symmetry

MDPI

Article

An Innovative Framework for Coincidental Correctness Impacting on Fault Localization

Heling Cao [1,2], Lei Li [1,2,*] and Yigui Sun [1]

[1] College of Information Science and Engineering, Henan University of Technology, Zhengzhou 450001, China; caohl@haut.edu.cn (H.C.); effectiveagui@haut.edu.cn (Y.S.)
[2] Henan International Joint Laboratory of Grain Information Processing, Henan University of Technology, Zhengzhou 450001, China
* Correspondence: leili@haut.edu.cn

Abstract: An important research aspect of Spectrum-Based Fault Localization (SBFL) is the influence factors of the effectiveness of suspiciousness formulas from the perspective of symmetry. Coincidental correctness is one of the most important factors impacting the effectiveness of suspiciousness formulas. The influence of fault localization by coincidental correctness has attracted a large amount of research in the perspective of empirical study; however, it can hardly be considered as sufficiently comprehensive when there are a large number of the symmetrical suspiciousness formulas. Therefore, we first develop an innovative theoretical framework with function derivation investigating suspiciousness formulas impacted by coincidental correctness. We define three types of relations between formulas affected by coincidental correctness: namely, improved type, invariant type and uncertain type. We investigated 30 suspiciousness formulas using this framework and group them into three categories. Furthermore, we conduct an empirical study to verify the effectiveness of SBFL affected by coincidental correctness on four relatively large C programs. We proved that coincidental correctness has a positive effect on 23 out of these 30 formulas, no effect on 5 of them, and the effect on the remaining 2 of them depend on certain conditions. The experimental results show that the effectiveness of some suspiciousness formulas can be enhanced and that of some suspiciousness formulas remain unchanged.

Keywords: fault localization; coincidental correctness; theoretical analysis; suspiciousness formulas

Citation: Cao, H.; Li, L.; Sun, Y. An Innovative Framework for Coincidental Correctness Impacting on Fault Localization. *Symmetry* **2022**, *14*, 1267. https://doi.org/10.3390/sym14061267

Academic Editors: Aviv Gibali and Adam Glowacz

Received: 29 April 2022
Accepted: 11 June 2022
Published: 19 June 2022

1. Introduction

Software testing and debugging are important and time-consuming activities in software engineering. Attempting to reduce the faults in software is estimated to consume 50–70% of the time devoted to software debugging and maintenance [1]. Due to the cost of expensive debugging, locating faults is a very resource-consuming activity in the overall software development lifecycle. Therefore, much research has been devoted to fault localization in order to increase the quality of software and reduce its cost. Spectrum-based fault localization is one promising approach that aims at identifying the executed program statements that contain likely faults. Fault localization has a wide range of application scenarios in real life. Due to software faults in the body control module, about 433,000 cars will be recalled on safety specifications in North America in 2015, as shown in Figure 1. The automatic train protection system stopped for the Wuhan–Guangzhou high speed railway 55,901 because the unconditional emergency shutdown included software bugs in the radio block center system that received the stop instructions during the operation in 2012, as shown in Figure 2. It can be seen that software failure may cause huge economic losses and even endanger people's lives. Therefore, it is very necessary to locate faults to improve and ensure software quality. The common assumption is that the program will produce the wrong output and fail when the faulty statements are executed. However, in some cases,

the faulty statements were executed and did not induce failure. The PIE model proposed by Voas [2] emphasizes that the occurrence of a failure must satisfy three conditions: (a) the faulty statement was executed; (b) the program has transitioned to an infectious state; and (c) the infection state propagated to program output. When the program produces the correct output, when condition (a) is met, but not condition (c), coincidental correctness (CC) arises. In particular, coincidental correctness might degrade the effectiveness of fault localization.

Figure 1. Ford Car.

Figure 2. High-Speed Railway.

Masri et al. [3] demonstrated that the occurrence of coincidental correctness was prevalent, and that coincidental correctness was a safety-reducing factor for Tarantula and could enhance its effectiveness. The previous studies [3–6] have tackled the prevalence of coincidental correctness and its degrading effect on the effectiveness of fault localization. They indicated that cleansing coincidental correctness from the test suite might enhance the effectiveness of fault localization. Assi et al. [7] empirically study the impact of weak and strong CC on coverage-based fault localization techniques and indicated that small and large proportions of CC tests are strongly harmful to fault localization. In addition, Masri et al. [8] showed that the majority of program dynamic dependencies do not transfer any measurable information; that is, in most cases, two statements or variables in a dynamic dependency chain do not necessarily mean the connection between them. Therefore, the most infectious states in the program might not propagate to program output, which leads to a high rate of coincidental correctness.

In the study [9], the impact of coincidental correctness on Jaccard, Tarantula, AMPLE, and Ochiai formulas was investigated through the proof of inequality. Feyzi [10] proposed a framework, CGT-FL, diminishing the negative impact of coincidental correctness tests on the effectiveness of fault localization by a value-based cooperative game-theoretic method. Hofer [11] present a technique that identifies potential outcomes of coincidental correctness, and their empirical study shows that the removal of the actual coincidental correctness often positively influences the ranking of the faulty statements.

After Lee et al. [12] proved that Tarantula is equal to QE formulas, a more comprehensive investigation was conducted, where more groups of suspiciousness formulas have been proved equivalently [13]. Xie et al. [14] proposed a theoretical analysis of the risk evaluation formulas for fault localization. Such an observation can give an explanation on why the relations between two formulas is equivalent or better. However, they proved the relationship between suspiciousness formulas without considering the existence of coincidental correctness. It has not yet been studied how the other SBFL formulas are affected by coincidental correctness? Thus, it is worthwhile to theoretically investigate how the other SBFL formulas are affected by coincidental correctness.

Inspired by this observation, we first develop an innovative theoretical framework to prove the effectiveness of SBFL formulas in order to identify whether the effectiveness of a formula is affected by the factor of coincidental correctness. In this paper, we proposed a theoretical framework by the derivative of function to investigate how 30 SBFL formulas are affected by coincidental correctness. We prove 30 SBFL formulas, which were selected from Naish et al. [13], because their theoretical investigation is the most systematic and comprehensive one for SBFL formulas. We performed a set of empirical studies across four C programs to evaluate the effectiveness of SBFL formulas. The experimental results show that the effectiveness of some SBFL formulas was improved, and the effectiveness of some SBFL formulas was unchanged The contributions of this paper are summarized as follows.

- We developed a function theoretical framework, which compared the derivative value of a formula between the original test suite and the test suite removing coincidental correctness to determine the suspiciousness value of a formula.
- Using this framework, we investigated 30 SBFL formulas when removing coincidental correctness. We are able to find that 23 of these 30 formulas are improved formulas, five of them are unaffected formulas, and the remaining two of them are uncertain.
- We conducted the experiments with four C open-source programs to evaluate coincidental correctness on the effectiveness of SBFL. The results show that the effectiveness of some suspiciousness formulas was enhanced indeed and some other suspiciousness formulas were unchanged.

The remainder of this paper is organized as follows. Section 2 provides the background of spectrum-based fault localization and coincidental correctness. Section 3 presents our framework and empirical study. Finally, Section 4 provides the conclusions.

2. Background

2.1. Spectrum-Based Fault Localization (SBFL)

The spectrum of the program is an execution profile which indicates the parts of the program that are active during the execution of the program. Spectrum-based fault localization needs to identify the parts of the program whose activity is most related with the detection of faults. SBFL intends to locate program faults by utilizing various program spectra and the test result of passed or failed obtained from dynamically testing. After obtaining the necessary information, SBFL utilizes various formulas to compute the suspiciousness of program statements and produces a ranking list to locate the fault. Golagha et al. [15] proposed the model to assess the potential effectiveness of fault localization. Simalarly, Dutta et al. [16] proposed a modified Fisher's test-based statistical method that maked use of test execution results as well as statement coverage information to determine the suspiciousness of executable statements. Ghosh et al. [17] used logistic mapping function to achieve chaotic sequence, which first calculated the suspiciousness score for each program state-

ment and then assigned ranks according to that score. Considering that the interactive behaviors among software entities implied some fault patterns, Zhao et al. [18] introduced the fault influence of interactive entities and developed a novel synthetical fault localization approach based on the software network. Wu et al. [19] adopted OPTICS clustering to group failed test cases, the failed test cases in this cluster, with all passed test cases to locate a single-bug.

For each statement s, these data can be represented as a vector of four elements, $n(s) = < n_{ep}, n_{up}, n_{ef}, n_{uf} >$, where n_{ep} and n_{ef} denote the number of passed and failed test cases executing statement s, respectively; n_{up} and n_{uf} represent the number of passed and failed test cases not executing s, respectively. Obviously, the sum of test suite T is equal to the sum of these four parameters for each statement.

Many suspiciousness formulas, such as those of Tarantula [20,21], Jaccard [22], and Ochiai [23], are designed in order to make the faulty statements at the top of the ranking list. Generally speaking, different formulas were designed for various purposes in terms of different intuitions. With more and more formulas proposed, researchers started to study the factors of the performance of different formulas. In these studies [9,24], coincidental correctness is considered whether or not it can affect the effectiveness of fault localization.

2.2. Coincidental Correctness (CC)

Definition 1 (Coincidental correctness element cc_e)**.** *Given a program element e, $f_T(e)$ represents the ratio of failed test cases executing e, and $p_T(e)$ denotes the ratio of passed test cases executing e. A good candidate for coincidental correctness element cc_e is that a program element occurs in all failed runs and in a non-zero but not excessively large percentage of passing runs. A coincidental correctness element cc_e is defined by a characteristic function f(e) as follows:*

$$f(e) = \begin{cases} cc_e\,, & f_T(e) = 1 \wedge 0 < p_T(e) < 1 \\ 0\,, & otherwise \end{cases} \tag{1}$$

Definition 2 (Coincidental correctness test cases)**.** *A test case executed coincidental correctness element cc_e but did not result in a failure. Coincidental correctness test cases are denoted as T_{CC} and the size of T_{CC} is more than or equal to 0.*

A test suite T comprises a set of failed test cases T_F and a set of passed test cases T_P, where T_P might contain a subset of true passed test cases T_{trueP} and another subset of coincidental correctness test cases T_{CC}. Assume that the test suite contains n coincidental correctness test cases.

There are three locations of program fault in Figure 3. One fault only appears in the failed execution trace TRf, and there, accidental correctness did not occur,as shown in Figure 3a. When one fault both appears in the failed execution trace TRf and the passed execution trace TRp, accidental correctness occurs, as shown in Figure 3b. However, one fault only presents in the passed execution trace TRp and the fault could not be found in the case in Figure 3c.

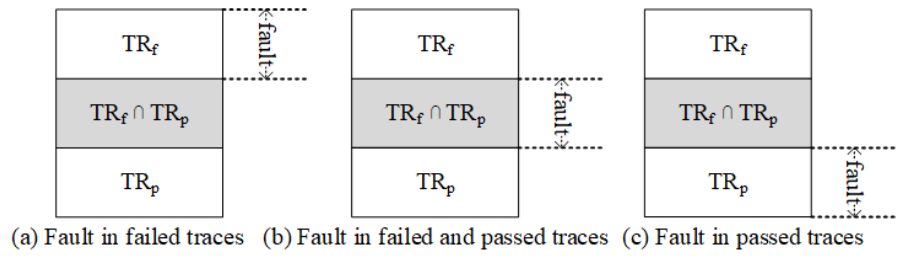

Figure 3. Three possible locations of the fault.

2.3. Assumptions

Before proposing our theoretical analysis framework of the suspiciousness formulas, we express several assumptions as follows.

- We assume that the SBFL formulas are applied to the program with test oracle; that is to say, the execution result of the program is failed or passed for any test case in premise.
- We assume that the program only contains one fault. In other words, we investigated 30 suspiciousness formulas under the single-fault scenario because multiple faults will interfere with each other.
- The test suite is assumed to contain at least one failed test case because there existed a failed test case for triggering the fault in the program. For any program statement s, we have $n_{ef} > 0$.

3. Our Framework

3.1. Motivation

All of this motivated us to investigate techniques to clean coincidental correct test cases to enhance SBFL. Coincidental correctness may degrade the effectiveness of fault localization [2,5]. We will use an example program shown in Figure 4 to explain the effect of coincidental correctness on fault localization. The function `foo()` returns the calculated results of three input integers. We generate six test cases (dubbed t_1 to t_6), whose values are as follows: $(6,3,9)$, $(2,4,6)$, $(4,0,3)$, $(5,2,0)$, $(5,4,2)$, $(5,0,2)$. We manually seed a fault into statement s_5, as shown in Figure 4.

```
     foo ( ) {
s1       int x, y, z;
s2       int ret = 10;
s3       read (x,y,z);
s4       if (x >= y) {
s5       ret = x-y; // correct: ret = x + y;
s6       if (y > z)
s7       ret = ret + 10;
         }
s8   return ret;
     }
```

Figure 4. A motivating example.

The symbol '•' denotes that a statement is executed under test in Table 1. Column 1 lists the number of program statements. Column `Coverage` presents coverage information in the original test suite and column `Coverage(CC)` presents coverage information in the test suite removing coincidental correctness test cases. Test cases t_3 and t_6 are coincidental correctness test cases by the definition. Columns 3 and 4 show the suspiciousness of each statement and the corresponding rank using Tarantula [20,21] and Naish1 [14] based on coverage information, respectively. Columns 6 and 7 show the suspiciousness of each statement and the corresponding rank using Tarantula and Naish1 based on coverage information after removing coincidental correctness test cases, respectively. The last row shows the number of the examined statements to locate the fault in the best case and in the worst case. The second to last row shows the result of the passed execution (dubbed "T") and failed execution(dubbed "F").

In columns 3 and 6, the suspiciousness value of statements s_5 , s_6 is changed from 0.6

to 1 and that of the other statements is unchanged based on the original coverage information and coverage information after removing coincidental correctness test cases using Tarantula. We need to examine one to three statements to locate a fault based on coverage information after removing coincidental correctness test cases using Tarantula. However, we need to examine two to three statements based on coverage information using Tarantula. It can be seen that removing coincidental correctness test cases can improve the effectiveness of Tarantula. Whereas, in columns 4 and 7, the suspiciousness value of all statements is unchanged using Naish1. It can be seen that removing coincidental correctness test cases cannot improve the effectiveness of Naish1. This is because Naish1 is already good enough, but this is not caused by removing CC.

The motivation of our paper is similar to a recent paper which presented a theoretical analysis of the risk evaluation formulas for fault localization [14]. However, they focus on the relationship between two formulas; we pay attention to the impact of the effectiveness of two formulas when removing coincidental correctness test cases. In additionally, the literature [3] used empirical methods to analyze the impact of coincidental correctness. Abou Assi et al. [7] conducted a study that aimed at assessing the impact of coincidental correctness, on the effectiveness of SBFL, in both of its forms, weak and strong. Our approach is different from them.

After the pioneering work by Masri et al.[3], who proposed to demonstrate the prevalence of coincidental correctness test cases, many methods have been presented for coincidental correctness, such as the following studies [4,5,10,11]. In our model, a sound method is presented to theoretically prove whether the formulas are impacted by coincidental correctness. Our paper aims to develop a function theoretical framework to investigate suspiciousness formulas when removing coincidental correctness.

Table 1. Fault localization affected by coincidental correctness on a motivating example.

Statements	**Coverage**						**Tarantula**		**Naish1**		**Coverage (CC)**				**Tarantula**		**Naish1**	
	t_1	t_2	t_3	t_4	t_5	t_6	**sus.**	**rank**	**sus.**	**rank**	t_1	t_2	t_4	t_5	**sus.**	**rank**	**sus.**	**rank**
s_1	•	•	•	•	•	•	0.5	4	0	3	•	•	•	•	0.5	4	0	3
s_2	•	•	•	•	•	•	0.5	4	0	3	•	•	•	•	0.5	4	0	3
s_3	•	•	•	•	•	•	0.5	4	0	3	•	•	•	•	0.5	4	0	3
s_4	•	•	•	•	•	•	0.5	4	0	3	•	•	•	•	0.5	4	0	3
s_5 (fault)	•		•	•	•	•	0.6	2	1	1	•		•	•	1	1	1	1
s_6	•		•	•	•	•	0.6	2	1	1	•		•	•	1	1	1	1
s_7				•	•		1	1	−1	8			•	•	1	1	−1	8
s_8	•	•	•	•	•	•	0.5	4	0	3	•	•	•	•	0.5	4	0	3
Result	F	T	T	F	F	T					F	T	F	F				
Fault rank								2–3		1–2						1–3		1–2

3.2. Theoretical Analysis

Generally speaking, the derivation is a calculation formula in mathematical calculation. The derivation is not only the basis of calculus but also an important pillar of calculus calculation. It is defined as the limit of the quotient between the increment of the dependent variable and the increment of the independent variable when the increment of the independent variable tends to zero. When a function has a derivative, it is said that the function is differentiable or differentiable. In mathematics, the derivation of a function is expressed as $f'(x)$. If the functions $u(x)$ and $v(x)$ are derivable, the formula $f'(x)$ is described as follows.

$$f'(x) = (\frac{u(x)}{v(x)})' = (\frac{u(x)'v(x) - u(x)v(x)'}{v^2(x)}) \quad (2)$$

We removed and discarded coincidental correctness test cases from Tp just like in the literature [2,7] in order to improve the effectiveness of SBFL. The SBFL formula (referred to as $N(e)$ in this paper) will arrive at a more faithful value when subtracting n from n_{ep}. Formula $N(e)$ is marked as $N(e)'$ when $N(e)$ is computed on the test suite removing coincidental correctness. A more faithful value of $N(e)$ means that fault localization is more accurate. Therefore, researchers expect to improve the suspiciousness value of formula $N(e)$ by removing coincidental correctness under the single-fault scenario. Our framework is based on the concept that the determinant for the effectiveness of a formula is the number of statements with risk values higher than the risk value of the faulty statement. Our framework is based on the concept of the derivative that suspiciousness formulas are considered as the function of variable n_{ep}, which is the number of passed test cases executing statement s, and the derivative of function $f(n_{ep})$ is computed to determine whether the value of the derivative is greater than 0.

Considering formula $N(e)$ as a function of variable n_{ep}, we propose a function framework to compute the derivative of function $f(n_{ep})$ (i.e., formula (3)) to investigate the effect of coincidental correctness on SBFL formulas. For formulas $N(e)$, the numerator is denoted as u, and the denominator is denoted as v.

$$f'(n_{ep}) = (\frac{u}{v})' = (\frac{u'v - uv'}{v^2}) \tag{3}$$

We classified that there are three cases for the value of $f'(n_{ep})$, as shown in Table 2.

Table 2. The relationship of monotonic suspiciousness function.

Function	Variable	Derivative	Monotonic	Function Relationship	Formula Type
$f(n_{ep})$	n_{ep}	less than 0	Monotonically decreasing	$f(n_{ep} - n) > f(n_{ep})$	T_1
	constant	—	—	$f(n_{ep} - n) = f(n_{ep})$	T_2
	n_{ep}	less than 0 or more than 0	Monotonically decreasing or increasing	$f(n_{ep} - n) > f(n_{ep})$ *or* $f(n_{ep} - n) < f(n_{ep})$	T_3

1. "T_1—improved typ". When $f'(n_{ep}) < 0$, $f(n_{ep})$ is a monotonically decreasing function to discriminate $f(n_{ep} - n) > f(n_{ep})$, i.e., $N(e)' \succeq N(e)$. That is, when removing coincidental correctness test cases, the suspiciousness value of the SBFL formulas is increasing; therefore, the effectiveness of the SBFL formulas is improved.
2. "T_2—invariant type". When $f'(n_{ep})$ is equal to 0, $f(n_{ep})$ is to discriminate $f(n_{ep} - n) = f(n_{ep})$, i.e., $N(e)' = N(e)$. That is, when removing coincidental correctness test cases, the suspiciousness value of SBFL formulas is unchanged; therefore, the effectiveness of the SBFL formulas is unchanged.
3. "T_3—uncertain type". It is uncertain that the value of $f'(n_{ep})$ is more than 0 or less than 0. The value of $f'(n_{ep})$ needs to be discussed inter-partition. That is, when removing coincidental correctness test cases, the suspiciousness value of SBFL formulas is uncertain; therefore, the effectiveness of SBFL formulas is uncertain.

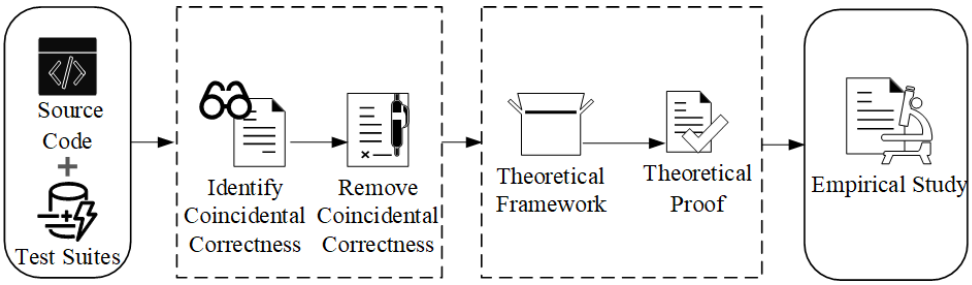

Figure 5. The block diagram of our approach.

In this paper, the block diagram of our approach is shown in Figure 5. Coincidental correctness test cases are removed from the whole test cases. We proposed a theoretical framework and theoretically proved 30 suspiciousness formulas by coincidental correctness, which were selected from Naish et al. [25], because their theoretical analysis for suspiciousness formulas is the most systematic and comprehensive one. Finally, we evaluated the impact of the effectiveness of SBFL techniques (i.e. Tarantula, Ochiai, Russel&Rao, Naish2, Jaccard, Naish1, Wong1) by coincidental correctness.

3.2.1. The Improved Formulas by Removing Coincidental Correctness Test Cases

Through the theoretical analysis framework we proposed, we investigated the 30 suspiciousness formulas in Table 3. The formulas, whose type is T_1, are the improved formulas by removing coincidental correctness. We mainly prove Jaccard, Goodman, Naish2, Wong3, Fleiss, Tarantula, Ochiai and Arithmetic Mean of "T_1-improved type" formulas by the framework proposed in the previous section.

Table 3. The investigated formulas by removing CC test cases.

Name	Formula Expression	Type
Jaccard	$\frac{n_{ef}}{n_{ef}+n_{uf}+n_{ep}}$	T_1
Anderberg	$\frac{n_{ef}}{n_{ef}+2(n_{uf}+n_{ep})}$	T_1
SØrensen-Dice	$\frac{2n_{ef}}{2n_{ef}+n_{uf}+n_{ep}}$	T_1
Dice	$\frac{2n_{ef}}{n_{ef}+n_{uf}+n_{ep}}$	T_1
qe	$\frac{n_{ef}}{n_{ef}+n_{ep}}$	T_1
Simple Matching	$\frac{n_{ef}+n_{up}}{n_{ef}+n_{uf}+n_{ep}+n_{up}}$	T_1
Sokal	$\frac{2(n_{ef}+n_{up})}{2(n_{ef}+n_{up})+n_{uf}+n_{ep}}$	T_1
Rogers&Tanimoto	$\frac{n_{ef}+n_{up}}{n_{ef}+n_{up}+2(n_{uf}+n_{ep})}$	T_1
Russel&Rao	$\frac{n_{ef}}{n_{ef}+n_{uf}+n_{ep}+n_{up}}$	T_1
M2	$\frac{n_{ef}}{n_{ef}+n_{up}+2(n_{uf}+n_{ep})}$	T_1
Kulczynski2	$\frac{1}{2}(\frac{n_{ef}}{n_{ef}+n_{uf}}+\frac{n_{ef}}{n_{ef}+n_{ep}})$	T_1
Rogot1	$\frac{1}{2}(\frac{n_{ef}}{2n_{ef}+n_{uf}+n_{ep}}+\frac{n_{up}}{2n_{up}+n_{uf}+n_{ep}})$	T_1
Goodman	$\frac{2n_{ef}-n_{uf}-n_{ep}}{2n_{ef}+n_{uf}+n_{ep}}$	T_1
Hamann	$\frac{n_{ef}+n_{up}-n_{uf}-n_{ep}}{n_{ef}+n_{uf}+n_{ep}+n_{up}}$	T_1
Naish2	$n_{ef}-\frac{n_{ep}}{n_{ep}+n_{up}+1}$	T_1

Table 3. *Cont.*

Name	Formula Expression	Type
AMPLE2	$\frac{n_{ef}}{n_{ef}+n_{uf}} - \frac{n_{ep}}{n_{ep}+n_{up}}$	T_1
Wong3	$n_{ef} - h$, *where* $h = \begin{cases} n_{ep}, & \text{if } n_{ep} \leqslant 2 \\ 2+0.1(n_{ep}-2), & \text{if } 2 < n_{ep} \leqslant 10 \\ 2.8+0.001(n_{ep}-10), & \text{if } n_{ep} > 10 \end{cases}$	T_1
Wong2	$n_{ef} - n_{ep}$	T_1
Fleiss	$\frac{4n_{ef}n_{up} - 4n_{uf}n_{ep} - (n_{uf}-n_{ep})^2}{(2n_{ef}+n_{uf}+n_{ep}) + (2n_{up}+n_{uf}+n_{ep})}$	T_1
Tarantula	$\frac{n_{ef}}{n_{ef}+n_{uf}} / (\frac{n_{ef}}{n_{ef}+n_{uf}} + \frac{n_{ep}}{n_{ep}+n_{up}})$	T_1
Ochiai	$\frac{n_{ef}}{\sqrt{(n_{ef}+n_{uf})(n_{ef}+n_{ep})}}$	T_1
Arithmetic Mean	$\frac{2n_{ef}n_{up} - 2n_{uf}n_{ep}}{(n_{ef}+n_{ep})(n_{up}+n_{uf}) + (n_{ef}+n_{uf})(n_{ep}+n_{up})}$	T_1
Mean	$\frac{2n_{ef}n_{up} - 2n_{uf}n_{ep}}{(n_{ef}+n_{ep})(n_{up}+n_{uf}) + (n_{ef}+n_{uf})(n_{ep}+n_{up})}$	T_1
Naish1	$\begin{cases} -1\,, & \text{if } n_{ef} < F \\ P - n_{ep}\,, & \text{if } n_{ef} = F \end{cases}$	T_2
Wong1	n_{ef}	T_2
Binary	$\begin{cases} 0\,, & \text{if } n_{ef} < F \\ 1\,, & \text{if } n_{ef} = F \end{cases}$	T_2
Hamming etc.	$n_{ef} + n_{up}$	T_2
Euclid	$\sqrt{n_{ef}+n_{up}}$	T_2
CBI Inc.	$\frac{n_{ef}}{n_{ef}+n_{ep}} - \frac{n_{ef}+n_{uf}}{n_{ef}+n_{uf}+n_{ep}+n_{up}}$	T_3
Cohen	$\frac{2n_{ef}n_{up} - 2n_{uf}n_{ep}}{(n_{ef}+n_{ep})(n_{up}+n_{ep}) + (n_{ef}+n_{uf})(n_{uf}+n_{up})}$	T_3

Theorem 1. *Jaccard*$' \succeq$ *Jaccard.*

Proof of Theorem 1. $f'(n_{ep}) = (\frac{n_{ef}}{n_{ef}+n_{uf}+n_{ep}})' = \frac{1}{v^2}(-n_{ef})$ (4)

It is clear that the numerator is less than 0 and the denominator is greater than 0, so $f'(n_{ep}) < 0$. $f(n_{ep})$ is a decreasing function; therefore, $f(n_{ep}-n) > f(n_{ep})$, that is *Jaccard*$' \succeq$ *Jaccard*. □

Theorem 2. *Goodman*$' \succeq$ *Goodman.*

Proof of Theorem 2. $f'(n_{ep}) = (\frac{2n_{ef}-n_{uf}-n_{ep}}{2n_{ef}+n_{uf}+n_{ep}})' = \frac{1}{v^2}[-v-u] = \frac{1}{v^2}(-4n_{ef})$ (5)

The numerator is less than 0 and the denominator is greater than 0, so $f'(n_{ep}) < 0$. $f(n_{ep})$ is a decreasing function; therefore, $f(n_{ep}-n) > f(n_{ep})$, that is *Goodman*$' \succeq$ *Goodman*. □

Theorem 3. *Naish*2$' \succeq$ *Naish*2*.*

Proof of Theorem 3. $f'(n_{ep}) = (n_{ef} - \frac{n_{ep}}{n_{ep}+n_{up}+1})' = \frac{1}{v^2}(-2n_{ep} - n_{up} - 1)$ (6)

The numerator is less than 0 and the denominator is greater than 0, so $f'(n_{ep}) < 0$. $f(n_{ep})$ is a decreasing function; therefore, $f(n_{ep} - n) > f(n_{ep})$, that is $Naish2' \succeq Naish2$. □

Theorem 4. $Wong3' \succeq Wong3$.

Proof of Theorem 4. If $n_{ep} \leqslant 2$, then we have $f'(n_{ep}) = -1$.

If $2 < n_{ep} \leqslant 10$, then we have $f'(n_{ep}) = -0.1$.

If $n_{ep} > 10$, then we have $f'(n_{ep}) = -0.001$.

No matter what value of n_{ep} is, $f'(n_{ep})$ is always less than 0; therefore, $f(n_{ep} - n) > f(n_{ep})$, that is $Wong3' \succeq Wong3$. □

Theorem 5. $Fleiss' \succeq Fleiss$.

Proof of Theorem 5. $f'(n_{ep}) = (\frac{u}{v})' = (\frac{u'v - uv'}{v^2}) = \frac{1}{v^2}[(-2n_{uf} - 2n_{ep})v - 2u]'$ (7)

$= \frac{1}{v^2}[-4(n_{uf} + n_{ep})^2 - 4(n_{uf} + n_{ep})(n_{ef} + n_{up}) - 8n_{ef}n_{up} + 2(n_{uf} + n_{ep})^2]$ (8)

$= \frac{1}{v^2}[-2(n_{uf} + n_{ep})^2 - 4(n_{uf} + n_{ep})(n_{ef} + n_{up}) - 8n_{ef}n_{up}]$ (9)

It is obvious that $f'(n_{ep}) < 0$. $f(n_{ep})$ is a decreasing function; therefore, $f(n_{ep} - n) > f(n_{ep})$, that is $Fleiss' \succeq Fleiss$. □

Theorem 6. $Tarantula' \succeq Tarantula$.

Proof of Theorem 6. Formula Tarantula can be re-written as

$f(n_{ep}) = \frac{n_{ef}n_{ep} + n_{ef}n_{up}}{n_{ep}(2n_{ef} + n_{uf}) + n_{ef}n_{up}}$ (10)

$f'(n_{ep}) = (\frac{u}{v})' = (\frac{u'v - uv'}{v^2}) = \frac{1}{v^2}[n_{ef}v - (n_{ef}n_{ep} + n_{ef}n_{up})(2n_{ef} + n_{uf})]$ (11)

$= \frac{1}{v^2}(-n_{ef}^2 n_{up} - n_{ef}n_{up}n_{uf})$ (12)

It is clear that $f'(n_{ep}) < 0$. $f(n_{ep})$ is a decreasing function; therefore, $f(n_{ep} - n) > f(n_{ep})$, that is $Tarantula' \succeq Tarantula$. □

Theorem 7. $Ochiai' \succeq Ochiai$.

Proof of Theorem 7. $f'(n_{ep}) = (\frac{n_{ef}}{\sqrt{(n_{ef} + n_{uf})(n_{ef} + n_{ep})}})'$ (13)

$= \frac{-n_{ef}v^{-1/2}}{2v^2} = \frac{-n_{ef}}{2v^{5/2}}$ (14)

It is clear that $f'(n_{ep}) < 0$. $f(n_{ep})$ is a decreasing function; therefore, $f(n_{ep} - n) > f(n_{ep})$, that is $Ochiai' \succeq Ochiai$. □

Theorem 8. $Arithmetic\ mean' \succeq Arithmetic\ mean$.

Proof of Theorem 8. $f'(n_{ep}) = (\frac{u}{v})' = (\frac{u'v - uv'}{v^2})$ (15)

$= \frac{1}{v^2}[-2n_{uf}v - (2n_{ef}n_{up} - 2n_{uf}n_{ep})(n_{np} + 2n_{uf} + n_{ef})]$ (16)

$= \frac{1}{v^2}[-2n_{uf}(4n_{ef}n_{up} + n_{ef}n_{uf} + n_{uf}n_{up}) - 2n_{ef}n_{up}^2 - 2n_{ef}^2 n_{up}]$ (17)

Because the molecular is less than 0, $f'(n_{ep}) < 0$. $f(n_{ep})$ is a decreasing function; therefore, $f(n_{ep} - n) > f(n_{ep})$, and we can prove that $Arithmetic\ mean' \succeq Arithmetic\ mean$. □

3.2.2. The Unaffected Formulas by Removing Coincidental Correctness Test Cases

Theorem 9. $Naish1' = Naish1$.

Proof of Theorem 9. Because $P = n_{ep} + n_{up}$, Naish1 is transformed into the following form:

$$f(n_{ep}) = \begin{cases} -1\,, & \text{if } n_{ef} < F \\ n_{up}\,, & \text{if } n_{ef} = F \end{cases} \tag{18}$$

It is clear that Naish1 is not related with variable n_{ep}; thus, $Naish1' = Naish1$. Through observing the parameters of formulas in Table 3, we found that these formulas, namely, Wong1, Binary, Hamming, Euclid, etc., have no relationship with variable n_{ep}. Therefore, removing coincidental correctness test cases from test suites does not affect the SBFL formulas, which type is T_2 in Table 3. □

3.2.3. Uncertain of Formulas Affected by Removing Coincidental Correctness Test Cases

Theorem 10. $\begin{cases} CBI\ Inc' \succeq CBI\ Inc\,, & if\ 0 < n_{ep} < x_2 \\ CBI\ Inc' \prec CBI\ Inc\,, & if\ x_2 < n_{ep} \\ CBI\ Inc' = CBI\ Inc\,, & if\ n_{ep} = x_2 \end{cases}$

Proof of Theorem 10 .

$$f'(n_{ep}) = \frac{-n_{ef}}{(n_{ef}+n_{ep})^2} + \frac{n_{ef}+n_{uf}}{(n_{ef}+n_{uf}+n_{ep}+n_{up})^2} \tag{19}$$

$$= \frac{1}{(n_{ef}+n_{ep})^2(n_{ef}+n_{uf}+n_{ep}+n_{up})^2} g(n_{ep}) \tag{20}$$

Suppose $g(n_{ep}) = n_{uf}n_{ep}^2 - 2n_{ef}n_{up}n_{ep} - (n_{uf}n_{ef}^2 + 2n_{up}n_{ef}^2 + n_{ef}n_{uf}^2 + 2n_{ef}n_{uf}n_{up} + n_{ef}n_{up}^2)$. □

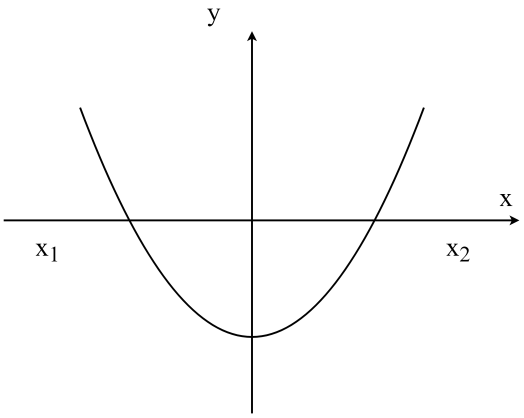

Figure 6. The parabola of function $g(n_{ep})$.

The denominator in formulas is more than 0; therefore, $f'(n_{ep})$ is determined by $g(n_{ep})$. Suppose $g(n_{ep}) = 0$, $a = n_{uf}$, $b = -2n_{ef}n_{up}$, $c = -(n_{uf}n_{ef}^2 + 2n_{up}n_{ef}^2 + n_{ef}n_{uf}^2 + 2n_{ef}n_{uf}n_{up} + n_{ef}n_{up}^2)$. We have a quadratic equation $g(n_{ep}) = an_{ep}^2 + bn_{ep} + c = 0$. We adopt $\triangle = b^2 - 4ac$ to judge the solutions of the equation. $\triangle$ is more than 0, and the equation has a positive or negative solution because the constant c is less than 0. We suppose two solutions of equation are x_1 and x_2 (i.e., $x_1 < 0 < x_2$), as shown in Figure 6. We discuss three cases as follows:

(1) Suppose $0 < n_{ep} < x_2$, we have $g(n_{ep}) < 0$, i.e., $f'(n_{ep}) < 0$, thus, $f(n_{ep} - n) > f(n_{ep})$, that is, $CBI\ Inc' \succeq CBI\ Inc$;

(2) Suppose $n_{ep} > x_2$, we have $g(n_{ep}) > 0$, i.e., $f'(n_{ep}) > 0$, thus, $f(n_{ep} - n) < f(n_{ep})$, that is, $CBI\ Inc' \prec CBI\ Inc$;

(3) Suppose $n_{ep} = x_2$, we have $g(n_{ep}) = 0$, i.e., $f'(n_{ep}) = 0$, thus, $f(n_{ep} - n) = f(n_{ep})$, that is, $CBI\ Inc' = CBI\ Inc$.

Theorem 11. $\begin{cases} Cohen' \succeq Cohen\ , & if\ 0 < n_{ep} < x_2 \\ Cohen' \prec Cohen\ , & if\ x_2 < n_{ep} \\ Cohen' = Cohen\ , & if\ \ n_{ep} = x_2\ or\ \ n_{ep} = x_2 \end{cases}$
$Cohen' > Cohen$.

Proof of Theorem 11. $v = n_{ep}^2 + (n_{ef} + n_{up})n_{ep} + n_{ef}n_{up} + (n_{ef} + n_{uf})(n_{uf} + n_{up})$ (21)

$v' = 2n_{ep} + (n_{ef} + n_{up})$ (22)

$f'(n_{ep}) = (\frac{u}{v})' = (\frac{u'v - uv'}{v^2}) = \frac{1}{v^2}(-2n_{uf}v - uv') = \frac{1}{v^2}g(n_{ep})$ (23)

$g(n_{ep}) = 2n_{uf}n_{ep}^2 - 4n_{ef}n_{up}n_{ep} - (2n_{ef}^2 n_{up} + 2n_{ef}n_{up}^2 + 2n_{uf}n_{ef}n_{up} + 2n_{uf}(n_{ef} + n_{uf})(n_{uf} + n_{up}))$ (24)

The denominator v^2 is more than 0; hence $f'(n_{ep})$, is determined by $f(n_{ep})$. Suppose $g(n_{ep}) = 0$, we adopt $\triangle = b^2 - 4ac$ to judge the solution of the equation. $\triangle$ is more than 0, and the equation has a positive or negative solution, because the constant is less than 0. We suppose that two solutions of the equation are x_1 and x_2 (i.e., $x_1 < 0 < x_2$). Consider the following three cases

(1) Assume $0 < n_{ep} < x_2$, then $f(n_{ep}) < 0$, i.e., $f'(n_{ep}) < 0$, we have $f(n_{ep} - n) > f(n_{ep})$, that is, $Cohen' > Cohen$;
(2) Assume $n_{ep} > x_2$, then $f(n_{ep}) > 0$, i.e., $f'(n_{ep}) > 0$, we have $f(n_{ep} - n) < f(n_{ep})$, that is, $Cohen' \prec Cohen$;
(3) Assume $n_{ep} = 0$ ro $n_{ep} = x_2$, then $f(n_{ep}) = 0$, i.e., $f'(n_{ep}) = 0$, we have $f(n_{ep} - n) = f(n_{ep})$, that is, $Cohen' = Cohen$.

In all, the effectiveness of formulas $CBI\ Inc$ and $Cohen$ (Type T_3 in Table 3) can be improved by cleansing removing coincidental correctness test cases, which depends on the range of n_{ep}. If n_{ep} is less than x_2, then the effectiveness of them can be improved by cleansing removing coincidental correctness test cases. Otherwise, the case would be the contrary. □

3.3. Empirical Study

In this section, we evaluated the effectiveness of "T_1- improved type" formulas (e.g., Tarantula [20] and Naish2 [14]) and "T_2- invariant type" formulas (e.g., Naish1 [14] and Wong [24]) based on coverage information using test suite T and test suite T discarding T_{CC}. To obtain coincidental correctness test cases accurately, we instrumented the faulty statement of a program that sets a flag as soon as the faulty statement is met. That is, coincidental correctness arises when the faulty statement is met in the passed execution. After that, removing coincidental correctness test cases, we verified the effectiveness of four fault localization techniques. Given T_F and T_P, our aim is to identify T_{CC} so that test cases T_{CC} would be discarded from T in order to verify whether it can improve the effectiveness of SBFL.

3.3.1. Subject Programs

Table 4 lists the program name, description information, lines of code not including non-blank and non-comment in the base versions, number of faulty versions, test suite size and fault type. These programs are written in C language. All four unix programs along with their test suites are obtained from SIR (Software Infrastructure Repository) [26]. To evaluate our technique, we use four UNIX programs as subject programs. The gzip has the real faults with 211 test cases, which is a file compression/decompression tool with 5365 lines of code. The grep has the real faults with 806 test cases, which is a text search tool to find one or more input files for lines with 9205 lines of code. The sed has the real and seeded faults with 360 test cases, which is a stream editor to filter the text with 6763 lines of code. The flex has the real faults with 567 test cases and a lexical analyzer generator

with 9766 lines of code. Among them, we tested 20 faulty versions of the gzip program and 22 faulty versions of the sed program; similarly, we test 14 faulty versions of the grep program and 20 faulty versions of the flex program. We conducted the experiments on these programs because they were utilized in related work. In addition, faulty versions with no failed test cases were excluded from our experiments because testing a fault required one failed test case at least. In all, 76 faulty versions of relatively large programs were tested for the experiments. Our experiments are run on a Linux server with a 3.07 GHz Intel(R) Xeon(R) CPU and 16G memory. The operating system is Ubuntu 12. 04. 2 with Open-JDK 1.7 installed and the compiler is gcc-4.6.3 [27].

Table 4. Subject programs.

Program Name	Description Information	Lines of Code	Number of Faulty Version	Test Suite Size	Fault Type
gzip	data compression/decompression	5365	20	211	real
grep	search for a pattern in a file	9205	14	806	real
sed	a stream text editor	6763	22	360	real, seeded
flex	a fast lexical analyzer generator	9766	20	567	real

3.3.2. Results and Analysis

For a visual comparison, we adopted EXAM score [28], the ratio of the percentage of faults located to the percentage of code examined, to assess the effectiveness of fault localization by line chart. Figure 7 illustrates the effectiveness comparisons between SBFL using test suite T and test suite T discarding T_{CC} in the faulty versions of the program. For clarity, the horizontal axis shows the percentage of code examined, whereas the vertical axis denotes the percentage of the faults located. As shown in Figure 7, the evaluated SBFL techniques based on coverage information using test suite T discarding T_{CC} are marked as Tarantula (CC), Ochiai (CC), Russel&Rao(CC), Naish2(CC), Jaccard(CC), Naish1(CC), Wong1(CC), while the SBFL techniques based on coverage information using test suite T are marked as Tarantula, Ochiai, Russel&Rao, Naish2, Jaccard, Naish1, and Wong1.

As can be seen from Figure 7, the curve of SBFL based on coverage information using test suite T discarding T_{CC} is higher than that of SBFL based on coverage information using test suite T. This suggests that removing coincidental correctness test cases can significantly enhance the effectiveness of "T_1—improved Type" formulas, while removing coincidental correctness test cases does not improve the effectiveness of "T_2—invariant Type" formulas. For example, by examining approximately 10% of the code, in Figure 7a, Tarantula(CC) can locate 82.78% of the faults, whereas Tarantula can only locate 79.16%; in Figure 7b, Ochiai(CC) can locate 71.261%, whereas Ochiai can only locate 59.425%; in Figure 7c, Russel&Rao(CC) can locate 76.113%, whereas Russel&Rao can only locate 66.105%; in Figure 7d, Naish2(CC) can locate 81.89%, whereas Naish2 can only locate 73.88%; in Figure 7e, Jaccard(CC) can locate 70.261%, whereas Jaccard can only locate 58.424%; in Figure 7f, both Naish1(CC) and Naish1 can locate 91.66%, and both Wong1(CC) and Wong1 can locate 77.78%.

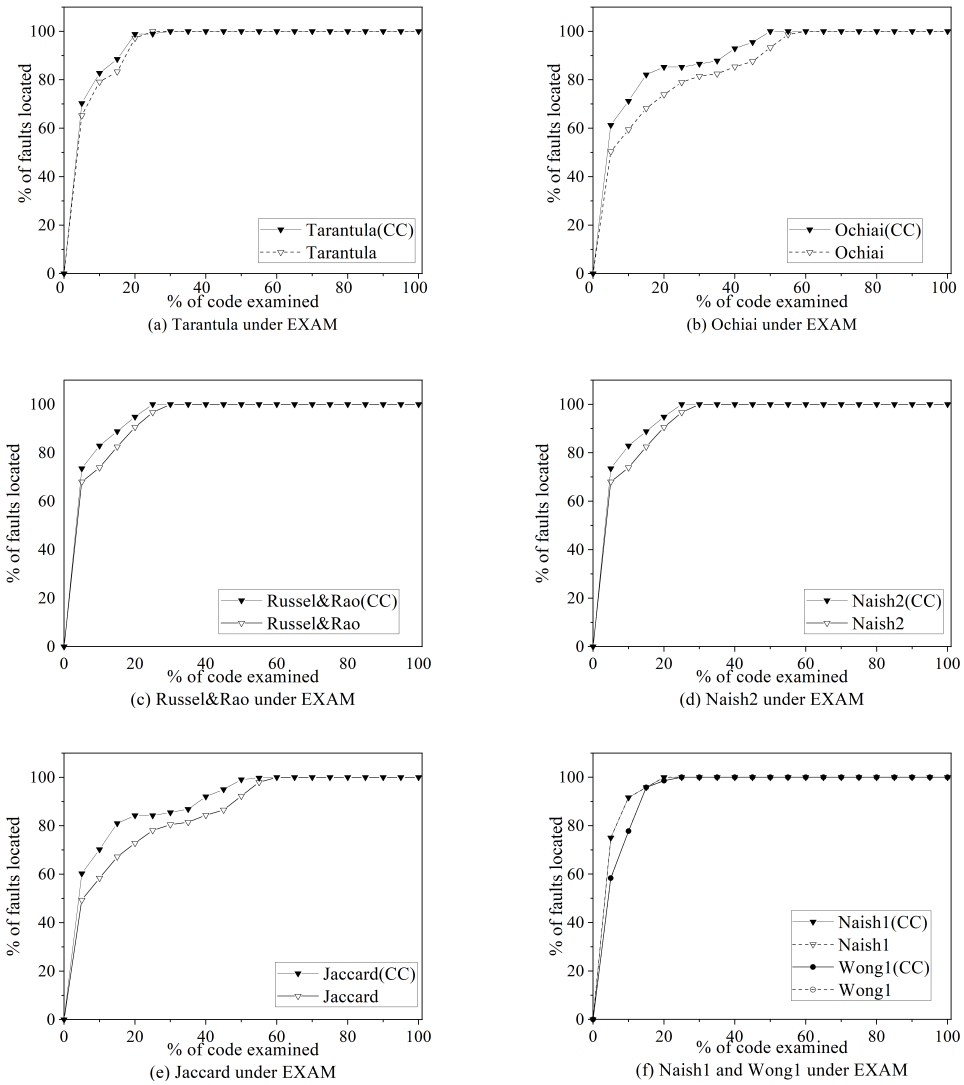

Figure 7. EXAM score comparisons on all four subjects.

Through the empirical study, we demonstrated that coincidental correctness sometimes does degrade the effectiveness of SBFL, such as Tarantula and Naish2, and that coincidental correctness sometimes does not degrade the effectiveness of SBFL, such as Naish1 and Wong1. However, the effectiveness of "T_3—uncertain type" formulas affected by coincidental correctness depends on the range of passed test cases and inter-partition.

Table 5 shows the time overhead of SBFL(CC) and SBFL in all faulty versions of each subject. The evaluated SBFL techniques based on coverage information using test suite T discarding T_{CC} are marked as SBFL(CC). SBFL(CC) consumes the time to collect coverage information, to cleanse coincidental correctness test cases and to compute the suspiciousness. In contrast, SBFL consumes the time to obtain coverage information and compute the suspiciousness. Without loss of generality, we computed the total time overhead of Tarantula, which is a presentative of SBFL. In Table 5, column Trace&Cleaning(s) denotes the time overhead on all test cases by collecting coverage information and cleansing coincidental correctness, column Susp.computation illustrates the time to compute the suspiciousness in both cases, and column Total(s) means the total time of fault localization. As seen in Table 5, SBFL(CC) requires additional time to identify accidental correctness test case; therefore, it has high time overhead. In our experiments, the last column indicates that the ratio of the total time of SBFL(CC) in all faulty versions for each subject is 1.03 to 1.25 times that of SBFL.

Table 5. The time overhead of SBFL(CC) and SBFL in each subject with all test cases(s).

Program	Trace and Cleaning(s)		Susp. Computation(s)		Total(s)		#SBFL(CC)
	SBFL(CC)	SBFL	SBFL(CC)	SBFL	SBFL(CC)	SBFL	#SBFL
gzip	6912.36	6738.39	303.27	291.06	2405.21	2343.15	1.03
grep	33,617.52	28,025.28	1512.96	1386.80	4391.31	3676.51	1.19
sed	6922.23	6541.32	288.15	282.06	2403.46	2274.46	1.06
flex	44,072.20	35,001.90	1950.20	1752.40	4602.24	3675.43	1.25

3.3.3. Discussion

We assumed that at least one failed test case and one passed test case were contained in the test suite. It is feasible and reasonable because such an assumption is widely accepted and required for debugging. One failed test case can trigger the fault, and one passed test case is needed for the computation of the suspiciousness. In addition, we have the assumption that the programs under test have a test oracle for suspiciousness formulas, that is, the execution result of either failed or passed can be decided. Through theoretical analysis and empirical study, we investigated the 30 suspiciousness formulas.

The work [3] showed that removing coincidental correctness test cases was a safety reducing factor for SBFL formulas, in which they only considered Jaccard, Tarantula, AMPLE, and Ochiai formulas, and they did not investigate the effect of other SBFL formulas by removing coincidental correctness. Our theoretical study showed that 23 of the 30 suspiciousness formulas were improved by removing coincidental correctness. However, through our theoretical analysis, five of these 30 SBFL techniques are not at all affected by coincidental correctness, and two of them are affected by coincidental correctness depending on certain conditions. Cleansing coincidental correctness test cases from the test suites is not meaningful and time-consuming in the above two cases.

(1) We assume that the program only contains one fault in the proof. In addition, we investigated 30 suspiciousness formulas under the single-fault scenario because of multiple faults interfering with each other.
(2) We conducted the experiments to verify the effectiveness of suspiciousness formulas, such as "T_1—improved type" formulas and "T_2—invariant type" formulas. However, we could not verify the effectiveness of the "T_3—uncertain type" formulas in Table 3, because the value of $f'(n_{ep})$ depends on the number of passed test cases and further discuss the inter-partition of n_{ep}.
(3) The number of lines of code for the four programs, including gzip, grep, sed and flex, is from 5000 to 10,000. We apply the whole test suite as the input to individual subject programs. They have been adopted to evaluate fault localization techniques in previous work [1,29]. Some of them, which are relatively large, are real-world programs and have real-life scales [30]. This allows us to integrate our approach into a larger-scale code analysis tool.

3.3.4. Summary

Overall, using the proposed framework, we have theoretically proved 30 suspiciousness formulas after removing coincidental correctness test suites. We can find that 23 of these 30 formulas are improved formulas, five of them are unaffected formulas, and the remaining two of them are uncertain. We conducted the experiments with four C open-source programs to evaluate coincidental correctness on the effectiveness of SBFL (i.e. Tarantula, Ochiai, Russel&Rao, Naish2, Jaccard, Naish1, Wong1). The experimental results show that removing coincidental correctness test cases can significantly enhance the effectiveness of "T_1—improved Type" formulas; Tarantula, Ochiai, Russel&Rao, Naish2, Jaccard are "T_1—improved Type" formulas through empirical study. Whereas, removing coincidental correctness test cases sometimes does not improve the effectiveness of "T_2—invariant Type" formulas. Both Naish1 and both Wong1 are "T_2—invariant Type" formulas through empirical study.

4. Conclusions

With the prevalence of coincidental correctness, it is important to know how suspiciousness formulas are affected by coincidental correctness. In this paper, we proposed a functional framework to prove that coincidental correctness impacts SBFL techniques. Through theoretical analysis, we demonstrated that removing coincidental correctness degrades the effectiveness of SBFL in most cases, but sometimes, it does not affect the effectiveness of some SBFL formulas. Considering coincidental correctness, we evaluated the effectiveness of the 30 SBFL formulas by theoretical analyses. Our approach classified the 30 investigated SBFL formulas into three categories: removing coincidental correctness improved the effectiveness of SBFL, which is named "T_1—improved type" formulas; removing coincidental correctness did not affect on the effectiveness of SBFL, which is named'T_2—invariant type" formulas; and the effect on the effectiveness of SBFL by removing coincidental correctness depends on certain conditions, which are named "T_3—uncertain type" formulas. We conducted an empirical study on four C programs and found that the effectiveness of some suspiciousness formulas can be enhanced; however, the effectiveness of some other suspiciousness formulas can be unchanged. Generally speaking, our empirical study is useful to highlight the interesting phenomena, which can suppose the generalization needed to be tested and verified by theoretical proof. This study indicates that theoretical proof and empirical study are essential and complementary in software engineering. This provides a theoretical basis for removing coincidental correctness test cases for locating faults in the field of software engineering.

In the future, we want to deeply solve the interesting problem highlighted by this study: to compare the effectiveness of more formulas under the factors influencing coincidental correctness. This has motivated us to adopt a theoretical approach to study some other factors influencing the effectiveness of fault localization. Finally, we want to apply our approach to more subject programs written in other languages and perform more detailed empirical studies.

Author Contributions: Introduction, H.C. and L.L.; Background, L.L. and Y.S.; Our Framework, H.C.; L.L. and Y.S.; Proof, H.C.; Empirical study, H.C. and Y.S.; Writing and editing, H.C. and L.L.;All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Cultivation Programme for Young Backbone Teachers in Henan University of Technology, Key scientific research project of colleges and universities in Henan Province (No.22A520024), Major Public Welfare Project of Henan Province (No. 201300311200), National Natural Science Foundation of China (Nos. 61602154, 61340037), and Natural Science Project of the Henan Science and Technology Department under Grant (No.212102210148).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We would like to show the acknowledgments to the Cultivation Programme for Young Backbone Teachers in Henan University of Technology, Key scientific research project of colleges and universities in Henan Province (No.22A520024), Major Public Welfare Project of Henan Province (No. 201300311200), National Natural Science Foundation of China (Nos. 61602154, 61340037), and Natural Science Project of the Henan Science and Technology Department under Grant (No.212102210148).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Vessey, I. Expertise in debugging computer programs: A process analysis. *Int. J. Man-Mach. Stud.* **1985**, *23*, 459–494.
2. Voas, J.M. PIE: A Dynamic Failure-based Technique. *IEEE Trans. Softw. Eng.* **1992**, *18*, 717–727.
3. Masri, W.; Abou-Assi, R.; El-Ghali, M.; Fatairi, N. An empirical study of the factors that reduce the effectiveness of coverage-based fault localization. In Proceeedings of the 2nd International Workshop on Defects in Large Software Systems, Chicago, IL, USA, 19 July 2009; ACM; pp. 1–5.

4. Hierons, R.M. Avoiding Coincidental Correctness in Boundary Value Analysis. *ACM Trans. Softw. Eng. Methodol.* **2006**, *15*, 227–241.
5. Wang, X.; Cheung, S.C.; Chan, W.K.; Zhang, Z.; Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. In Proceedings of the 31th International Conference on Software Engineering, Vancouver, BC, Canada, 16–24 May 2009; IEEE; pp. 45–55.
6. Feyzi, F.; Parsa, S. A program slicing-based method for effective detection of coincidentally correct test cases. *Computing* **2018**, *2*, 1–4.
7. Assi, R.A.; Masri, W.; Trad, C. How detrimental is coincidental correctness to coverage-based fault detection and localization? An empirical study. *Softw. Testing Verif. Reliab*. **2021**, *31*, 113–124.
8. Masri, W.; Podgurski, A. An empirical study of the strength of information flows in programs. In Proceedings of the International Workshop on Dynamic Systems Analysis, Shanghai, China, 20–28 May 2006; ACM; pp. 73–80.
9. Masri, W.; Assi, R.A. Prevalence of Coincidental Correctness and Mitigation of its Impact o n Fault Localization. *ACM Trans. Softw. Eng. Methodol.* **2014**, *23*, 1–28.
10. Feyzi, F. CGT-FL: Using cooperative game theory to effective fault localization in presence of coincidental correctness. *Empir. Softw. Eng*. **2020**, *25*, 3873–3927.
11. Hofer, B. Removing Coincidental Correctness in Spectrum-Based Fault Localization for Circuit and Spreadsheet Debugging. In Proceedings of the 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Toulouse, France, 23–26 October 2017; IEEE; pp. 23–33.
12. Lee, H.J.; Naish, L.; Ramanohanrao, K. Study of the relationship of bug consistency with respect to performance of spectra metrics. In Proceedingsof the 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, China, 8–11 August 2009; IEEE; pp. 501–508.
13. Nais, L.; Lee, H.J.; Ramamohanarao, K. A model for spectra-based software diagnosis. *ACM Trans. Softw. Engining Methodol.* **2011**, *20*, 11–32.
14. Xie, X.Y.; Chen, T.Y.; Kuo, F.C.; Xu, B.W. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Trans. Softw. Eng. Methodol.* **2013**, *22*, 31–70.
15. Golagha, M.; Pretschner, A.; Briand, L.C. Can we predict the quality of spectrum-based fault localization? In Proceedingsof 2020 IEEE 13th International Conference on Software Testing, Validation and Verification, Porto, Portugal, 24–28 October 2020; IEEE; pp. 4–15.
16. Dutta, A.; Kunal, K.; Srivastava, S.S.; Shankar, S.; Mall, R. FTFL: A Fisher's test-based approach for fault localization. *Innov. Syst. Softw. Eng.* **2021**, *17*, 1–25.
17. Ghosh, D.; Singh, J. Spectrum-based multi-fault localization using chaotic genetic algorithm. *Inf. Softw. Technol.* **2021**, *133*, 106512–106524.
18. Zhao, G.; He, H.; Huang, Y. Fault centrality: Boosting spectrum-based fault localization via local influence calculation. *Appl. Intell.* **2022**, *52*, 7113–7135.
19. Wu, Y.H.; Li, Z.; Liu, Y. Chen, X. Fatoc: Bug isolation based multi-fault localization by using optics clustering. *J. Comput. Sci. Technol.* **2020**, *35*, 979–998.
20. Jones, J.A.; Harrold, M.J. Empirical evaluation of the tarantula automatic fault-localization technique. In Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA , 7–11 November 2005; IEEE; pp. 273–282.
21. Jones, J. A.; Harrold, M. J.; Stasko, J. Visualization of test information to assist fault localization. In Proceedings of the 24th International Conference on Software Engineering. Orlando, FL, USA, 25 May 2002; IEEE; pp. 467–477.
22. Chen, M.Y.; Kiciman, E.; Fratkin, E.; Fox, A.; Brewer, E. Pinpoint: Problem determination in large, dynamic internet services. In Proceedings of the International Conference on Dependable Systems and Networks, Goteborg, Sweden, 1–4 July 2002; IEEE; pp. 595–604.
23. Abreu, R.; Zoeteweij, P.; Van Gemund, A.J.C. An evaluation of similarity coefficients for software fault localization. In Proceedingsof the 12th Pacific Rim International Symposium on Dependable Computing, Riverside, CA, USA, 18-20 December 2006; IEEE; pp. 39–46.
24. Masri, W.; Assi, R.A.; Zaraket, F.; Fatairi, N. Enhancing fault localization via multivariate visualization. In Proceedings of the 5th International Conference onin Software Testing, Verification and Validation, Montreal, QC, Canada, 17–21 April 2012; IEEE; pp. 737–741.
25. Rui A.; Zoeteweij, P.; Golsteijn, R.; Van Gemund, A.J. A practical evaluation of spectrum-based fault localization. *J. Syst. Softw.* **2009**, *82*, 1780–1792.
26. Software Infrastructure Repository. Available online: http://sir.unl.edu/php/index.php (accessed on 29 April 2022).
27. Gcc compiler. Available online: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html (accessed on 29 April 2022).
28. Ju, X.; Jiang, S.; Chen, X.; Wang, X.; Zhang, Y.; Cao, H. HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices. *J. Syst. Softw.* **2014**, *90*, 3–17.

29. Zhang, L.; Yan, L.; Zhang, Z.; Zhang, J.; Chan, W.K.; Zheng, Z. A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization. *J. Syst. Softw.* **2017**, *129*, 35–57.
30. Yoo, S.; Xie, X.; Kuo, F.C.; Chen, T.Y.; Harman, M. Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis. *ACM T. Softw. Eng. Meth.* **2017**, *26*, 1–30.