5%** compared to *ochiai*. Furthermore, in the multi-bug dataset, using Zoltar with window size $w = 9$ requires inspecting **15.7%** fewer lines of code than the best standalone SBFL.

We also present the comparison of Hit Rate Per n in Figure 3. Each dot represents the performance of a single SBFL formula with different window sizes. In all of the PER5, PER10, and PER20 evaluations, our fuzzy-window approach outperforms *ochiai* (recall that higher is better for PERn). In Figure 3, the Hit Rate increases as the window size becomes larger in multi-line bugs. The best peformance is achieved when window size $w = 9$ in Zoltar, resulting in an increase in Hit Rate of **15%**, **17.2%**, **8.8%** for Hit rate Per5, Per10, and Per20 compared to *ochiai*. In Figure 3, the highest Hit Rate is achieved with window size $w = 4$ and 7 across the whole

TABLE V
COMPARISON WITH STATE-OF-THE-ART INDIVIDUAL AND ML-BASED FUSION APPROACHES ON THE METHOD-LEVEL FL, WITH MULTI-LINE FL PERFORMANCE GIVEN IN PARENTHESES.

FL Method	Top-1	Top-3	Top-5	MFR	MAR
Ochiai	80(23)	165(75)	196(96)	38(46)	43(49)
Multric	80(25)	163(81)	195(104)	38(32)	44(51)
Zoltar_win9	102(47)	176(93)	215(114)	76(65)	101(95)
Zoltar_win4	122(60)	190(101)	226(123)	55(47)	92(92)

dataset. They outperform all single SBFL formulae in PER5, PER10, and PER20.

C. RQ3: Fusing Uncertain and Ambiguous FL Techniques

In this section, we first apply our fuzzy-window approach to smooth ambiguous suspiciousness scores, then fuse together the resulting values.

We compare our combined *fuzzy-window fusion* approach with the most widely-used individual SBFL formulae. In the statement level, we present a comparison with the best three combination-only localizers in [12] — we reproduced their methods on Defects4J: $F^{zero-one,overlap}_{Comb_{ANZ}}$: Computing the average of the non-zero scores from Overlap-Based Selection with zero-one score normalization for all SBFL results. $F^{zero-one,bias}_{Comb_{ANZ}}$: Computing the average of non-zero scores from Bias-Based Selection with zero-one score normalization for all SBFL results. $F^{zero-one,bias}_{Comb_{SUM}}$: Computing the sum of scores from Bias-Based Selection with zero-one score normalization for all SBFL results.

In Table VII and Table VI, we include our two top-performing fuzzy-window fusion approaches: ZAM_win9: fusion between *zoltar*, *ample* and *muse* with fuzzy-window (window size=9). zoltar_win$n$: the original *zoltar* with fuzzy-window (window size=n).

As shown in Table VII, our methods substantially outperform individual SBFL techniques and the state-of-the-art localizers. Across both Exam Score and Hit Rate Per n for *zoltar_win4* and *zoltar_win7*, we substantially improve fault localization compared to individual SBFL formulae.

In Table VI, our combined fuzzy window and fusion approach obtains even better performance on the multi-line bug scenarios. In particular, *zoltar_win9* obtains a 35.6% improvement in Exam Score and 52.2% improvement in Hit Rate Per 5%. *ZAM_win9* also improves Hit Rate Per 10% by 43.2%. These results show that our approaches outperform state-of-the-art localization techniques.

D. Evaluation Conclusion

Our evaluation encompassed the application of 9 different state-of-the-art SBFL formulae to the Defects4J benchmark suite. We explored how we can use information fusion techniques to combine suspiciousness scores produced by disparate formulae to improve overall suspiciousness rankings. We demonstrated measurable improvements in two key metrics in statement level: (1) Exam Score and (2) Hit Rate Per n. These metrics represent the engineering effort required to search for actual defects in the code. Therefore, our approach of smoothing and fusing suspiciousness scores substantially enhances fault localization.

Furthermore, we also provide the comparison against the state of art ML-based method-level fusion approach (i.e., Multric) after adjusting our statement-level fault localization, using 5 widely applied metrics: Top-1, Top-3, Top-5, MFR, and MAR in Table V (see Section IV for more details on the metrics and adjustment for comparison). Meanwhile, we also include Ochiai in the comparison to provide an intuition on the method-level performance of the traditional SBFL. Despite using fewer computation resources than Multric, our method has a better accuracy on the top ranked buggy methods (i.e., *zoltar_win9* and *zoltar_win4* identify more buggy methods in the Top-1, 3, and 5 evalution.), at the cost of a reduced accuracy on average ranks for all buggy methods (i.e., higher MFR and MAR). Considering that previous research has shown developers most likely only review the top 5 suspicious methods [16], [59], a better top n performance is more likely to assist developers in real world practice.

All of our code and data, including all fault localization and fusion experimental results, are publicly released.

V. DISCUSSION

In this section, we discuss how our fuzzy window approach may inspire future work in multiple fault localization. Finally, we discuss threats to validity of our experiments.

Modeling uncertainty and ambiguity in FL. In our approach, fusing localization information can reduce the uncertainty in standalone SBFL formulae, and our fuzzy window method addresses ambiguity among defects in highly suspicious areas. Our approaches outperform existing SBFL, especially on multi-line fault localization. The improvement is more significant when we apply both approaches together.

Fuzzy windows. Moreover, our window-based suspiciousness calculation represents a larger range of program element fault evaluation that addresses ambiguity in localization. In SBFL formulae, suspiciousness is usually computed with respect to functions or lines. These suspiciousness scores depend not only on uncertainty among SBFL formulae, but also on ambiguity of suspiciousness among nearby lines of code.

Threats to validity. We compared multi-line and single-line defect localization, focusing on a dataset with both types of defects (i.e., we did not pick a dataset containing only multi-line defects). Next, our evaluation was limited only to Java code (i.e., the Defects4J benchmark). Moreover, we consider only three categories of SBFL techniques, but it is possible that there are more accurate formulae or ecologically valid datasets not in our evaluation. It is important to consider that different combinations may yield varying results in datasets with diverse test coverage or fault frequency.

VI. CONCLUSION

In this paper, we present new approaches to combine Spectrum-Based Fault Localization techniques to improve localization accuracy. By leveraging evidence theory, we can treat sets of suspiciousness scores produced by different SBFL

TABLE VI
EVALUATION FOR FL PERFORMANCE FOR MULTI-LINE BUG SCENARIOS AMONG INDIVIDUAL STATE-OF-THE-ART SBFL AND OUR TWO TOP-PERFORMING FUZZY-WINDOW FUSION APPROACHES (SHOWN IN BOLD) APPLIED TO DEFECTS4J. WE ALSO LIST THE INCREASE IN PERFORMANCE FROM OUR METHODS (SHOWN IN PARENTHESES) COMPARED TO THE AVERAGE OF THE INDIVIDUAL SBFL METHODS (SHOWN AS *Average*). RECALL THAT A DECREASE IN EXAM SCORE REFLECTS AN IMPROVEMENT, WHILE AN INCREASE IN HIT RATE PER n REFLECTS AN IMPROVEMENT.

FL Method	Mean_Exam	Per5	Per10	Per20
ochiai	0.1308	0.4551	0.5169	0.7247
tarantula	0.1305	0.4551	0.5337	0.7247
dstar2	0.1423	0.3933	0.4831	0.7079
zoltar	0.1304	0.4607	0.5281	0.7247
jaccard	0.1418	0.4045	0.4831	0.7079
ample	0.1352	0.4101	0.5393	0.7360
gp02	0.2279	0.0562	0.1966	0.5056
muse	0.1462	0.4551	0.5506	0.6854
Piatetsky_Shapiro	0.3639	0.0056	0.0169	0.1573
Average	0.1721	0.3439	0.4276	0.6305
$F_{Comb_{ANZ}}^{zero-one,overlap}$ [12]	0.2172	0.0730	0.2135	0.5562
$F_{Comb_{ANZ}}^{zero-one,bias}$ [12]	0.2157	0.0787	0.2303	0.5618
$F_{Comb_{SUM}}^{zero-one,bias}$ [12]	0.2177	0.2753	0.3652	0.5281
ZAM_win9	0.1184 (30.8%)	0.4888 (42.1%)	0.6124 (43.2%)	0.7640 (21.1%)
zoltar_win9	0.1103 (35.6%)	0.5235 (52.2%)	0.6059 (41.6%)	0.7882 (25%)

TABLE VII
EVALUATION OF OVERALL FL PERFORMANCE AMONG INDIVIDUAL SBFL AND TWO OF OUR TOP-PERFORMING FUZZY-WINDOW FUSION APPROACHES (SHOWN IN BOLD) APPLIED TO DEFECTS4J. WE ALSO SHOW THE INCREASE IN PERFORMANCE OF THE PROPOSED METHODS (SHOWN IN PARENTHESIS) COMPARED TO THE AVERAGE INDIVIDUAL SBFL METHODS (SHOWN AS *Average*). RECALL THAT A DECREASE IN EXAM SCORE IS AN IMPROVEMENT, AND AN INCREASE IN PERn IS AN IMPROVEMENT.

FL Method	Mean_Exam	Per5	Per10	Per20
ochiai	0.0994	0.6152	0.6835	0.8177
tarantula	0.0995	0.6127	0.6937	0.8177
dstar2	0.1064	0.5823	0.6608	0.8051
zoltar	0.0993	0.6177	0.6886	0.8177
jaccard	0.1059	0.5899	0.6608	0.8051
ample	0.1036	0.5671	0.6759	0.8253
gp02	0.2073	0.1063	0.2608	0.5696
muse	0.1105	0.6152	0.6962	0.7899
Piatetsky_Shapiro	0.3573	0.0127	0.0380	0.1747
Average	0.1433	0.4799	0.5620	0.7136
$F_{Comb_{ANZ}}^{zero-one,overlap}$ [12]	0.1989	0.1316	0.2759	0.6051
$F_{Comb_{ANZ}}^{zero-one,bias}$ [12]	0.1963	0.1367	0.2886	0.6101
$F_{Comb_{SUM}}^{zero-one,bias}$ [12]	0.2261	0.3139	0.3873	0.5291
zoltar_win4	0.0939 (34.5%)	0.6278 (30.1%)	0.7114 (26.6%)	0.8329 (16.7%)
zoltar_win7	0.0957 (33.2%)	0.6101 (27.1%)	0.719 (27.9%)	0.838 (17.4%)

techniques as different sources of evidence that can be fused together. Doing so accounts for uncertainty between different SBFL techniques. Moreover, we introduce a fuzzy window approach to redistributing suspiciousness scores among nearby lines of code. In doing so, we resolve ambiguity in localization among nearby lines of code.

We evaluate our approach using the Defects4J dataset. We demonstrate that fusing raw SBFL scores can result in better localization accuracy in some situations. Next, our fuzzy window approach can reduce the amount of code inspected by as much as 34.5%, and furthermore provides up to a 27.9% increase in reporting the top 10% buggy lines compared to ochiai localization. Our combined fuzzy window fusion approach achieves an improvement in Exam score and hit rate compared to state-of-the-art fault localization work. In multi-line defect localization, the hit rate for the top 10% of lines increases from 55% to 61.2%, and inspected code drops from 13.1% to 11% compared to the best single methods. In the method-level evaluation, our approach outperforms the state-of-the-art ML-based fusion approach by 52.5% on the Top-1 accuracy. For scenarios with a high bug frequency, a larger window size like *zoltar_win9* is advisable as it can accommodate more bugs. However, if the focus is on localizing bugs within the top k rather than overall performance, choosing *zoltar_win4* is recommended. We further explore the suitability of different window sizes and fusion approaches for various scenarios.

References

[1] H. Krasner, "The cost of poor software quality in the us: A 2020 report," *Proc. Consortium Inf. Softw. QualityTM (CISQTM)*, 2021.

[2] A. Alaboudi and T. D. LaToza, "An exploratory study of debugging episodes," *arXiv preprint arXiv:2105.02162*, 2021.

[3] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," in *2016 6th international conference on information and communication technology for the Muslim world (ICT4M)*. IEEE, 2016, pp. 177–182.

[4] 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.

[5] C. Le Goues, S. Forrest, and W. Weimer, "Current challenges in automatic software repair," *Software quality journal*, vol. 21, no. 3, pp. 421–443, 2013.

[6] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 364–374.

[7] C. Le Goues, W. Weimer, and S. Forrest, "Representations and operators for improving evolutionary software repair," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 2012, pp. 959–966.

[8] 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.

[9] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "On the accuracy of spectrum-based fault localization," in *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 2007, pp. 89–98.

[10] ——, "Spectrum-based multiple fault localization," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009, pp. 88–99.

[11] L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi, "Extended comprehensive study of association measures for fault localization," *Journal of software: Evolution and Process*, vol. 26, no. 2, pp. 172–219, 2014.

[12] D. Lo and X. Xia, "Fusion fault localizers," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 127–138.

[13] T.-D. B. Le, D. Lo, C. Le Goues, and L. Grunske, "A learning-to-rank based fault localization approach using likely invariants," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 177–188.

[14] J. Xuan and M. Monperrus, "Learning to combine multiple ranking metrics for fault localization," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 191–200.

[15] 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.

[16] X. Li, W. Li, Y. Zhang, and L. Zhang, "Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 169–180.

[17] H. Ziv, D. Richardson, and R. Klösch, "The uncertainty principle in software engineering," in *submitted to Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, 1997.

[18] R. Pelissari, M. C. Oliveira, A. J. Abackerli, S. Ben-Amor, and M. R. P. Assumpção, "Techniques to model uncertain input data of multi-criteria decision-making problems: a literature review," *International Transactions in Operational Research*, vol. 28, no. 2, pp. 523–559, 2021.

[19] Y. Zhang and Y. Huang, "Leveraging fuzzy system to reduce uncertainty of decision making in software engineering automation," in *GECCO 2022 Workshop 11th International Workshop on Genetic Improvement*. ACM, 2022, pp. 130–140.

[20] P. S. M. dos Santos and G. H. Travassos, "On the representation and aggregation of evidence in software engineering: A theory and belief-based perspective," *Electronic notes in theoretical computer science*, vol. 292, pp. 95–118, 2013.

[21] M. Zabihimayvan and D. Doran, "Fuzzy rough set feature selection to enhance phishing attack detection," in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2019, pp. 1–6.

[22] T. Rutkowski, J. Romanowski, P. Woldan, P. Staszewski, R. Nielek, and L. Rutkowski, "A content-based recommendation system using neuro-fuzzy approach," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2018, pp. 1–8.

[23] N. Napoli, K. Leach, L. Barnes, and W. Weimer, "A mapreduce framework to improve template matching uncertainty," 01 2016.

[24] 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.

[25] A. P. Dempster, "Upper and lower probabilities induced by a multivalued mapping," in *Classic works of the Dempster-Shafer theory of belief functions*. Springer, 2008, pp. 57–72.

[26] G. Shafer, *A mathematical theory of evidence*. Princeton university press, 1976, vol. 42.

[27] C. Kahraman, S. C. Onar, and B. Oztaysi, "Fuzzy multicriteria decision-making: a literature review," *International journal of computational intelligence systems*, vol. 8, no. 4, pp. 637–666, 2015.

[28] N. Naik, P. Jenkins, R. Cooke, and L. Yang, "Honeypots that bite back: A fuzzy technique for identifying and inhibiting fingerprinting attacks on low interaction honeypots," in *2018 IEEE International Conference on fuzzy systems (FUZZ-IEEE)*. IEEE, 2018, pp. 1–8.

[29] P. Jamadi Khiabani, M. E. Basiri, and H. Rastegari, "An improved evidence-based aggregation method for sentiment analysis," *Journal of Information Science*, vol. 46, no. 3, pp. 340–360, 2020.

[30] R. Oruche, V. Gundlapalli, A. P. Biswal, P. Calyam, M. L. Alarcon, Y. Zhang, N. R. Bhamidipati, A. Malladi, and H. Regunath, "Evidence-based recommender system for a covid-19 publication analytics service," *Ieee Access*, vol. 9, pp. 79 400–79 415, 2021.

[31] H. Lee and H. Kwon, "Dbf: Dynamic belief fusion for combining multiple object detectors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1499–1514, 2019.

[32] V. V. Baba and F. HakemZadeh, "Toward a theory of evidence based decision making," *Management decision*, 2012.

[33] X. Pan, H. Wang, and W. Chang, "A fuzzy synthetic evaluation method for failure risk of aviation product r&d project," in *2010 IEEE International Conference on Management of Innovation & Technology*. IEEE, 2010, pp. 1106–1111.

[34] X. Su, S. Mahadevan, P. Xu, and Y. Deng, "Dependence assessment in human reliability analysis using evidence theory and ahp," *Risk Analysis*, vol. 35, no. 7, pp. 1296–1316, 2015.

[35] Y. Zhang, X. Deng, D. Wei, and Y. Deng, "Assessment of e-commerce security using ahp and evidential reasoning," *Expert Systems with Applications*, vol. 39, no. 3, pp. 3611–3623, 2012.

[36] D. Dubois and H. Prade, "Representation and combination of uncertainty with belief functions and possibility measures," *Computational intelligence*, vol. 4, no. 3, pp. 244–264, 1988.

[37] C. K. Murphy, "Combining belief functions when evidence conflicts," *Decision support systems*, vol. 29, no. 1, pp. 1–9, 2000.

[38] R. R. Yager, "On the dempster-shafer framework and new combination rules," *Information sciences*, vol. 41, no. 2, pp. 93–137, 1987.

[39] 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.

[40] T. Janssen, R. Abreu, and A. J. Van Gemund, "Zoltar: A toolset for automatic fault localization," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009, pp. 662–664.

[41] 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 (TOSEM)*, vol. 22, no. 4, pp. 1–40, 2013.

[42] A. Zeller, "Isolating cause-effect chains from computer programs," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 6, pp. 1–10, 2002.

[43] M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," in *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.* IEEE, 2003, pp. 30–39.

[44] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff, "Sober: statistical model-based bug localization," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 5, pp. 286–295, 2005.

[45] J. Jiang, R. Wang, Y. Xiong, X. Chen, and L. Zhang, "Combining spectrum-based fault localization and statistical debugging: An empirical study," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 502–514.

[46] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating & improving fault localization techniques," *University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-16-08-03*, 2016.

[47] J. Tu, X. Xie, T. Y. Chen, and B. Xu, "On the analysis of spectrum based fault localization using hitting sets," *Journal of Systems and Software*, vol. 147, pp. 106–123, 2019.

[48] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of ir-based fault localization techniques," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 1–11.

[49] X. Xue and A. S. Namin, "How significant is the effect of fault interactions on coverage-based fault localizations?" in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2013, pp. 113–122.

[50] A. Zakari, S. P. Lee, R. Abreu, B. H. Ahmed, and R. A. Rasheed, "Multiple fault localization of software programs: A systematic literature review," *Information and Software Technology*, vol. 124, p. 106312, 2020.

[51] A. Zakari and S. P. Lee, "Parallel debugging: An investigative study," *Journal of Software: Evolution and Process*, vol. 31, no. 11, p. e2178, 2019.

[52] C. Gong, Z. Zheng, Y. Zhang, Z. Zhang, and Y. Xue, "Factorising the multiple fault localization problem: Adapting single-fault localizer to multi-fault programs," in *2012 19th Asia-Pacific Software Engineering Conference*, vol. 1. IEEE, 2012, pp. 729–732.

[53] V. Debroy and W. E. Wong, "Insights on fault interference for programs with multiple bugs," in *2009 20th International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 165–174.

[54] Y. Xiaobo, B. Liu, and W. Shihai, "An analysis on the negative effect of multiple-faults for spectrum-based fault localization," *IEEE Access*, vol. 7, pp. 2327–2347, 2018.

[55] P. Agarwal and A. P. Agrawal, "Fault-localization techniques for software systems: A literature review," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 5, pp. 1–8, 2014.

[56] M. Duran, X.-Y. Zhang, P. Arcaini, and F. Ishikawa, "What to blame? on the granularity of fault localization for deep neural networks," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 264–275.

[57] M. Aruldoss, T. M. Lakshmi, and V. P. Venkatesan, "A survey on multi criteria decision making methods and its applications," *American Journal of Information Systems*, vol. 1, no. 1, pp. 31–43, 2013.

[58] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, "Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1936–1964, 2017.

[59] 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.