

Normalization of Notation NCF for Improving Fault Localization

Zhao Li^a, Yi Song^a, Siwei Zhou^{b,c}, Dongcheng Li^c, and Peng Chen^{a,*}

^aCollege of Computer and Information Technology, China Three Gorges University, Yichang, 443002, China

^bSchool of Computer Science and Technology, Wuhan University of Technology, Wuhan, 430070, China

^cDepartment of Computer Science, University of Texas at Dallas, Richardson, 75080, USA

Abstract

In view of the importance and the high cost of effective software fault localization, how to improve the effectiveness of the software fault localization has become an important and persistent issue in software engineering. Featuring simple operation and high popularity, the spectrum-based fault localization technique obtains program spectrum information by executing test cases on the program and then calculates the suspiciousness of each statement, which provides a basis for the programmers to debug. This paper proposed *CFNorm*, a new fault localization parameter, which can be obtained by processing the column data of the spectrum information matrix. *CFNorm* emphasizes and amplifies the role of NCF (the number of times a statement is executed by failed test cases) to optimize the traditional fault localization technique for better fault localization. Three fault localization techniques were selected to be used in an experiment involving 111 versions of Siemens Suite. The results showed that the effectiveness of fault localization was significantly improved with the increase in the weight of *CFNorm* over a certain range.

Keywords: *CFNorm*; software fault localization; testing; debugging; suspiciousness; normalization

(Submitted on May 28, 2019; Revised on June 20, 2019; Accepted on July 16, 2019)

© 2019 Totem Publisher, Inc. All rights reserved.

1. Introduction

Software troubleshooting is an effective means of ensuring software quality and improving software reliability, and it plays a vital role in the process of software testing [1]. A programmer can analyze and solve a fault only when it is accurately localized. Therefore, how to improve the effectiveness of fault localization has become a concern for many researchers. The spectrum-based fault localization technique [2], among the many existing fault localization techniques, has been favored by an increasing number of researchers because of its numerous branches and accurate localization. The spectrum-based fault localization technique has four objects, i.e., the program to be tested, *P*; the test cases, *T*; the spectrum information, *Notation*; and the suspiciousness formula, *Coefficient*.

The classified test cases *T* (*t_{successful}* or *t_{failed}*) [3] is executed on *P*, and the spectrum information *Notation* can be obtained while the program is running and then substituted into the *Coefficient* to obtain the suspiciousness of each statement. The higher the value, the larger the possibility of a bug being found in the statement.

The spectrum information mainly contains eight notations [4] (as shown in Table 1), where *N_{CF}* characterizes the number of times a statement is executed by failed test cases. The higher the value, the more the failed test cases covering the statement, and the larger the possibility of the bug being found in the statement [5]. *N_{CF}* is almost equivalent to the numerator in the existing suspiciousness formula, i.e., *N_{CF}* is directly proportional to the possibility of the bug occurring in the statement, and it is highly consistent with the target of fault localization. Therefore, *N_{CF}* should be given more attention compared to other notations in the spectrum information.

Wong et al. proposed DStar [6], a new fault localization technique whose suspiciousness is calculated as $(N_{CF})^*/(N_{UF} + N_{CS})$. *N_{CF}* was separately used as the numerator in the formula, and its weight was adjusted by changing the exponent. The

* Corresponding author.

E-mail address: chenpeng@ctgu.edu.cn

result showed that, in a certain range, the greater the weight of N_{CF} , the better the effectiveness of DStar, hence confirming the positive effect of N_{CF} on fault localization. However, DStar is a specific formula designed to strengthen the role of N_{CF} and has a relatively single application scenario; its role can only be reflected in the DStar formula, and it cannot be used to optimize more spectrum-based fault localization techniques. In addition, for the ease of fault localization, researchers often organize the spectrum information into a coverage matrix of statement [6] (as shown in Table 2), i.e., by incorporating all the spectrum information of a statement into each row [7], and each column contains all the values of a notation in the spectrum information. Previous studies mainly adopted the *Statement-Oriented* perspective, which obtains the suspiciousness by processing the row information, while a few adopted the *Spectrum-Oriented* perspective, which takes the column information of the matrix as the research object.

This paper proposed *CFNorm*, a new fault localization parameter, which is obtained by normalizing the data in the column of N_{CF} in the coverage matrix of statement and then adding 1 to it (to avoid the normalized value of 0). *CFNorm* can be combined with the traditional fault localization formula by calculating the product, and it features good compatibility and portability. We took the Siemens Suite [8] as a dataset and combined the *CFNorm* with Tarantula [5], Ochiai [9], and Crosstab [10]. The results showed that *CFNorm* has significantly improved the original technique.

We have proposed a spectrum-based multi-technique fusion approach (FA) for software fault localization, which fuses several existing techniques to narrow the gap between the performance in the best and worst assumptions (see Section 3.3 for more details). The key to improving FA is to enhance the effectiveness of the sub-techniques used. Therefore, although the existing techniques optimized by *CFNorm* may be inferior to DStar, they can contribute to the improvement of the effectiveness of FA, so it is necessary to extend the advantages of N_{CF} to more techniques.

The major contributions of this paper are summarized as follows:

- The adoption of the *Spectrum-Oriented* perspective in extracting the column information of the statement coverage matrix as the research object;
- The proposal of *CFNorm* in optimizing the three traditional fault localization techniques.

The remainder of this paper is organized as follows: Section 2 introduces the spectrum-based fault localization technique, Section 3 describes the details and implementation method of *CFNorm*, Section 4 conducts case studies and compares the improved technique with the original one, Section 5 discusses the threats to validity, Section 6 describes the related work, and Section 7 presents the conclusion and future plans.

2. The Spectrum-based Fault Localization Technique

2.1. Details of Spectrum Information

The program to be tested, P , and the test cases, T , are the two inputs of the spectrum-based fault localization technique. P is a series of codes whose bugs need to be found, and it consists of k executable statements s_i , where $i = 1, 2, 3, \dots, k$ (comments, blank rows, functions and variable declarations, etc. are non-executable statements). T is a dataset entered into P that consists of m test cases t_j , where $j = 1, 2, 3, \dots, m$. t_j can be divided into successful test case ($t_{successful}$) and failed test case (t_{failed}) groups according to the consistency of the expected output with the actual output. Putting T on P and executing P will produce the spectrum information of each statement, as shown in Table 1.

Table 1. Notations used in this paper

$N_{CF}(\omega)$	Number of failed test cases covering ω
$N_{UF}(\omega)$	Number of failed test cases not covering ω
$N_{CS}(\omega)$	Number of successful test cases covering ω
$N_{US}(\omega)$	Number of successful test cases not covering ω
$N_C(\omega)$	Number of test cases covering ω
$N_U(\omega)$	Number of test cases not covering ω
N_S	Total number of successful test cases
N_F	Total number of failed test cases

2.2. The Coverage Matrix of the Statement

In practice, in order to facilitate the processing of data and make it more intuitive and easier to use, researchers often organize the spectrum information into a coverage matrix of the statements, in which each row characterizes all the

spectrum information of an executable statement and each column characterizes all values of a notation in the spectrum information, as shown in Table 2.

Table 2. The coverage matrix of a statement

	N_{CF}	N_{UF}	N_{CS}	N_{US}	N_C	N_U	N_S	N_F
s_1	•	•	•	•	•	•	•	•
s_2	•	•	•	•	•	•	•	•
s_3	•	•	•	•	•	•	•	•
...
s_k	•	•	•	•	•	•	•	•

For example, the second row in the table is the spectrum information of the statement s_1 . Substituting N_{CF} , N_{UF} , ..., N_F into the formula returns the suspiciousness of s_1 , and the suspiciousness of each executable statement can be obtained in the same way. By sorting the statement suspiciousness in descending order, the programmer will be presented with the most bug-susceptible statement, thus localizing the fault quickly and accurately [11-13].

2.3. The Role of Each Notation in the Spectrum Information

The meanings of the eight notations in the spectrum information are listed in Table 1. The characteristic of each notation needs to be grasped and will be analyzed below (assuming that the following analysis is carried out in a single fault environment, i.e., only one bug in each faulty version).

N_{CF} refers to the number of times a statement is covered by failed test cases. Obviously, a higher N_{CF} implies a higher number of failed test cases executing the statement, which in turn results in a larger possibility of the bug being found in the statement [14]. Therefore, N_{CF} should be directly proportional to the suspiciousness of the statement. An important rule is that ω must be a non-faulty statement [15] if $N_{CF}(\omega) < N_F(\omega)$. This is because one or more test cases do not cover ω but will still fail if the above inequality is true. At this time, ω is a faulty statement and a single fault environment are obviously contradictory. If $N_{CF}(\omega) = N_F(\omega)$, all failed test cases cover ω . At this time, we can only assume that ω may have a bug, because ω may be a general but critical statement. On the contrary, if ω is a faulty statement, $N_{CF}(\omega) = N_F(\omega)$.

N_{UF} refers to the number of times a statement is not covered by failed test cases. Obviously, the higher the N_{UF} , the lower the number of failed test cases executing the statement, and the smaller the possibility of the bug being found in the statement. Therefore, N_{UF} should be inversely proportional to the suspiciousness of the statement.

N_{CS} refers to the number of times a statement is covered by successful test cases. Obviously, the higher the N_{CS} , the higher the number of successful test cases executing the statement, and hence the smaller the possibility of the bug being found in the statement. Therefore, N_{CS} should be inversely proportional to the suspiciousness of the statement. N_{CS} and N_{CF} are mutually-exclusive notations, because the higher the number of successful test cases executing the same statement, the lower the number of failed test cases executing the statement, and vice versa. It should be pointed out that the N_{CS} of a statement has no obvious relationship with its bug.

N_{US} refers to the number of times a statement is not covered by successful test cases. Obviously, the higher the N_{US} , the lower the number of successful test cases executing the statement, and the larger the possibility of the bug being present in the statement. Therefore, N_{US} should be directly proportional to the suspiciousness of the statement.

N_C and N_U respectively refer to the total number of times a statement is covered and uncovered by the test cases ($N_C = N_{CS} + N_{CF}$, $N_U = N_{US} + N_{UF}$). The higher the N_C (the lower the N_U , and vice versa), the higher the frequency of the statement executed in the program. However, N_C and N_U play a supporting role in fault localization, because their only function is to count the statement coverage, and they are not responsible for whether they are executed by a successful or failed test case.

N_S and N_F respectively refer to the number of successful test cases $t_{successful}$ and failed test cases t_{failed} in T ($N_S = N_{CS} + N_{US}$, $N_F = N_{CF} + N_{UF}$). The test cases are classified according to the consistency of the expected output with the actual output, so N_S and N_F are determined by both the test case and the program.

The above analysis shows that N_{CF} is of great importance to fault localization and therefore should have a higher weight than other notations in the spectrum information. DStar, mentioned in Section 1, is proposed on the basis of this theory. How to make N_{CF} improve the effectiveness of more fault localization techniques, rather than being limited to one technique, has become a problem to be solved.

CFNorm, which is proposed in this paper, is a new fault localization parameter that can combine the advantages of N_{CF} with various traditional fault localization techniques to improve the effectiveness of the original technique. Its details are described below.

3. The Details and Evaluation Metrics of *CFNorm*

It should be noted that *CFNorm* is not a separate technique; it is a new parameter obtained by processing N_{CF} that cannot be applied separately and needs to be used in combination with the existing fault localization techniques.

3.1. How to Obtain *CFNorm*

Unlike the *Statement-Oriented* perspective, *CFNorm* is the result of the *Spectrum-Oriented* perspective. In order to take into full account all executable statements in the program and reduce the complexity of the exponential calculations later, the data of the N_{CF} column in the statement coverage matrix (as shown in Table 2) are normalized before converting the irregular values to values between 0 and 1. That is, because N_{CF} represents the number of times a statement is executed by failed test cases, its value can be large, so normalization of the data of the N_{CF} column is necessary for reducing the complexity of exponential operations and avoiding excessively large *CFNorm* values to show the effect of other notations. In addition, *CFNorm* is used as a factor in the existing suspiciousness formula, so it should not be 0. For this reason, we add 1 to the normalized values to get a set of values between 1 and 2.

The steps to calculate *CFNorm* of a statement ω in the program P are as follows.

3.1.1. Normalize the Data in the NCF Column

$$NewValue(\omega) = \frac{OldValue(\omega) - min}{max - min} \quad (1)$$

Where $OldValue(\omega)$ and $NewValue(\omega)$ are the values before and after the N_{CF} normalization of the statement ω , respectively, while max and min are the maximum and minimum values of the N_{CF} column, respectively.

3.1.2. Add 1 to the Normalized Value

$$CFNorm(\omega) = newValue(\omega) + 1 \quad (2)$$

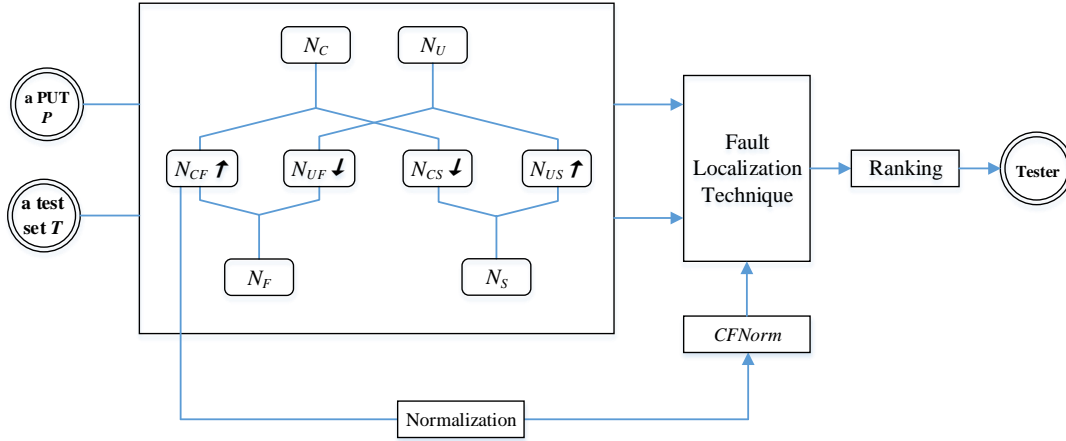
3.2. How to Use *CFNorm*

Through the steps in Section 3.1, we can obtain the *CFNorm* of each executable statement in the program P and then apply them to the existing fault localization technique. Figure 1 shows the flow chart of *CFNorm*, from which we can see that in the previous fault localization flow, the spectrum information is the only object processed in the fault localization formula. In our technique, *CFNorm* is extracted from the spectrum information and becomes another input in the formula. The arrows in rectangles N_{CF} , N_{UF} , N_{CS} , and N_{US} show the relationships between the corresponding notation and suspiciousness. The upward arrow indicates that the suspiciousness of statements increases as the value of the corresponding notation increases; on the contrary, the downward arrow indicates that the suspiciousness of statements decreases as the value of the corresponding notation increases.

The method of combining *CFNorm* with the existing fault localization techniques is as follows:

$$AF' = AF \times CFNorm^* \quad (3)$$

Where AF is the algebraic form of the existing fault localization technique, $CFNorm^*$ is the result of the exponentiation of *CFNorm* (with $*$ being the exponent), and AF' is the improved fault localization technique. Obviously, when $*$ = 0, $AF' = AF$; the larger the value of $*$, the greater the weight of *CFNorm*, and the greater the extent of its change to AF . The suspiciousness of each statement is generated with the improved fault localization technique and submitted to the programmer after being sorted in descending order, providing a basis for fault localization. If the ranking of suspiciousness generated by AF' can help programmers localize bugs more effectively compared to the one generated by AF , AF' has a better effectiveness than AF . We adopt two mainstream metrics to evaluate the technique before and after the improvement.

Figure 1. Flow chart of *CFNorm*

3.3. How to Evaluate *CFNorm*

The EXAM score and *NSE* (Number of Statement Examined) are widely used by researchers as two mainstream evaluation metrics for fault localization techniques. In this paper, we use the improved two metrics for evaluation. When the statement with bug is assigned with the same suspiciousness as other statements, the statements with the same suspiciousness will form a *Tie*. Two different cases will occur: if the statement with the bug is the first to be examined in the *Tie*, it is called the *best case*; if the statement with the bug is the last to be examined in the *Tie*, it is called the *worst case*. In different cases, the evaluation metric will provide different judgments.

3.3.1. EXAM Score

EXAM score [16-17] refers to the percentage of examined statements in all the executable statements when a bug has been found by examining the statements from top to bottom according to the suspiciousness ranking generated. As multiple faulty versions of the program *P* are required to be tested in this paper, the EXAM score was improved to the median EXAM score (*MEE*) and the average EXAM score (*AVE*) in order to make the results more intuitive. *MEE* is the median of EXAM scores, and *AVE*'s formula is as follows:

$$AVE = \frac{\sum_{i=1}^v EXAM_i}{v} \quad (4)$$

3.3.2. *CNSE* (Cumulative Number of Statements Examined)

Similar to the EXAM score, the *CNSE* also represents the cost for finding a bug, but in a different form. *CNSE* refers to the cumulative number of statements examined when all versions find bugs by examining the statements from top to bottom according to the suspiciousness ranking generated [18-19]. Its formula is as follows:

$$CNSE = \sum_{i=1}^v N_i \quad (5)$$

Where N_i is the number of statements to be examined when finding a bug on the i^{th} faulty version and v is the number of faulty versions. If $CNSE(AF') < CNSE(AF)$, the evaluation metric will evaluate AF' as having a better effectiveness than AF .

4. Case Study

Three existing fault localization techniques were selected to conduct the experiment on Siemens Suite, and they are described as follows:

4.1. Fault Localization Technique

We selected Tarantula, Ochiai, and Crosstab to conduct the experiment for the following reasons [19]: Tarantula is simple

and popular, while Ochiai and Crosstab have been proven to perform better than Tarantula. They also have different mathematical forms.

4.1.1. Tarantula

Tarantula assumes that the higher the number of failed test cases covering a statement, the higher the possibility that the statement contains a bug. Tarantula assigns a suspiciousness to each statement according to Equation (6):

$$sups = \frac{\frac{N_{CF}}{N_F}}{\frac{N_{CF}}{N_F} + \frac{N_{CS}}{N_S}} \quad (6)$$

4.1.2. Ochiai

Ochiai attaches great importance to the role of N_{CF} and places it alone in the numerator position. N_{CF} is directly proportional to the suspiciousness assigned to the statement by Ochiai. The formula of Ochiai is shown in Equation (7):

$$sups = \frac{N_{CF}}{\sqrt{N_F N_C}} \quad (7)$$

4.1.3. Crosstab

Crosstab is a statistical localization method that creates a cross tabulation for each executable statement with the spectrum information to obtain the chi-square statistics and contingency correlation coefficient [14]. Its formula is shown in Equation (8):

$$sups = \frac{(N_{CF} - E_{CF})^2}{E_{CF}} + \frac{(N_{CS} - E_{CS})^2}{E_{CS}} + \frac{(N_{UF} - E_{UF})^2}{E_{UF}} + \frac{(N_{US} - E_{US})^2}{E_{US}} \quad (8)$$

For details of the formula, please refer to [10].

4.2. The Siemens Suite Programs

Siemens Suite has been widely applied in fault localization studies [20-21]. It consists of seven C language programs and contains 132 faulty versions, as shown in Table 3, which can be downloaded in SIR (Software-artifact Infrastructure Repository).

Table 3. Summary of Siemens Suite

Program	Num. faulty versions	Num. statements	Num. test cases
<i>print_tokens</i>	7	565	4130
<i>print_tokens2</i>	10	510	4115
<i>replace</i>	32	563	5542
<i>schedule</i>	9	412	2650
<i>schedule2</i>	10	307	2710
<i>tcas</i>	41	173	1608
<i>tot_info</i>	23	406	1052

Since a bug occurred in the header file, the non-executable row or test cases cannot find the bug, so the following versions have been excluded from our experiment: versions 4 and 6 in *print_tokens*; version 10 in *print_tokens2*; versions 12, 27, and 32 in *replace*; versions 1, 5, 6, and 9 in *schedule*; versions 8 and 9 in *schedule2*; versions 6, 10, 19, and 21 in *tot_info*; and versions 13, 14, 15, 36, and 38 in *tcas*.

4.3. Experiment Environment

A computer equipped with a 2.83-GHz Intel Core 2 Quad CPU and an 8GB RAM was taken as the experiment medium, and

the program was executed on a VMware Workstation Pro 12.0.0 virtual machine that ran Ubuntu-14.04 LTS and gcc (version 4.8.4). The spectrum information was then collected with gcov [22], after which the data were processed with Python.

4.4. Experiment Progress

We combined *CFNorm* with three selected fault localization techniques to obtain *Tarantula'*, *Ochiai'*, and *Crosstab'* using Equations (9) to (11).

$$\text{Tarantula}' : \frac{\frac{N_{CF}}{N_F}}{\frac{N_{CF}}{N_F} + \frac{N_{CS}}{N_S}} \times CFNorm^* \quad (9)$$

$$\text{Ochiai}' : \frac{N_{CF}}{\sqrt{N_F N_C}} \times CFNorm^* \quad (10)$$

$$\text{Crosstab}' : \left(\frac{(N_{CF} - E_{CF})^2}{E_{CF}} + \frac{(N_{CS} - E_{CS})^2}{E_{CS}} + \frac{(N_{UF} - E_{UF})^2}{E_{UF}} + \frac{(N_{US} - E_{US})^2}{E_{US}} \right) \times CFNorm^* \quad (11)$$

The *AVEs* obtained by localizing faults on 111 versions of Siemens Suite with the above three improved techniques are shown in Figure 2.

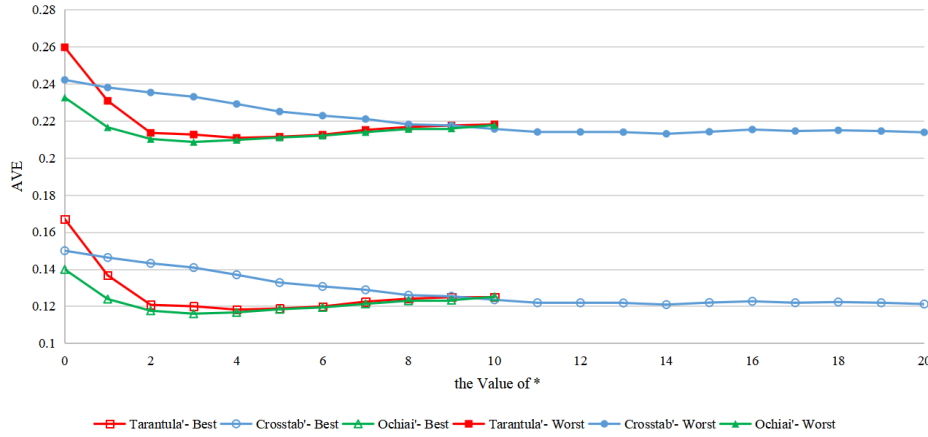


Figure 2. The *AVEs* of the three improved techniques

It can be seen that the *AVEs* of *Tarantula'* (*Tarantula'* at $*$ = 0) in the *best* and *worst cases* are 16.70% and 25.97%, respectively. Also, the *AVE* of *Tarantula'* continuously decreases as the $*$ value increases. *Tarantula'* has the best effectiveness when $*$ = 4, during which its *AVEs* in the both cases are 11.81% and 21.08%, respectively, which are 4.89% lower than the original technique. After $*$ exceeds 4, the *AVEs* in both cases start to increase, so the experiment was stopped when $*$ = 10.

The *AVEs* of *Ochiai'* (*Ochiai'* at $*$ = 0) in the *best* and *worst cases* are 13.98% and 23.26%, respectively. The *AVE* of *Ochiai'* continuously decreases as the $*$ value increases. *Ochiai'* has the best effectiveness when $*$ = 3, during which the *AVEs* in the both cases are 11.59% and 20.86%, which are 2.39% and 2.4% lower than the original technique, respectively. After $*$ exceeds 3, the *AVEs* in both cases start to increase, so the experiment was stopped when $*$ = 10.

The *AVEs* of *Crosstab'* (*Crosstab'* at $*$ = 0) in the *best* and *worst cases* are 14.99% and 24.21%, respectively. The effectiveness of *Crosstab'* continuously improves as the $*$ value increases, so the experiment scale was expanded. The results showed that the *AVEs* of *Crosstab'* in both cases continues to decrease when $*$ is between 1 and 14 (the effectiveness is the same when $*$ is 11 and 12). *Crosstab'* has the best effectiveness when $*$ = 14, and its *AVEs* in both cases are 12.08% and 21.30%, which are 2.91% lower than the original technique. However, when $*$ exceeds 14, the effectiveness of *Crosstab'* starts to fluctuate, while its *AVE* gradually stabilizes. In light of this pattern, the experiment was stopped at $*$ = 20.

The three improved techniques were also evaluated with *MEE*, the result of which is shown in Table 4.

Table 4. The medians of the three improved techniques

Tarantula'			Ochiai'			Crosstab'					
*	Best Case	Worst Case	*	Best Case	Worst Case	*	Best Case	Worst Case	*	Best Case	Worst Case
0	10.53%	17.43%	0	6.56%	10.77%	0	7.69%	10.77%	11	4.10%	8.96%
1	5.50%	8.96%	1	4.92%	8.96%	1	6.56%	10.77%	12	4.10%	8.96%
2	4.10%	8.96%	2	4.69%	8.96%	2	5.50%	8.96%	13	4.10%	8.96%
3	4.10%	8.96%	3	4.10%	8.96%	3	5.50%	8.96%	14	4.10%	9.50%
4	4.10%	8.96%	4	4.10%	8.96%	4	5.33%	8.96%	15	4.10%	9.00%
5	4.10%	9.23%	5	4.10%	8.96%	5	4.92%	8.96%	16	4.10%	9.00%
6	4.10%	9.23%	6	4.10%	10.77%	6	4.92%	8.96%	17	4.10%	8.96%
7	4.10%	10.77%	7	4.10%	10.77%	7	4.69%	8.96%	18	4.10%	8.96%
8	4.10%	10.77%	8	4.10%	10.77%	8	4.69%	8.96%	19	4.10%	8.96%
9	4.10%	10.77%	9	4.10%	10.77%	9	4.69%	8.96%	20	4.10%	8.96%
10	4.10%	11.00%	10	4.10%	10.77%	10	4.10%	8.96%			

When $*$ = 2, 3, and 4, *MEEs* of Tarantula' in the *best* and *worst cases* are 4.10% and 8.96%, respectively, which are 6.43% and 8.47% lower than Tarantula; when $*$ = 3, 4, and 5, *MEEs* of Ochiai' in the *best* and *worst cases* are 4.10% and 8.96%, respectively, which are 2.46% and 1.81% lower than Ochiai; when $*$ are 10, 11, 12, 13, 17, 18, 19, and 20, the *MEEs* of Crosstab' in the *best* and *worst cases* are 4.10% and 8.96%, respectively, which are 3.59% and 1.81% lower than Crosstab.

The *CNSEs* are also shown in Table 5. It can be seen that the improved techniques have better fault localization effectiveness than the original one.

Table 5. The *CNSEs* of the three improved techniques

Tarantula'			Ochiai'			Crosstab'					
*	Best Case	Worst Case	*	Best Case	Worst Case	*	Best Case	Worst Case	*	Best Case	Worst Case
0	2370	3240	0	1886	2756	0	2027	2893	11	1573	2436
1	1832	2722	1	1626	2496	1	1960	2813	12	1576	2439
2	1579	2449	2	1529	2399	2	1901	2764	13	1573	2436
3	1614	2484	3	1503	2373	3	1861	2724	14	1578	2441
4	1597	2467	4	1545	2425	4	1800	2663	15	1599	2462
5	1619	2489	5	1601	2471	5	1741	2604	16	1613	2483
6	1651	2521	6	1632	2502	6	1702	2565	17	1611	2481
7	1733	2603	7	1690	2560	7	1675	2538	18	1623	2493
8	1779	2649	8	1743	2613	8	1637	2500	19	1622	2492
9	1807	2677	9	1752	2622	9	1627	2490	20	1610	2480
10	1822	2695	10	1814	2684	10	1602	2465			

When $*$ = 2, *CNSEs* of Tarantula' in the *best* and *worst cases* are 1579 and 2449, respectively, which are 791 lower than Tarantula; when $*$ = 3, *CNSEs* of Ochiai' in the *best* and *worst cases* are 1503 and 2373, respectively, which are 383 lower than Ochiai; when $*$ is 11 and 13, the *CNSEs* of Crosstab' in the *best* and *worst cases* are 1573 and 2436, respectively, which are 454 and 457 lower than Crosstab.

It should be noted that *AVE*, *MEE*, and *CNSE* do not always show the same results, but all of them are effective metrics for evaluating the fault localization technique. *AVE* reflects the average of EXAMs, while *MEE* reflects the intermediate of them. In addition, when the *AVE* of Technique A is less than that of Technique B, the latter may have a smaller *CNSE* than the former. This is because the *AVE* considers the percentage of the statements to be examined in all the executable statements rather than the absolute number of statements. On the contrary, *CNSE* only calculates the total number of statements to be examined rather than its proportion. In practical applications, evaluation metrics should be chosen according to specific needs so that the judgments can be made more flexibly.

5. Threats to Validity

The validity of *CFNorm* may be affected by various factors, including, but not limited to, the following:

5.1. The Selection of the *

The experiment results showed that the effectiveness of the improved fault localization technique does not continuously improve with the increase of the *, but it starts to rebound or gradually stabilize when * reaches a certain value. N_{CF} is the most effective representation of the possibility that a statement contains faults, so it is important for improving the effectiveness of fault localization techniques. Nevertheless, we should not ignore the role played by other notations. Simply increasing the exponent of N_{CF} will over-emphasize the role it plays and weakens the impact of other notations, which is not conducive to fault localization. This entails the appropriate selection of the *, or else the best result cannot be obtained.

5.2. The Programmers' Identification of the Bug

Our experiment assumes that the programmers can correctly judge the bug, that is, they can identify the bug in the statement and not judge the correct statement as a bug. If this assumption is not true, the effectiveness of the fault localization will also be affected.

5.3. The Combination of $CFNorm$ and Traditional Fault Localization Techniques

Section 3.2 describes the process of combination. We take the traditional fault localization technique as a whole and multiply it with $CFNorm$ of different exponents to improve it. This means that the value of the whole new formula will also be 0 if $CFNorm$ is 0 (although we avoid this case by adding the normalized value to 1, and the above assumption still indicates that this combination may threaten the effectiveness of the improved fault localization technique).

6. Related Work

Analogous to fault localization being the focus of research in software testing, the spectrum-based fault localization technique is a popular topic of discussion in this field. Reps et al. [23] pointed out that the spectrum information can be used for fault localization, which has been applied and confirmed in several papers [5-6, 8-10].

N_{CF} is one of the spectrum information, and its special role in fault localization has caught the attention of many researchers. Jones et al. [11] first proposed the spectrum-based software fault localization technique. They pointed out that the larger the proportion of failed test cases cover a statement in total test cases, the larger the possibility of the bug being present in the statement. Wong et al. [6] proposed DStar based on the high consistency of N_{CF} and fault localization target. Reference [14] explained the meaning of each notation in the spectrum information and analyzed the role of N_{CF} . Furthermore, it has been proven that N_{CS} is inversely proportional to the suspiciousness of the statement, which indirectly confirms the role of N_{CF} . It can be seen that understanding and strengthening the advantages of N_{CF} will be the future research direction in spectrum-based software fault localization.

Xia et al. [24] found that fault localization techniques can help professionals reduce the debugging time, and these improvements can be represented as both statistical and substantive significance. Lastly, in order to study how to improve fault localization for the benefit of practitioners, Wong et al. [25] and Kochhar et al. [26] highlighted some directions by conducting a literature review.

7. Conclusions

This paper proposes $CFNorm$, a new fault localization parameter. It assumes that the higher the number of failed test cases covering a statement, the larger the possibility that the statement contains a bug. $CFNorm$ is obtained by adding 1 to the normalized data in the N_{CF} column of the spectrum information. Traditional fault localization techniques can be improved by combining them with $CFNorm$ of a certain exponent, thus improving the effectiveness of fault localization. The experiment of three traditional fault localization techniques on 111 faulty versions of Siemens Suite showed that the effectiveness of the original fault localization techniques are significantly improved after they have been improved by $CFNorm$.

Our future work plan includes testing $CFNorm$ with more datasets in a multiple-bug environment to verify its extendibility. In addition, we also intend to optimize the steps to obtain the $CFNorm$ through various methods, such as by considering other notations, to further enhance the effectiveness of traditional fault localization techniques.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No. 2016YFC0802500,

2016YFB0800403), the Hubei Provincial Natural Science Foundation of China (No. 2018CFC852), and the Research Fund for Excellent Dissertation of China Three Gorges University (No. 2019SSPY072).

References

1. A. Bandyopadhyay, "Improving Spectrum-based Fault Localization using Proximity-based Weighting of Test Cases," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pp. 660-664, 2011
2. R. Abreu, P. Zoetewij, and A. J. C. Van Gemund, "Spectrum-based Multiple Fault Localization," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 88-99, 2009
3. X. Y. Xie, T. Y. Chen, F. C. Kuo, and B. W. Xu, "A Theoretical Analysis of the Risk Evaluation Formulas for Spectrum-based Fault Localization," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 22, No. 4, 2013
4. R. Z. Gao, W. E. Wong, Z. Y. Chen, and Y. B. Wang, "Effective Software Fault Localization using Predicted Execution Results," *Software Quality Journal*, Vol. 25, No. 1, pp. 131-169, 2017
5. J. A. Jones and M. J. Harrold, "Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pp. 273-282, 2005
6. W. E. Wong, V. Debroy, R. Z. Gao, and Y. H. Li, "The DStar Method for Effective Software Fault Localization," *IEEE Transactions on Reliability*, Vol. 63, No. 1, pp. 290-308, 2013
7. D. Hao, L. Zhang, Y. Pan, H. Mei, and J. S. Sun, "On Similarity-Awareness in Testing-based Fault Localization," *Automated Software Engineering*, Vol. 15, No. 2, pp. 207-249, 2008
8. M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the Effectiveness of Dataflow-and Control-Flow-based Test Adequacy Criteria," in *Proceedings of 16th International Conference on Software Engineering*, pp. 191-200, 1994
9. R. Abreu, P. Zoetewij, R. Golsteijn, and A. J. C. van Gemund, "A Practical Evaluation of Spectrum-based Fault Localization," *Journal of Systems and Software*, Vol. 82, No.11, pp. 1780-1792, 2009
10. W. E. Wong, V. Debroy, and D. Xu, "Towards Better Fault Localization: A Crosstab-based Statistical Approach," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 42, No. 3, pp. 378-396, 2011
11. J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of Test Information to Assist Fault Localization," in *Proceedings of the 24th International Conference on Software Engineering*, pp. 467-477, 2002
12. R. Abreu, P. Zoetewij, and A. J. C. van Gemund, "On the Accuracy of Spectrum-based Fault Localization," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION*, pp. 89-98, 2007
13. L. Naish, H. J. Lee, and K. Ramamohanarao, "A Model for Spectra-based Software Diagnosis," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 20, No. 3, pp. 11, 2011
14. S. K. Sahoo, J. Criswell, C. Geigle, and V. Adve, "Using Likely Invariants for Automated Software Fault Localization," *ACM SIGARCH Computer Architecture News*, Vol. 41, No. 1, pp. 139-152, 2013
15. Y. Q. Huang, J. H. Wu, Y. Feng, Z. Y. Chen, and Z. H. Zhao, "An Empirical Study on Clustering for Isolating Bugs in Fault Localization," in *Proceedings of 2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2013
16. M. Renieres and S. P. Reiss, "Fault Localization with Nearest Neighbor Queries," in *Proceedings of 18th IEEE International Conference on Automated Software Engineering*, 2003
17. W. E. Wong, V. Debroy, R. Golden, X. F. Xu, and B. Thuraisingham, "Effective Software Fault Localization using an RBF Neural Network," *IEEE Transactions on Reliability*, Vol. 61, No. 1, pp. 149-169, 2011
18. R. Z. Gao and W. E. Wong, "MSeer—An Advanced Technique for Locating Multiple Bugs in Parallel," *IEEE Transactions on Software Engineering*, Vol. 45, No. 3, pp. 301-318, 2017
19. X. Xu, V. Debroy, W. E. Wong, and D. H. Guo, "Ties within Fault Localization Rankings: Exposing and Addressing the Problem," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 21, No. 6, pp. 803-827, 2011
20. H. Cleve and A. Zeller, "Locating Causes of Program Failures," in *Proceedings of the 27th International Conference on Software Engineering*, pp. 342-351, 2005
21. C. Liu, L. Fei, X. F. Yan, J. W. Han, and S. P. Midkiff, "Statistical Debugging: A Hypothesis Testing-based Approach," *IEEE Transactions on Software Engineering*, Vol. 32, No. 10, pp. 831-848, 2006
22. Gcov, "A Test Coverage Program," (<https://gcc.gnu.org/onlinedocs/gcc/Gcov.html>, accessed September 2016)
23. T. Reps, T. Ball, M. Das, and J. Larus, "The Use of Program Profiling for Software Maintenance with Applications to the Year 2000 Problem," in *Proceedings of ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp. 432-449, 1997
24. X. Xia, L. Bao, D. Lo, and S. P. Li, "Automated Debugging Considered Harmful" Considered Harmful: A User Study Revisiting the Usefulness of Spectra-based Fault Localization Techniques with Professionals using Real Bugs from Large Systems," in *Proceedings of 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016
25. W. E. Wong, R. Z. Gao, Y. H. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Transactions on Software Engineering*, Vol. 42, No. 8, pp. 707-740, 2016
26. P. S. Kochhar, X. Xia, D. Lo, and S. P. Li, "Practitioners' Expectations on Automated Fault Localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pp. 165-176, 2016