Exploring the Characteristics of Spectra Distribution and Their Impacts on Fault Localization

Xiao-Yi Zhang*
National Institute of Informatics
Tokyo, Japan
xiaoyi@nii.ac.jp

Zheng Zheng
School of Automation Science and Electronic Engineering,
Beihang University, BUAA
Beijing, China
zhengz@buaa.edu.cn

ABSTRACT

Spectrum-Based Fault Localization (SBFL) follows the basic intuitions that the faulty parts are more likely to be covered by failure-revealing test cases and less likely to be covered by passed test cases. However, due to the diversity of programs and faults, many other characteristics (related to program structure, test suites, and type of faulty components) will influence the practical application of SBFL. For example, a statement can be covered by numerous failure-revealing test cases, and also covered by numerous passed test cases. To get more indicators about the faulty components towards a better application of SBFL, we extend the scope of spectrum-based knowledge from the basic intuitions to the *Characteristics of Spectra Distribution* (*CSD*s for short). That is, we explore the relationships between different types of statements and their spectra. Firstly, we introduce the concepts of *Failure-Independent*, *Failure-Related*, and *Failure-Exclusionary* to describe the relationships between different types of statements and their executions. Then, we propose two probabilistic models, with and without the *noise* of fault interference, respectively, to identify various *CSD*s for each type of statements. As the analysis results, we introduce a visualization technique to generalize the identified *CSD*s and provide an overall picture of spectra distribution and its dynamics. Finally, based on our analysis and also the observation of the program spectra of current benchmarks, we design a technique to filter the potential non-faulty statements to improve the accuracy of SBFL.

CCS CONCEPTS

• **Software and its engineering** → Software creation and management; *Software verification and validation*; *Software defect analysis*; **Software testing and debugging**.

KEYWORDS

software fault localization, spectrum-based fault localization, probabilistic model, spectrum-based characteristics, visualization

*Xiao-Yi Zhang is the corresponding author.

EASE'20, April 15–17, Trondheim, Norway

ACM ISBN 978-1-4503-7731-7/20/04…$15.00
https://doi.org/10.1145/3383219.3383230

ACM Reference Format:
Xiao-Yi Zhang and Zheng Zheng. 2020. Exploring the Characteristics of Spectra Distribution and Their Impacts on Fault Localization. In *Evaluation and Assessment in Software Engineering (EASE 2020), April 15–17, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3383219.3383230

1 INTRODUCTION

Software fault localization [22] aims to find the faulty components that cause the program failures detected during testing. To date, various studies pointing to the automation of fault localization process have been conducted [2, 13, 15]. One of the most famous fault localization techniques is Spectrum-Based Fault Localization (SBFL) [23, 26, 28]. Here, a program spectrum refers to a collection of information that provides the characteristics of a specific program component [23]. SBFL takes the program spectra information derived during testing to find the suspicious faulty components of the subject program. To date, studies on SBFL have been conducted from various aspects, such as different component particles [2, 20, 29], suspiciousness metrics [16, 21, 23, 26], and SBFL-orient testing techniques [7, 25]. Note that we take a program statement as the basic unit of component in this paper.

Generally, SBFL is based on two intuitions [28]:

(1) statements covered by more failure-revealing test cases are more likely to be faulty, and
(2) statements covered by more passed test cases are less likely to be faulty.

The above intuitions consider the passed and failure-revealing test cases separately. Nevertheless, in practice, the fault propagation process is complicated. Only considering these intuitions could be insufficient. For example, even if a test case is failure-revealing, it may cover many non-faulty components. Also, even if a faulty statement has been executed, it may not cause failures (i.e., the coincidental correctness phenomenon [14]). Furthermore, if the program has multiple faults interacting with each other, the spectrum of a specific faulty statement may become more difficult to be identified (i.e., the multi-fault problem [5]). Although various SBFL-orient techniques have been proposed [1, 16], the challenge of providing more knowledge of program spectra to evaluate the practical application of SBFL always exists.

In fact, except for the basic intuitions, there are also various characteristics related to the distribution of program spectra and can be useful to improve SBFL. For example, statements in the *main* function are likely to be covered by both passed and failure-revealing

test cases. Also, to meet the safety requirements, redundant modules are usually introduced [10]. Then for such modules, they are rarely covered by both the passed and failure-revealing test cases.

However, exploring and generalizing such characteristics are challenging. As SBFL involves various artifacts, including the test suite, and different types of statements, the *Characteristics of Spectra Distribution* (*CSD*s for short) could also be complicated and be related to multiple and heterogeneous artifacts (e.g., the property of statements and the test coverage information). Moreover, the interactions among these artifacts are also complicated.

To pave the way from basic intuitions towards comprehensive spectrum-related knowledge and get a better understanding and evaluation of the practical application of SBFL, this paper proposes an appraoch based on probabilistic modeling and data visualization to conduct an overall analysis of *CSD*s beyond the basic intuitions of SBFL. Specifically, we focus on which types of statements have which shapes of spectra. Firstly, we identify the program statements as three types: *Failure-Related*, *Failure-Exclusionary*, and *Failure-Independent*, referring to the statements whose executions are relevant, exclusionary and independent to the manifestation of program failures, respectively. Then, we propose a probabilistic model to formalize the behaviors among different types of statements and program failures, and summarize them as specific *CSD*s.

Particularly, we further consider the influence of fault interference in our model, under the assumption that different faulty elements are independent. Specifically, the influence of different faults is described as the *noise*. As a result, we derive some common points about the influence of the *noise* on different types of statements and describe them as the dynamics of *CSD*s. This work is our primary step to explore formal descriptions for the instances involving more than one faulty elements.

To generalize the analysis results, we propose an approach based on visualization that presents program statements in the entire spectra space. Through visualization, we integrate various identified *CSD*s belonging to different types of statements in an intuitive way, and then evaluate their influences on the performance of SBFL. Finally, as an example to put our analysis into practice, we propose a technique that identifies the potential non-faulty statements based on the identified *CSD*s and eliminates them during SBFL.

The remainder of this paper is organized as follows. The background of SBFL is introduced in Section 2. The modeling of different types of statements, and the exploration of *CSD*s are presented in Section 3. The visualization of the derived results is presented in Section 4. Based on visualization, we design a filtering technique to improve SBFL in Section 5. An experimental study is conducted in Section 6. Related works are discussed in Section 7. In Section 8, we make conclusions and discuss the future works.

2 BACKGROUND: SPECTRUM-BASED FAULT LOCALIZATION

Given the subject program PG, with the set of statements $S = \{s_1, s_2, \ldots, s_n\}$. Assume PG has a faulty statement s^f, then the basic task of fault localization is to find s^f in S. Here let $s^f \leftarrow s$ denote that statement s is faulty. Now concerning Spectrum-Based Fault Localization (SBFL), given a test suite $T = \{t_1, t_2, \ldots, t_m\}$, we let C^t denote the set of statements covered by t and $o^t \in \{pass, fail\}$ denote the correctness of t. Thus, based on T, considering the coverage of each $s \in S$, we can calculate the following quantities:

- $a_{ef}(s)$: the number of *failure-revealing* test cases that *cover* statement s, i.e., $|\{t \in T | s \in C^t, o^t = fail\}|$
- $a_{ep}(s)$: the number of *passed* test cases that *cover* statement s, i.e., $|\{t \in T | s \in C^t, o^t = pass\}|$
- $a_{nf}(s)$: the number of *failure-revealing* test cases that *do not cover* statement s, i.e., $|\{t \in T | s \notin C^t, o^t = fail\}|$
- $a_{np}(s)$: the number of *passed* test cases that *do not cover* statement s, i.e., $|\{t \in T | s \notin C^t, o^t = pass\}|$

In addition, let F and P denote the total number of the *failure-revealing* and *passed* test cases, respectively.

The basic idea of SBFL is to predict the likelihood of being faulty for each $s \in S$. Specifically, the suspiciousness degree that $s^f \leftarrow s$ can be calculated by a formula R composed of the above quantities. In this sense, the basic intuition of SBFL can be interpreted as:

(1) the larger $a_{ef}(s)$ value that statement s has, the more suspicious it is, and
(2) the smaller $a_{ep}(s)$ value that statement s has, the more suspicious it is.

Currently, various metrics have been proposed following the above two intuitions. Also, different metrics are based on their own heuristics to arrange the composition of $a_{ef}(s)$ and $a_{ep}(s)$. One of the earliest metric, *Tarantula* [8], denoted by R_τ is designed as follows:

$$R_\tau(s) = \frac{a_{ef}(s)/F}{a_{ef}(s)/F + a_{ep}(s)/P} \tag{1}$$

Subsequently, it has been observed that failure-revealing test cases are more important to expose the faulty statements. Based on this observation, Abreu et al., [1, 2] proposed the *Ochiai* metric, denoted by R_O, expressed as follows:

$$R_O(s) = \frac{a_{ef}(s)}{\sqrt{(a_{ef}(s) + a_{nf}(s))(a_{ef}(s) + a_{ep}(s))}} \tag{2}$$

Furthermore, Naish et al. [16] proposed a famous metric called *Op*, denoted by $R_{\mathscr{O}}$, which put the term $a_{ef}(s)$ at a dominant position. The expression of $R_{\mathscr{O}}$ is shown as follows:

$$R_{\mathscr{O}}(s) = a_{ef}(s) - a_{ep}(s)/(P+1) \tag{3}$$

Note that although various metrics as been proposed, such as the families of DStars[21] and GPs [26], current studies [30] and also our ongoing works has observed that some metrics, although they may not be equivalent, have similar performance (e.g., R_O and DStars, and the family of continuous maximal metrics) [27]. Thus, according to our knowledge, this paper will consider R_τ, R_O, and $R_{\mathscr{O}}$ as representative. After the calculation based on the selected metric, SBFL ranks the statements according to their suspiciousness degrees in descending order and provides the rank list as output. If it is accurate, the faulty statement s^f will be ranked ahead and then can be found earlier by the testers.

3 PROBABILISTIC MODEL FOR STATEMENTS' SPECTRA AND PROGRAM FAILURES

3.1 Three Types of Statements

In this section, we categorize the program statements into three types according to how their spectra can be affected by program failures: the *Failure-Independent* statements, the *Failure-Related* Statements, and the *Failure-Exclusionary* Statements.

Definition 1. **Failure-Independent *Statement.*** *If the execution of statement s is not correlated with the manifestation of program failures, s will be considered as Failure-Independent, denoted by* $s^{\mathcal{I}} \leftarrow s$.

Example 3.1.

- *If statement s is included in the* main *function of PG, then it will be covered by all the test cases, whether these test cases are failure-revealing or not.*

Definition 2. **Failure-Related *Statements.*** *If the execution of s is correlated with the manifestation of program failures, then s will be considered as Failure-Related, denoted by* $s^{\mathcal{R}} \leftarrow s$.

Example 3.2.

(1) *If statement s belongs to the function that triggers the execution of the faulty statement* s^f*, then the execution of s may cause a higher failure rate.*
(2) *If statement s belongs to the module that proceeds a specific type of system exception caused by* s^f*, then the execution of s will lead to a higher failure rate.*

Definition 3. **Failure-Exclusionary *Statement.*** *If the execution of statement s leads to a lower probability of failure, then s is considered as Failure-Exclusionary, denoted by* $s^{\mathcal{E}} \leftarrow s$.

Example 3.3.

- *If modules A and B belong to different branches of program PG, while module B contains the faulty statement* s^f*, then statements in A could have a relatively lower failure rate.*

3.2 Probabilistic Model without Noises

To analyze the relationship between the execution of each statement $s \in S$ and the manifestation of program failures, we make the following assumptions.

Assumption 1. *The faulty statement* $s^f \in S$ *exists.*

Assumption 2. *PG is deterministic. That is, a failure-revealing test case t will definitely cause failures whenever it is executed.*

Assumption 3. *The test suite T has already been given and we know both its coverage and correctness information.*

Assumption 4. *The test cases in T are generated randomly following a common profile*[1].

Based on these assumptions, the relationship between the execution of a statement s and the manifestation of program failures can be modeled as a discrete Markov Chain, illustrated in Fig. 1.

[1]Here, we do not expect an entirely random testing. Instead, we assume that the generation of test cases follows a common rule of randomness (e.g., pure random testing or proportional sampling) and does not rely on fault-based knowledge.

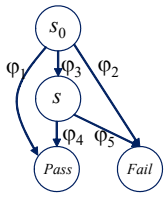


Figure 1: The basic model for the relationship between the execution of statement s and the manifestation of program failures.

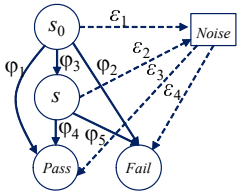


Figure 2: The model of statement-failure relationship involving the *noise.*

Fig. 1 presents the simplified transition model from the program entrance to the final output, in which statement s is treated as an intermediate state[2]. When executing a test case t, the execution trace will start from the program entry s_0 and have a probability to cover statement s. Finally, it produces an output with a certain probability of failure. Specifically, for statement s and test case t, the probability of each transitions in Fig. 1 is presented below.

- φ_1^s: the probability that t *does not cover* statement s and *gets passed*, i.e., $\mathcal{P}(s \notin C^t, o^t = pass)$
- φ_2^s: the probability that t *does not cover* statement s and *reveals a failure*, i.e., $\mathcal{P}(s \notin C^t, o^t = fail)$
- φ_3^s: the probability that t *covers* statement s, i.e., $\mathcal{P}(s \in C^t)$
- φ_4^s: the probability that t *gets passed*, under the condition that t *covers* statement s, i.e., $\mathcal{P}(o^t = pass | s \in C^t)$
- φ_5^s: the probability that t *reveals a failure*, under the condition that t *covers* statement s, i.e., $\mathcal{P}(o^t = fail | s \in C^t)$

Here $\mathcal{P}(\cdot)$ represents the meaning of probability. In addition, the following relations should be satisfied:

$$\begin{aligned} \varphi_1^s + \varphi_2^s + \varphi_3^s = 1,\ \varphi_4^s + \varphi_5^s = 1 \\ \varphi_1^s + \varphi_2^s = \mathcal{P}(s \notin C^t) \\ \varphi_1^s + \varphi_4^s \cdot \varphi_3^s = \mathcal{P}(o^t = pass) \\ \varphi_2^s + \varphi_5^s \cdot \varphi_3^s = \mathcal{P}(o^t = fail) \end{aligned} \tag{4}$$

Now suppose test suite T is executed based on this model, then we can calculate some expected spectrum values as follows:

$$\begin{aligned} E(P) &= |T|(\varphi_1^s + \varphi_4^s \cdot \varphi_3^s) \\ E(F) &= |T|(\varphi_2^s + \varphi_5^s \cdot \varphi_3^s) \\ E(a_{ef}(s)) &= |T|(\varphi_3^s \cdot \varphi_5^s) \\ E(a_{ep}(s)) &= |T|(\varphi_3^s \cdot \varphi_4^s) \end{aligned} \tag{5}$$

In this way, this probabilistic model, can be regarded as a formal description of the fault propagation process within the domain of program spectra. Then, we make further exploration of each type of statements identified in Section 3.1.

[2]Although there could be other statements that influence the execution of s, from the perspective of s, all these influences can be integrated into the transition from the entrance s_0 to s.

(1) ***Failure-Independent* Statement.** Based on *Definition 1*, a *Failure-Independent Statement* s^I should have

$$\varphi_5^{s^I} = \varphi_2^{s^I} \text{ and } \varphi_4^{s^I} = \varphi_1^{s^I} \tag{6}$$

Then we have

$$\begin{aligned} E(P) &= \varphi_1^{s^I}(1+\varphi_3^{s^I}) \\ E(F) &= \varphi_2^{s^I}(1+\varphi_4^{s^I}) \\ E(a_{ef}(s^I)) &= |T|(\varphi_3^{s^I}\cdot\varphi_2^{s^I}) \\ E(a_{ep}(s^I)) &= |T|(\varphi_3^{s^I}\cdot\varphi_4^{s^I}) \end{aligned} \tag{7}$$

Thus we can derive

$$\begin{aligned} \frac{a_{ef}(s^I)}{a_{ef}(s^I)+a_{ep}(s^I)} &\approx \frac{|T|(\varphi_3^{s^I}\cdot\varphi_5^{s^I})}{|T|(\varphi_3^{s^I}\cdot\varphi_4^{s^I})+|T|(\varphi_3^{s^I}\cdot\varphi_5^{s^I})} \\ = \frac{\varphi_5^{s^I}}{\varphi_4^{s^I}+\varphi_5^{s^I}} &\approx \frac{F}{F+P} \end{aligned} \tag{8}$$

(2) ***Failure-Related* Statements.** Based on *Definition 2*, if a statement $s^{\mathcal{R}}$ is *Failure-Related*, we should have

$$\varphi_5^{s^{\mathcal{R}}} > \varphi_2^{s^{\mathcal{R}}} \text{ and } \varphi_4^{s^{\mathcal{R}}} < \varphi_1^{s^{\mathcal{R}}} \tag{9}$$

and then we can derive that

$$\frac{a_{ef}(s^{\mathcal{R}})}{a_{ef}(s^{\mathcal{R}})+a_{ep}(s^{\mathcal{R}})} > \frac{F}{F+P} \tag{10}$$

(3) ***Failure-Exclusionary* Statements.** Based on *Definition 3*, if a statement $s^{\mathcal{E}}$ is *Failure-Exclusionary*, then we have

$$\varphi_5^{s^{\mathcal{E}}} < \varphi_2^{s^{\mathcal{E}}} \text{ and } \varphi_4^{s^{\mathcal{E}}} > \varphi_1^{s^{\mathcal{E}}} \tag{11}$$

and then we can derive that

$$\frac{a_{ef}(s^{\mathcal{E}})}{a_{ef}(s^{\mathcal{E}})+a_{ep}(s^{\mathcal{E}})} < \frac{F}{F+P} \tag{12}$$

In summery, let $q_e(s) = \frac{a_{ef}(s)}{a_{ep}(s)+a_{ef}(s)}$ denote the failure rate of statement s and let $q = \frac{F}{F+P}$ denote the overall failure rate, then we have the following characteristic.

Characteristic 1. *Based on the* Assumption 1 *to* Assumption 4*, the failure rate of the Failure-Independent, Failure-Related, and Failure-Exclusionary statements are equal, higher, and lower than the overall failure rate, respectively.*

Now assume the program failures are caused by the only faulty statement s^f and not affected by other factors. Then, s^f is the root cause of the detected failures. In this case, all the failure-revealing test cases will cover s^f. Otherwise, if a test case does not cover s^f, the program failures will not be detected. Then we can derive the following characteristic.

Characteristic 2. *Assume there is only one faulty statement s^f. Based on* Assumptions 1 *to* 4*, s^f should be Failure-Related and all the failure-revealing test cases should cover s^f, i.e., we have*

$$\frac{a_{ef}(s^f)}{a_{ef}(s^f)+a_{ep}(s^f)} > \frac{F}{F+P} \tag{13}$$

$$\varphi_2^{s^f} = 0 \tag{14}$$

3.3 Probabilistic Model With Noises

One of the basic assumptions behind the model in Section 3.2 is that the subject program *PG* has only one faulty statement. That is why we can derive Eq. (14). However, in many situations, the subject program contains multiple faulty factors (e.g., incorrect environmental settings or other faulty components). Different faults interact with each other and influence the program spectra used in SBFL. As a result, the accuracy of SBFL could be degraded. It has been observed that, for some SBFL metrics, a slight degree of fault interference can lead to a significant performance degradation.

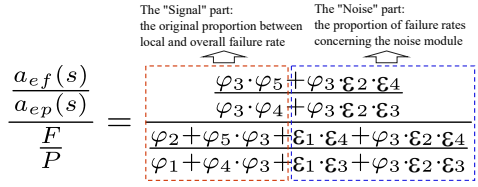
In this work, we focus on improving the explainability of fault interference. Then, we could get more guidances to solve to the multi-fault problems better. Specifically, we conduct a further analysis based on our probabilistic model to explain the occurrence of fault interference and then find some meaningful characteristics.

Here, we model the mechanism of fault interference as a module of the *noise*, denoted by $\mathcal{N}$, as shown in Fig. 2. In other words, we use $\mathcal{N}$ to represent the complicated effects caused by system environmental factors or the other faulty statements[3]. Here, we assume that, during the execution of each test case t, the program has the probability of being infected by $\mathcal{N}$. Once the program is infected, whether it fails or not will be entirely decided by $\mathcal{N}$. Specifically, we have $\mathcal{P}(o^t = fail | s \in C^t, \mathcal{N} \in C^t) = \mathcal{P}(o^t = fail | s \notin C^t, \mathcal{N} \in C^t)$. Modeling the fault interference as the *noise* under this assumption makes our analysis explicit, and it is reasonable because of the following reasons:

(1) The analysis of the *noise* can provide a fundamental and quantitative evaluation of the robustness of SBFL.
(2) If the program has multiple faults, it is likely to exist a dominant faulty statement that causes the main characteristics of spectra distribution. Then, other faults are likely to be correlated with only a few statements, and their effects can be eliminated [5].
(3) Although there are situations that multiple faults determine the program failures together, such situations can be approximated to our model through the assignment of the values of ε_2, ε_3, and ε_4 in Fig. 2.
(4) Many other faulty statements have lower a_{ef} values (e.g., faults in some remote modules) and do not have much influence on the target faulty statement. As a results, their behaviors are similar to the *noise*.

Consequently, the probabilistic model considering the *noise* module is shown in Fig. 2. Compared with the basic model shown in Fig. 1, concerning a test case t and statement s, additional notations are listed below.

[3] Although we may not have a "target" faulty statement in practice, the result of SBFL will probably assign one faulty statement with higher priority. Then, this faulty statement can be treated as "target" in fault localization.



$$\frac{\frac{a_{ef}(s)}{a_{ep}(s)}}{\frac{F}{P}} = \frac{\begin{matrix}\varphi_3\cdot\varphi_5 + \varphi_3\cdot\mathfrak{E}_2\cdot\mathfrak{E}_4\\ \varphi_3\cdot\varphi_4 + \varphi_3\cdot\mathfrak{E}_2\cdot\mathfrak{E}_3\end{matrix}}{\begin{matrix}\varphi_2+\varphi_5\cdot\varphi_3 + \mathfrak{E}_1\cdot\mathfrak{E}_4+\varphi_3\cdot\mathfrak{E}_2\cdot\mathfrak{E}_4\\ \varphi_1+\varphi_4\cdot\varphi_3 + \mathfrak{E}_1\cdot\mathfrak{E}_3+\varphi_3\cdot\mathfrak{E}_2\cdot\mathfrak{E}_3\end{matrix}}$$

Figure 3: The characteristics of spectra distribution in the probabilistic model with the *noise* module.

- ε_1^s: the probability that t *covers* $\mathcal{N}$ and *does not cover* s, i.e., $\mathcal{P}(\mathcal{N} \in C^t, s \notin C^t)$
- ε_2^s: the probability that t *covers* $\mathcal{N}$ and also *covers* s, i.e., $\mathcal{P}(\mathcal{N} \in C^t, s \in C^t)$
- ε_3^s: the probability that t *covers* $\mathcal{N}$ and *gets passed*, i.e., $\mathcal{P}(\mathcal{N} \in C^t, o^t = pass)$
- ε_4^s: the probability that t *covers* $\mathcal{N}$ and *reveals a failure*, i.e., $\mathcal{P}(\mathcal{N} \in C^t, o^t = fail)$

Obviously, due to the introduction of ε_1^s to ε_4^s, the probabilities from φ_1^s to φ_5^s will decrease. Specifically, similar to Eq. (5), the quantities used by SBFL can be calculated as follows:

$$\begin{aligned} P &\approx |T|(\varphi_1^s + \varphi_4^s\cdot\varphi_3^s + \varepsilon_1^s\cdot\varepsilon_3^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_3^s)\\ F &\approx |T|(\varphi_2^s + \varphi_5^s\cdot\varphi_3^s + \varepsilon_1^s\cdot\varepsilon_3^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_4^s)\\ a_{ef}(s) &\approx |T|(\varphi_3^s\cdot\varphi_5^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_4^s)\\ a_{ep}(s) &\approx |T|(\varphi_3^s\cdot\varphi_4^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_3^s) \end{aligned} \tag{15}$$

Then, we can calculate the following ratios related to the failure rate:

$$\frac{a_{ef}(s)}{a_{ep}(s)} \approx \frac{\varphi_3^s\cdot\varphi_5^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_4^s}{\varphi_3^s\cdot\varphi_4^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_3^s} \tag{16}$$

$$\frac{F}{P} \approx \frac{\varphi_2^s + \varphi_5^s\cdot\varphi_3^s + \varepsilon_1^s\cdot\varepsilon_4^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_4^s}{\varphi_1^s + \varphi_4^s\cdot\varphi_3^s + \varepsilon_1^s\cdot\varepsilon_3^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_3^s} \tag{17}$$

Finally, we can derive the equation shown in Fig.3.

The dashed boxes in Fig. 3 divide the equation into two parts. The left part and the right part express the spectra characteristics related to the *signal* and the *noise*, respectively. Specifically, the left part enclosed by the red dashed box can be regarded as the *signal*, denoted by $\mathcal{S}$. It approximates to the proportion between the original failure rate of s, denoted by $a_{ef}^{\mathcal{S}}(s)/a_{ep}^{\mathcal{S}}(s)$ and the original overall failure rate, denoted by $F^{\mathcal{S}}/P^{\mathcal{S}}$, as identified in Section 3.2.

On the other hand, the right part enclosed by the blue dashed box can be treated as the spectra of s only considering the *noise* module, such that each term contains at least one properties from ε_1^s to ε_4^s. Also, it just approximates to the proportion between the failure rate of s, denoted by $a_{ef}^{\mathcal{N}}(s)/a_{ep}^{\mathcal{N}}(s)$ and the overall failure rate, denoted by $F^{\mathcal{N}}/P^{\mathcal{N}}$, only considering the *noise*.

Now, suppose statement s is *Failure-Independent*, *Failure-Related*, and *Failure-Exclusionary*, respectively. Then, without the *noise*, the characteristic of its spectrum can be reflected by its failure rate $a_{ef}^{\mathcal{S}}(s)/a_{ep}^{\mathcal{S}}(s)$.[4] However, after considering the *noise*, such proportion becomes to $\frac{a_{ef}^{\mathcal{S}}(s)+a_{ef}^{\mathcal{N}}(s)}{a_{ep}^{\mathcal{S}}(s)+a_{ep}^{\mathcal{N}}(s)}$, which is closer to the failure rate of the *noise* module, i.e, $a_{ef}^{\mathcal{N}}(s)/a_{ep}^{\mathcal{N}}(s)$. On the other hand, the overall failure rate has also changed from the original signal part $F^{\mathcal{S}}/P^{\mathcal{S}}$ to $(F^{\mathcal{S}}+F^{\mathcal{N}})/(P^{\mathcal{S}}+P^{\mathcal{N}})$, getting closer to the overall failure rate of the *noise* module, i.e., $F^{\mathcal{N}}/P^{\mathcal{N}}$.

Moreover, from Fig. 3, we can also derive that

$$\frac{a_{ef}^{\mathcal{N}}(s)}{a_{ep}^{\mathcal{N}}(s)} \approx \frac{\varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_4^s}{\varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_3^s} = \frac{\varepsilon_1^s\cdot\varepsilon_4^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_4^s}{\varepsilon_1^s\cdot\varepsilon_3^s + \varphi_3^s\cdot\varepsilon_2^s\cdot\varepsilon_3^s} \approx \frac{F^{\mathcal{N}}}{P^{\mathcal{N}}} \approx \frac{\varepsilon_4^s}{\varepsilon_3^s} \tag{18}$$

In general, based on the model with the *noise* module, both the local failure rate of statement s and the overall failure rate, are getting closer to the failure rate of the *noise* module, i.e., $\varepsilon_4^s/\varepsilon_3^s$. Specifically, we have the following conclusions:

Characteristic 3. *After the introduction of the* noise *module, the failure rate of statements, i.e., $a_{ef}(s)/a_{ep}(s)$, changes towards the failure rate of the* noise *module, i.e., $\varepsilon_4^s/\varepsilon_3^s$.*

Characteristic 4. *After the introduction of the* noise*, the overall failure rate, i.e., F/P, changes towards the overall failure rate of the* noise *module, i.e., $\varepsilon_4^s/\varepsilon_3^s$.*

Characteristic 5. *The relationship between a statement s and program failures (i.e., Failure-Independent, Failure-Related, and Failure-Exclusionary) may change after adding the* noise*. The direction of this change depends on both the failure rate of the* noise *module $\mathcal{N}$, i.e., $\varepsilon_4^s/\varepsilon_3^s$, and the relationship between s and $\mathcal{N}$. Specifically,*

1. *Assume $\varepsilon_4^s/\varepsilon_3^s \approx F/P$, i.e., the failure rate of the* noise *module is approximate to the failure rate of the signal parts. Then, F/P is approximate to the original $F^{\mathcal{S}}/P^{\mathcal{S}}$. Also, the original correlation between s and program failures can become weak. Specifically, the failure rates are getting lower, higher, and keeping the same for the Failure-Related, Failure-Exclusionary, and Failure-Independent statements, respectively.*
2. *If $\varepsilon_4^s/\varepsilon_3^s \neq F^S/P^S$, the situation is complicated. Specifically, the initial failure rate that larger or smaller than $\varepsilon_4^s/\varepsilon_3^s$ will become smaller or larger, respectively. However, the change of the correlation between s and program failures, i.e., whether it becomes more Failure-Related or Failure-Exclusionary also depends on the change of F/P.*
3. *Assume s has a low execution rate in the signal part, while it has a high execution rate in the* noise *module. Then, the influence of the* noise *on the original correlation between s and program failures is obvious. On the contrary, the influence on the faulty statement s with the spectrum $a_{ef}^{\mathcal{S}}(s^f) = F$ is likely to be slighter than other statements with similar failure rates.*

4 VISUALIZATION OF THE SPECTRA DISTRIBUTION CHARACTERISTICS

In this section, we introduce a visualization approach to generalize the results in Section 3 and provide an overall observation about

[4]Here, we use $a_{ef}^{\mathcal{S}}(s)/a_{ep}^{\mathcal{S}}(s)$ instead of the actual failure rate $a_{ef}^{\mathcal{S}}(s)/(a_{ep}^{\mathcal{S}}(s) + a_{ef}^{\mathcal{S}}(s))$ for the convenience of calculation. This operation will be applied in the following parts of the paper.

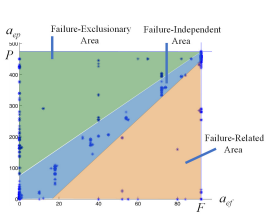


Figure 4: The visualization of the characteristics of spectra distribution

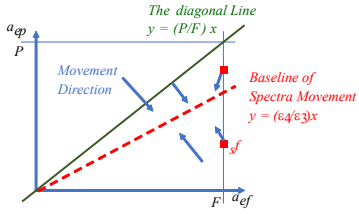


Figure 5: Visualization of the dynamics of program spectra when concerning the *noise*.

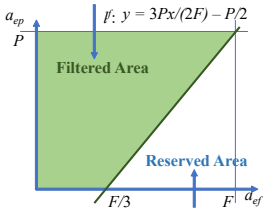


Figure 6: The filtered area in SS

spectra distribution and its dynamics after involving the *noise*. The approach of visualization is based on the concept of *Spectra Space (SS)* [27], which is a coordination system in which the x-axis indicates the meaning of a_{ef}, and the y-axis indicates the meaning of a_{ep}, as shown in Fig. 4. Then, the spectrum of each statement s is presented as an image point in SS, denoted by $\iota(s)$, with the coordination values of $(a_{ef}(s), a_{ep}(s))$. Note that, because at each s, we have $a_{ef}(s) < F$ and $a_{ep}(s) < P$, SS is actually a box enclosed by the lines $x = F$, $y = P$, $x = 0$, and $y = 0$. Now, given an SBFL instance, we can obtain the value of F and P, and then visualize all the statements in SS according to their spectra. Then, we can observe the characteristics of the spectra distribution of each SBFL instance (illustrated as the distribution of image points in Fig. 4). Through summarizing different SBFL instances, we can get an intuitive image about the distribution patterns of program spectra, which corresponds to the results found in Section 3.

Fig. 4 reflects the *CSD*s of each type of statements. Specifically, the image points of *Failure-Independent* statements will be distributed around the diagonal line of SS, depicted as the blue area. Meanwhile, the image points of the *Failure-Related* and *Failure-Exclusionary* statements are distributed below and above the diagonal line of SS, respectively (shown as the red and green areas).

Furthermore, Fig. 5 illustrates the visualization of the change of the *CSD*s after suffering the *noise* caused by multi-fault interference, as discussed in Section 3.3. Here, the analytic expression of the diagonal line, i.e., $y = (P/F)x$, denoted by l^d, is treated as an indicator. Specifically, suppose the fault inference, caused by other faulty elements and modeled as the *noise*, exists. Then, the position of all the artifacts in SS, including the image points of program statements, the diagonal line, and the target faulty statement s^f, will change. According to Characteristics 3 to 5, the baseline of such movement is the line with analytic expression of $y = (\varepsilon_3/\varepsilon_4)x$ (i.e., the failure rate of the *noise* module $\mathcal{N}$), denoted by l^b. All the points of program statements including the faulty statement, and the diagonal line l^d will get closer to the baseline l^b. Specifically, we can have the following graphical *CSD*:

Characteristic 6.

(1) *For statement s, if $\iota(s)$ is below both the lines of l^b and l^d, then $\iota(s)$ may take a left-top move, and its correlation with program failures may become weak.*
(2) *For statement s, if $\iota(s)$ is above both l^b and l^d, then $\iota(s)$ may take a right-bottom move, and its correlation with program failures tends to be enhanced.*
(3) *For statement s, if $\iota(s)$ is between the lines of l^b and l^d, then $\iota(s)$ tends to move towards the line of l^b. Nonetheless, because l^d also tends to move toward l^b, the variation of its correlation with the program failures should further depend on the relative degrees of these movements.*

Compared with the Characteristics in Section 3.3, Characteristic 6 is more intuitive and it presents a general picture of the program spectra's movement under the *noise*. Specifically, the results can be understood from two aspects:

(1) Under the *noise* introduced from fault interference, all the image points tend to move towards the baseline l^b, including the faulty statement s^f. Thus, the property of s^f may get more implicit (i.e., $\iota(s^f)$ may not be at the boundary line $x = F$). In this sense, fault interference indeed takes negative effects.
(2) On the other hand, the movements of the statements' image points follow explicit rules. If the noise is not strong (i.e., ε_1^s and ε_2^s are not significant), the degree of movement will be slight. Furthermore, the faulty statement s^f has the largest original $a_{ef}(s^f)$ value. Thus, the degree of movement of its spectrum could be less than other statements with similar failure rates. In this sense, SBFL is also reasonable under a slight degree of fault interference.

5 EXAMPLE FOR USAGE: A FILTER FOR ELIMINATING NON-FAULTY STATEMENTS

From the above analysis, we find a series of *CSD*s. In this section, we explore the usage of these *CSD*s to improve SBFL. Specifically, as a first step, we design a filter that eliminates the statements whose spectra are not fault-prone based on the derived *CSD*s. The following strategies are adopted:

(1) The *Failure-Independent* and *Failure-Exclusionary* statements should be filtered. Intuitively, because the faulty statement is the root cause of program failures, it should be *Failure-Related*. However, *Failure-Independent* and *Failure-Exclusionary* statements may also have large values of a_{ef} or small values of a_{ep}, which may mislead the application of SBFL metrics. Thus, we can eliminate them during the execution of SBFL.
(2) The negative effects cause by fault interference can be, to some extent, filtered. According to Characteristic 5, because of the *noise*, the image point of the faulty statement s^f, i.e., $\iota(s^f)$, is likely to take a left-top move. This may cause the negative effect that $a_{ef}(s^f) < F$. However, some SBFL metrics (e.g., *Op*) assign the term a_{ef} with a high weight. As

a result, the statements whose image points are near the right-top area of SS may have higher suspiciousness values than s^f. Thus, it is reasonable to filter such statements which are likely to be *Failure-Independent*. On the other hand, as analyzed in Section 4, because the movement of $\iota(s^f)$ is slighter than other $\iota(s)$s, $a_{ef}(s^f)$ should not be very small, even though the *noise* exists. Thus, it is reasonable to filter statement s, if $\iota(s)$ is at the left part of SS.

Based on the above analysis, Fig. 6 illustrates the design of the statement filter. Specifically, in SS, we introduce a filter line, denoted by l^f, with the analytical expression of $y = \frac{3P}{2F}x - P/2$. Statements whose image points are *above* this line will be filtered. Concerning l^f, it is near the diagonal line of SS. Thus, the eliminated statements are more likely to be *Failure-Independent* or *Failure-Exclusionary*. Also, l^f is below the diagonal line. Thus, it also filtered the statements with image points near the original point $(0, 0)$ and the right-top point (F, P). Generally, the SBFL technique integrated with statement filter (*SBFL-Filter* for short) is presented in Algorithm 1.

Algorithm 1 SBFL with statement filter

Input: PG : the subject program with set of statements S; T: the executed test suite;
Output: L: the rank list of SBFL;
1: calculate the overall quantities F and P;
2: **for** each s in S **do**
3: calculate $a_{ef}(s)$ and $a_{ep}(s)$;
4: calculate the suspiciousness value $R(s)$ use conventional SBFL;
5: **end for**
6: present all the information in the Spectra Space;
7: draw the filter line l^f;
8: Initialize $S^{filter} \leftarrow \emptyset$, $S^{reserve} \leftarrow \emptyset$;
9: **for** each s in S **do**
10: **if** s is above l^f **then**
11: Let $S^{filter} \leftarrow S^{filter} \cup \{s\}$
12: **else**
13: Let $S^{reserve} \leftarrow S^{reserve} \cup \{s\}$
14: **end if**
15: **end for**
16: **for** each s in S^{filter} **do** $R(s) \leftarrow R(s) - M$, M is large enough satisfying $R(s) > M$, $\forall s \in S$;
17: **end for**
18: Re-rank the statements according to their new suspiciousness values and get L;

Note that, when filtering the negative effects caused by the *noise*, some *Failure-Related* statements will also be included. Thus we may have threats to eliminate the real faulty statement. Therefore, how to trade off the potential benefits and the threats should be considered. In our approach, the filter area (i.e., the green shadow part in Fig. 6) is quite small at the top-right part of SS. This indicates that the filtering strategy on this part is quite moderate. Also, the shadow area contains only a small region of the left-bottom part of SS (i.e., near the original point). Specifically, l^f intersects with the x-axis at the point of $(F/3, 0)$, which means only the $a_{ef}(s^f)$ value smaller than $F/3$ will be eliminated. Unless the program has more than three completely exclusive faulty statements with similar failure rates, $\iota(s^f)$ cannot move to the right part of $x = F/3$. Furthermore, we try to improve the stability of our filter by reserving the top ten statements in the rank list of SBFL. That is, for SBFL instances whose outputs are already accurate, our filter does not take effects.

Table 1: Details of the benchmark programs and faults

Suite	Programs	Fault No.	LOC	Size of test suites
Unix	*flex*	53	10,459	567
	grep	17	10,068	809
	gzip	17	5,680	217
	sed	26	14,427	370
Siemens	*print_tokens*	5	472	1,608
	print_tokens2	9	399	2,650
	replace	29	512	2,710
	tcas	41	141	1,052
	schedule	5	292	4,130
	schedule2	9	301	4,115
	tot_info	23	440	5,542
Space	*space*	33	6,199	13,585
Defects4J	*Chart*	26	96k	2205
	Closure	133	90k	7927
	Math	106	85k	3602
	Lang	65	22k	2245
	Time	27	28k	4130

6 EMPIRICAL STUDY

6.1 Research Questions

In this section, we conduct an empirical study to show the effectiveness of the statement filter, and also examine the *CSD*s in real SBFL instances. The following research questions are explored:

RQ1: Can the statement filter improve the accuracy of SBFL?

RQ2: Which factors (e.g., different subject programs and SBFL metrics) can affect the performance of *SBFL-Filter*?

RQ3: Which *CSD*s cause the performance of *SBFL-Filter* decline/increase? Are they in accord with our analysis?

6.2 Empirical Setup

In the empirical study, four *UNIX utility* programs (including *flex*, *grep*, *gzip*, and *sed*), the *Space* program, and the *Siemens* suite (including seven subprograms: *print_tokens*, *print_tokens2*, *replace*, *tcas*, *schedule*, *schedule2*, and *tot_info*) are selected as benchmarks. These programs are derived from the Software Infrastructure Repository (SIR) [6]. In SIR, each program is attached to several benchmark faults. This paper discusses the *CSD*s for both single-fault and fault-interference situations. Thus, we generate and examine both the single-fault and multi-fault versions. For single-fault instances, all the benchmark faults are examined one by one. For multi-fault instances, we combine different faulty statements such that they can exist together in the same instance. Here, we check 2-fault and 3-fault situations, respectively [5, 24].

In addition, we also use *Defects4J*[9] , a dataset including five java projects *Chart*, *Closure*, *Lang*, *Math* and *Time*. The projects are with the scales from 28 KLOC to 96 KLOC and with real-life faults. Here, we do not combine different faulty statements to form multi-fault instances, such that the property of real-life faults remains. Specifically, we use the data derived in [18]. The details of the benchmarks are shown in Table 1.

In the empirical study, we run conventional SBFL and *SBFL-Filter*, respectively, and compare their performances. There are many criteria to evaluate the performance of SBFL according to different requirements of practical applications. Because our work mainly focuses on theoretical analysis, we intend to evaluate the

accuracy of SBFL in a typical way. Therefore, we adopt a tradition criterion *Expense*, defined as the percentage of statements that rank above the faulty statement s^f. Note that we use average ranking as the tie-break strategy. During the execution, we also record the $a_{ef}(s)$ and $a_{ep}(s)$ values for each statement s such that the spectra distribution of each SBFL instance can be observed. When dealing with multi-fault versions, we set the faulty statement that ranks above the other faulty statements as the target faulty statement, i.e., s^f, and treat the other faulty statements as the *noise*.

6.3 Result and analysis

To answer **RQ1**, we conduct an overall comparison and present the results in Table 2. The first column records the benchmark programs, while the first row records the adopted SBFL metrics. At each metric, the columns "better" and "worse" indicate the number of faulty versions in which *SBFL-Filter* takes positive and negative effects (i.e., increases and decreases the SBFL accuracy), respectively.

Table 2: Comparison of conventional SBFL and *SBFL-Filter*.

	Op		*Ochiai*		*Tarantula*	
	Better	Worse	Better	Worse	Better	Worse
flex	32	0	39	14	741	77
grep	8	0	26	13	269	23
gzip	9	3	3	4	154	15
sed	38	0	2	0	114	0
Siemens	52	0	89	30	637	98
Space	15	0	33	18	322	78
Chart	1	0	1	0	3	0
Closure	7	1	1	0	11	2
Lang	0	0	1	0	4	0
Math	1	0	0	0	6	0
Times	0	0	0	0	1	0
Total	163	4	195	79	2262	293

From Table 2, the filter designed based on the identified *CSD*s can indeed improve SBFL. Concerning all the programs and metrics, *SBFL-Filter* performs better than conventional SBFL on 2620 SBFL instances, including both single-fault and multi-fault instances, over eight times than the number of instances that *SBFL-Filter* performs worse. Note that we do not consider the instances with *Expense* values larger than 0.3 (i.e., the outputs are entirely inaccurate). Because if SBFL itself is not suitable for these instances, it does not make much sense whether we adopt the filter or not (In fact, *SBFL-Filter* can get improvement at most of these instances).

Furthermore, we try to answer **RQ2**. Specifically, we further examine the performance of *SBFL-Filter* on different subject programs and SBFL metrics. Considering different subject programs, *SBFL-Filter* has different performances. It has the most stable performance on *sed*, in which we have 154 better SBFL instances and 0 worse instances. The filter also performs well at *Siemens* programs with 778 better SBFL instances. However, we also have 128 worse SBFL instances. We infer that the impacts of different subject programs are mainly due to the variety of program structures and fault types, which leads to the diversity of spectra distribution. Some programs are organized with the structure in which the execution of a component may not depend on the status of other components (e.g., the faulty statement). As a result, the SBFL instances based on these programs may contain a large number of *Failure-Independent* statements that may mislead some SBFL metrics (e.g., $R_{\mathcal{O}}$). In these situations, our filter can effectively eliminate these statements. In addition, for *Defects4J* programs, we only examine the real-life faulty versions (i.e., considering the fixing patches one by one). Thus, the produced SBFL instances are likely to possess the property of single-fault scenarios (i.e., $a_ef(s^f) = F$), even if the number of faulty statements is more than one, as shown in Fig. 8. Thus, the improvement of *SBFL-Filter* is not significant. However, it is also visible and stable, including 37 improved instances and only 3 degraded instances.

Moreover, the properties of faulty statements may also influence the performance of *SBFL-Filter*. Suppose the program has multiple faulty statements, that cause the movement of s^f. Then, our filter can eliminate some statements located in the right part of $\iota(s^f)$ produced due to this movement. However, suppose both the degree of fault interference and the degree of coincidental correctness are high[5]. Then, $\iota(s^f)$ itself could move to the area that is eliminated by the filter. In such situations, the filter may take adverse effects. Nevertheless, as discussed in Section 5, the design of the filter is quite conservative and it does not take negative effects in most instances. On the other hand, it is still valuable to further explore more comprehensive knowledge of spectra distribution and then refine the *SBFL-Filter*.

Considering SBFL metrics, Fig. 7 presents the representative instances in which *SBFL-Filter* performs well on the metrics of *Op*, *Ochiai*, and *Tarantula*, respectively. Firstly, the *SBFL-Filter* has the most stable performance on metric *Op*, with 163 better instances and only 4 worse instances. One of the main reasons is the extreme strategy of *Op*, which puts $a_{ef}(s)$ in the dominant position. Because of this, it has maximal performance for single-fault situations but has lower robustness to fault interference [24]. Thus, the application of *SBFL-filter* could be useful (as shown in Fig. 7(a)).

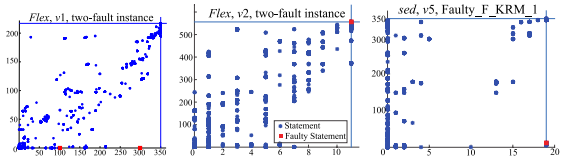
Also, *SBFL-filter* performs well at *Tarantula* on 2262 better instances and 293 worse instances. *Tarantula* mainly considers the failure rate of each statement, i.e., $a_{ef}(s)/a_{ef}(s)$. Thus, it does not good at identifying the statements that are *Failure-Related* but not frequently covered by test suite T. The image points of such statements are likely to be close to the original point of SS, i.e., $(0, 0)$, and will be eliminated by the filter (as shown in Fig. 7(c)).

Finally, for metric *Ochiai*, its expression is equivalent to the formula of $a_{ef}(s)^2/(a_{ef}(s) + a_{ep}(s))$, which considers both the statement's failure rate $a_{ef}(s)/(a_{ef}(s)+a_{ep}(s))$ and the failure frequency $a_{ef}(s)$. Compared with the other two metrics, it is more robust. However, the image points located between the diagonal line and the right-bottom area are also misleading, and these statements can be partially eliminated by our filter (as shown in Fig. 7(b)).

Table 3: Comparison of different SBFL metrics after applying statement filter.

Metric	*Op*	*Ochiai*	*Tarantula*	**All**
Better No.	39	50	2004	11
Worse No.	3	4	217	0

[5]Here, coincidental correctness refers to the phenomenon that test cases cover s^f but do not reveal failures. It is a critical issue to be considered in SBFL [14].



(a) Op (b) Ochiai (c) Tarantula

Figure 7: The spectra distribution of the instances with our strategy has good performance at each SBFL metric.

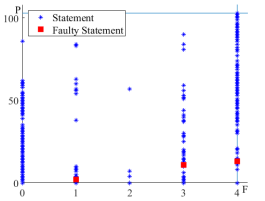


Figure 8: A representative instance of spectra distribution in Defect4J (*Closure*, faultID 76)

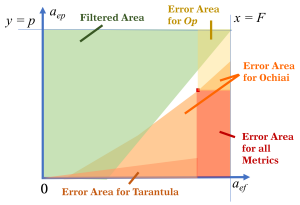
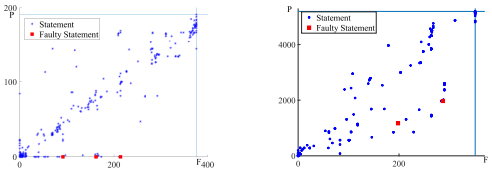


Figure 9: The areas in SS that could be misidentified by $R_{\mathscr{O}}$, R_O, R_τ



(a) *flex*, v1, three-fault instance (b) *Siemens, replace*

Figure 10: The spectra distribution of the instances at with our strategy has good performance on all the SBFL metrics.

To summarize the performance of *SBFL-Filter* on different SBFL metrics and also to answer **RQ3**, we present the number of faulty versions in which the filter can derive better/worse results at only one SBFL metric (See the columns of *Op*, *Ochiai*, and *Tarantula* of Table 3). In addition, Fig. 9 illustrates the areas in which each metric can produce errors (i.e., ranking the statements belonging to these areas ahead of the target faulty statement s^f)[6]. We can observe that, all the metrics have their own "blind" area and each of them can be partially eliminated by our filter. Also, there are situations that *SBFL-Filter* can perform well on all SBFL metrics (shown in the **All** column of Table 3). Fig. 10, illustrates the spectra distribution of the SBFL instances in which *SBFL-Filter* can improve the accuracy of fault localization at all the metrics. We can observe that the spectra distribution of these SBFL instances possess the characteristics that mislead each SBFL metrics, respectively.

Generally speaking, the *SBFL-Filter* is an effective technique to refine the localization results. It indeed takes into account the derived *Characteristics of Spectra Distribution* beyond the basic intuitions of SBFL. Furthermore, the *SBFL-Filter* processes the spectra information can be combined with other techniques (such as the test case prioritization techniques [25]).

[6]The figure is based on our analyses in answering **RQ2** and also based on the analysis of the performance of each SBFL metric in the Spectra Space

7 RELATED WORKS

Spectrum-Based Fault Localization (SBFL) is one of the most typical fault localization techniques [22]. The core of SBFL is the metrics that calculate the fault proneness for each program component. Jone et al. [8] developed the typical metric, named *Tarantula*. Since then, various metrics have been proposed [1, 2]. Naish et al. [16] innovatively made a general theoretical comparison among various metrics and grouped them into several equivalent categories. Xie et al. [23] further proved the performance among each equivalent categories and proposed the concept of maximal metrics. Furthermore, Yoo et al. [26] proposed a generative approach to find new metrics. To the best of our knowledge, studies on exploring the laws of the overall program spectra distribution have not been well conducted.

In addition, studies on optimizing the SBFL process have also been conducted from various aspects, such as the quality of test information [4, 7, 25, 28] and coincidental correctness [14]. Parnin et al. [17] discussed the challenge of how to make SBFL results support the debugging process. Pearson et al. [18] made an overall experimental study of different fault localization techniques using various benchmarks. Perez et al. [19] analyzed the diagnosability of fault localization.

Recently, Deep Learning (DL) is a hot topic and tends to be concerned in software engineering [3, 12]. Li et al. [11] adapted deep learning approaches to fault localization such that more features from different dimensions can be adopted. Nevertheless, we think that it is still useful to explore the knowledge of spectra distribution as well as other theories related to SBFL. Firstly, we can derive a more comprehensive understanding of fault localization. Moreover, The mining of spectra characteristics involving heterogeneous artifacts can help us identify more essential features. This will improve the explainability of DL-based approaches and facilitate the construction of more reasonable deep networks for fault localization.

8 CONCLUSION AND FUTURE WORK

To pursue the basic intuitions of SBFL, this paper conducts a further exploration of the *Characteristics of Spectra Distribution* (*CSD*s). We mainly focus on how the relationship between the coverage of program statements and the manifestation of failures can be reflected in program spectra. In particular, our analysis is not limited to single-fault scenarios. The effect of fault interference, model as the *noise* module, is also discussed. As a result, we extend the base of SBFL-related conclusions and present them as a series of *CSD*s. Specifically, the contributions of this paper are listed below:

(1) We proposed the concepts of *Failure-Independent*, *Failure-Related*, and *Failure-Exclusionary* to describe the relationships between program failures and the coverage of each statement within the scope of program spectra.
(2) We build a probabilistic model to derive the formal description of the characteristics of program spectra under the

concepts of *Failure-Independent*, *Failure-Related*, and *Failure-Exclusionary*. This is our first trial to explore quantitative conclusions beyond the basic intuitions of SBFL.

(3) We extend the probabilistic model by introducing the *noise* module based on the assumption of noise-style fault interference. Then, we quantitatively analyze the spectrum movement of each type of statement and the change of Spectra Space. This is our first trial to make a quantitative analysis for the complicated internal interactions in multi-fault instances.

(4) We summarize the derived *CSD*s using a visualization approach, which presents our analysis results in a general way.

(5) Several interesting results are found. For example, 1) different types of statements are distributed in different areas of the Spectra Space. 2) The *noise* will weaken the original characteristics of each type of statements. 3) The statements tend to move towards the diagonal lines of the Spectra Space after suffering the *noise*.

(6) To demonstrate the usefulness of our analysis, we design a statement filter to eliminate the *Failure-Independent* and *Failure-Exclusionary* statements. Our strategy is based on visualization and utilizes both the single-fault and noise-involved *CSD*s.

In the future, we will extend our approach by establishing more complicated probabilistic models and exploring more spectra distribution characteristics. Then we will use the derived spectra distribution rules to improve SBFL. Furthermore, the approach that intuitively summarizes the identified *CSD*s will be evaluated by practical use cases.

ACKNOWLEDGMENTS

Xiao-Yi Zhang is supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST. Funding Reference number: 10.13039/501100009024 ERATO.

REFERENCES

[1] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2006. An evaluation of similarity coefficients for software fault localization. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. IEEE, 39–46.

[2] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2007. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 89–98.

[3] Tien-Duy B Le, David Lo, Claire Le Goues, and Lars Grunske. 2016. A learning-to-rank based fault localization approach using likely invariants. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 177–188.

[4] Benoit Baudry, Franck Fleurey, and Yves Le Traon. 2006. Improving test suites for efficient fault localization. In *Proceedings of the 28th international conference on Software engineering*. ACM, 82–91.

[5] Nicholas DiGiuseppe and James A Jones. 2015. Fault density, fault types, and spectra-based fault localization. *Empirical Software Engineering* 20, 4 (2015), 928–967.

[6] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. 2005. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering* 10, 4 (01 Oct 2005), 405–435. https://doi.org/10.1007/s10664-005-3861-2

[7] Bo Jiang, Ke Zhai, Wing Kwong Chan, TH Tse, and Zhenyu Zhang. 2013. On the adoption of MC/DC and control-flow adequacy for a tight integration of program testing and statistical fault localization. *Information and Software Technology* 55, 5 (2013), 897–917.

[8] James A Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. IEEE, 467–477.

[9] René Just, Darioush Jalali, and Michael D Ernst. 2014. 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*. ACM, 437–440.

[10] G Latif-Shabgahi, Julian M Bass, and Stuart Bennett. 2004. A taxonomy for software voting algorithms used in safety-critical systems. *IEEE Transactions on Reliability* 53, 3 (2004), 319–328.

[11] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 169–180.

[12] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 120–131.

[13] Wes Masri, Rawad Abou-Assi, Marwa El-Ghali, and Nour Al-Fatairi. 2009. 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)*. ACM, 1–5.

[14] Wes Masri and Rawad Abou Assi. 2014. Prevalence of coincidental correctness and mitigation of its impact on fault localization. *ACM transactions on software engineering and methodology (TOSEM)* 23, 1 (2014), 8.

[15] Wolfgang Mayer and Markus Stumptner. 2008. Evaluating models for model-based debugging. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 128–137.

[16] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2011. A model for spectra-based software diagnosis. *ACM Transactions on software engineering and methodology (TOSEM)* 20, 3 (2011), 11.

[17] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers?. In *Proceedings of the 2011 international symposium on software testing and analysis*. ACM, 199–209.

[18] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. 2017. Evaluating and improving fault localization. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 609–620.

[19] Alexandre Perez, Rui Abreu, and Arie Van Deursen. 2019. A Theoretical and Empirical Analysis of Program Spectra Diagnosability. *IEEE Transactions on Software Engineering* (2019).

[20] Mehrdad Saadatmand, Detlef Scholle, Cheuk Wing Leung, Sebastian Ullström, and Joanna Fredriksson Larsson. 2014. Runtime verification of state machines and defect localization applying model-based testing. In *Proceedings of the WICSA 2014 Companion Volume*. ACM, 6.

[21] W Eric Wong, Vidroha Debroy, Ruizhi Gao, and Yihao Li. 2013. The DStar method for effective software fault localization. *IEEE Transactions on Reliability* 63, 1 (2013), 290–308.

[22] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[23] Xiaoyuan Xie, Tsong Yueh Chen, Fei-Ching Kuo, and Baowen Xu. 2013. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22, 4 (2013), 31.

[24] Yanhong Xu, Beibei Yin, Zheng Zheng, Xiaoyi Zhang, Chenglong Li, and Shunkun Yang. 2019. Robustness of spectrum-based fault localisation in environments with labelling perturbations. *Journal of Systems and Software* 147 (2019), 172–214.

[25] Shin Yoo, Mark Harman, and David Clark. 2013. Fault localization prioritization: Comparing information-theoretic and coverage-based approaches. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22, 3 (2013), 19.

[26] Shin Yoo, Xiaoyuan Xie, Fei-Ching Kuo, Tsong Yueh Chen, and Mark Harman. 2017. Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 1 (2017), 4.

[27] Xiao-Yi Zhang and Zheng Zheng. 2019. A Visualization Analytical Framework for Software Fault Localization Metrics. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 148–14809.

[28] Xiao-Yi Zhang, Zheng Zheng, and Kai-Yuan Cai. 2018. Exploring the usefulness of unlabelled test cases in software fault localization. *Journal of Systems and Software* 136 (2018), 278–290.

[29] Zhenyu Zhang, Wing Kwong Chan, TH Tse, Bo Jiang, and Xinming Wang. 2009. Capturing propagation of infected program states. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 43–52.

[30] Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D Ernst, and Lu Zhang. 2019. An empirical study of fault localization families and their combinations. *IEEE Transactions on Software Engineering* (2019).