Research Article

An Effective Strategy to Build Up a Balanced Test Suite for Spectrum-Based Fault Localization

Ning Li,¹ Rui Wang,² Yu-li Tian,¹ and Wei Zheng³

¹*School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China*

²*Beijing Institute of Information and Control, Beijing 100048, China*

³*School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an 710072, China*

Correspondence should be addressed to Ning Li; lining@nwpu.edu.cn

Received 25 December 2015; Revised 7 March 2016; Accepted 23 March 2016

Academic Editor: Wen Chen

Copyright © 2016 Ning Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

During past decades, many automated software faults diagnosis techniques including Spectrum-Based Fault Localization (SBFL) have been proposed to improve the efficiency of software debugging activity. In the field of SBFL, suspiciousness calculation is closely related to the number of failed and passed test cases. Studies have shown that the ratio of the number of failed and passed test case has more significant impact on the accuracy of SBFL than the total number of test cases, and a balanced test suite is more beneficial to improving the accuracy of SBFL. Based on theoretical analysis, we proposed an PNF (Passed test cases, Not execute Faulty statement) strategy to reduce test suite and build up a more balanced one for SBFL, which can be used in regression testing. We evaluated the strategy making experiments using the Siemens program and Space program. Experiments indicated that our PNF strategy can be used to construct a new test suite effectively. Compared with the original test suite, the new one has smaller size (average 90% test case was reduced in experiments) and more balanced ratio of failed test cases to passed test cases, while it has the same statement coverage and fault localization accuracy.

1. Introduction

Software fault localization is an activity of identifying the exact locations of program faults. It is one of the most tedious and time-consuming tasks in program debugging [1]. To decrease the cost of software debugging, many automatic software fault localization techniques have been proposed in recent years [2–8]. Among those, Spectrum-Based Fault Localization (SBFL) techniques have attracted a lot of attention, such as Tarantula [9] and Ochiai [10]. SBFL techniques are simple and highly efficient, and they usually calculate the suspiciousness of program entities according to different dynamic behaviours of failed and passed test executions.

In the field of SBFL, more than 30 kinds of formulas have been proposed to calculate suspiciousness of program entity [11]. And it is essential to count the number of failed and passed test cases in each SBFL technique. Due to that, we can speculate that the ratio of failed test cases to passed test cases might affect the accuracy of software fault location. In this paper, we call the ratio of the number of failed test cases to

the number of passed test cases as class ratio. Test cases reduction is an important topic in respect of the test case number. Some studies [12–15] focused on test cases reduction technologies for SBFL, but they primarily reduced the total number of test cases, without considering class ratio.

Only a few studies focused on the impact of class ratio on the accuracy of software fault localization [16–18]. The experiments in Gong et al.'s work [16] showed that the class imbalance phenomenon of test suites would negatively affect the efficiency of SBFL. They used two methods to change the class ratio of failed to passed test cases. (1) The first method fixed the total size of test suites and changed the ratio of failed to passed test cases. (2) The second method fixed the number of failed cases and changed the number of passed test cases. They selected some test cases from an original test suite randomly when generating a new test suite. Based on the work, Gao et al. [17] conducted a theoretical study to generate balanced test suite. They cloned the failed test cases for suitable number of times to catch up with the number of

passed test cases. Their theoretical analysis result suggested that the efficiency of SBFL can be improved under certain conditions and impaired at no time by using their strategy.

In this paper, we proposed a nonrandom PNF (Passed test cases, Not execute Faulty statement) test case selection strategy to build up a reduced and balanced test suite. This strategy is applicable for some regression testing. For example, part of test cases should be selected from the original test suite when software platform is updated. Different from cloning failed test cases in literature [17], we prefer to select passed test cases from the original test suite to construct a new balanced test suite. This strategy will not only make the test suite more balanced than before, but also significantly reduce the size of the original test suite without decreasing statement coverage.

This paper is structured as follows. In Section 2, we analyse the PNF strategy from a theoretical perspective. Then the experiments on Siemens and Space are presented in Section 3. Finally Section 4 concludes the paper and outlines our further work.

2. Theoretical Analysis

In this section, we conduct a theoretical analysis about the impact of increasing passed test cases on the accuracy of SBFL. Here, we take two typical SBFL techniques Tarantula and Nashi2 as representative. Based on the analysis result, we propose PNF strategy to build up a balanced test suite, which can not only reduce the size of test suite but also hold the accuracy of SBFL like the original test suite.

2.1. PNF Strategy. In order to describe our PNF strategy, we define the following symbols at first:

- (i) T_{orig} : the original test suite.
- (ii) T_{init} : the initial test suite which consists of some test cases selected from T_{orig} .
- (iii) T_{new} : the increased test suite in which some test cases are added based on T_{init} .
- (iv) Susp_s : suspiciousness of the statement s .
- (v) P_{orig} : the number of all passed test cases in T_{orig} .
- (vi) F_{orig} : the number of all failed test cases in T_{orig} .
- (vii) P : the number of all passed test cases in T_{init} .
- (viii) F : the number of all failed test cases in T_{init} .
- (ix) N_p^s : the number of passed test cases which execute the statement s in T_{init} .
- (x) N_f^s : the number of failed test cases which execute the statement s in T_{init} .
- (xi) $N_p'^s$: the number of passed test cases which execute the statement s in T_{new} .
- (xii) $N_f'^s$: the number of failed test cases which execute the statement s in T_{new} .

In the fault localization report of SBFL, all statements are often ranked by their suspiciousness in descending order. A smaller Rank indicates a higher likelihood of being faulty

statement. Let statement i represent the faulty statement and let j represent any of the nonfaulty statements. Suppose $\text{Rank}_i > \text{Rank}_j$ in the original test suite; if the inequality of $\text{Rank}_i > \text{Rank}_j$ is not changed after modifying the class ratio of failed to passed test cases, we regard it as a positive change strategy for modifying the class ratio.

In this paper, we use the PNF strategy to modify class ratio and construct the new test suite. In PNF strategy, we hold the failed test cases unchanged and then change the number of passed test cases. The detailed steps are listed as follows:

- (1) Build up an initial test suite T_{init} . Copy all failed test cases from T_{orig} and select part of passed test cases from T_{orig} . Here, the number of selected passed test cases is equal to F_{orig} . It means that the class ratio is 1:1 (failed: passed).
- (2) Build up a new test suite T_{new} in which the class ratio is 1: K (failed: passed). Here, the number of passed test cases is K times of P . Moreover, the increased test cases do not execute the faulty statement i .
- (3) Take the coverage information into consideration. We also calculate the statement coverage because of the importance of the coverage criterion in software testing. We give the priority to the passed test cases which contribute to the statement coverage when selecting a new test case. If the statement coverage of T_{new} is lower than that of T_{orig} , we add some additional passed test cases to ensure this goal (the statement coverage of T_{new} is equal to that of T_{orig}).

Algorithm 1 of PNF is used to select passed test cases and build up a balanced test suite.

To do a detailed theoretical analysis, we describe some common points when applying this strategy to increase passed test cases:

- (1) Since we do not increase any failed test cases, we get $N_f^{ti} = N_f^i = F$, $N_f^{tj} = N_f^j$.
- (2) N_p^{tj} denotes the number of passed test cases that executed the statement j , and its value range is $[N_p^j, N_p^j + (K - 1) \times P]$, where $K > 1$.
- (3) Since the increased passed test cases do not execute the faulty statement i , we get $N_p^{ti} = N_p^i$.
- (4) Because all failed test cases definitely execute the faulty statement i , $N_f^i = F = F_{\text{orig}}$.

In the next sections, we do the detailed theoretical analysis by taking typical Tarantula and Nashi2 as examples.

2.2. Theoretical Analysis in Tarantula. The suspiciousness formula of Tarantula is

$$\text{Susp}_s = \frac{N_f^s / F}{N_f^s / F + N_p^s / P}. \quad (1)$$

We try to calculate the object equation:

$$\text{Susp}_{ti} - \text{Susp}_{tj}, \quad (2)$$

```

Input: initial passed testcases: oldPassedList,
        initial failed testcases: oldFailedList,
        class ratio: ratio
Output: new passed testcases: newPassedList
// (1) find passed test cases which do not
//      execute faulty statements
for each tID (passed test cases ID) in oldPassedList do
    stmtSet: get executed statements set of tID;
    if stmtSet not contain faulty lines then
        add (tID, sizeofstmtSet) into nonFaultMap;
    end
if nonFaultMap is empty then
    add all tID  $\in$  oldPassedList into nonFaultMap;
// (2) find passed test cases which execute as
//      many non-faulty statements as possible
sort nonFaultMap by its value (size of stmtSet);
validPassList = nonFaultMap.Key;
// (3) build up a new test cases
newPasscnt = ratio * oldFailedList.size();
newPassedList = new ArrayList(newPasscnt);
// statement coverage
smSet = get the set of all executed statements;;
while newPassedList.size < newPasscnt do
    ti = a new test case of validPassList;
    smSetTi = get executed statements by ti;
    if one element in smSetTi  $\notin$  smSet then
        add ti to newPassedList;
        add smSetTi to smSet;
    end
if statement coverage of  $T_{\text{new}}$  < that of  $T_{\text{orig}}$  then
    // improve statement coverage
    find out the valuable passed test cases;
    add valuable passed test cases to newPassedList;
// make sure of the required class ratio
while newPassedList.size < newPasscnt do
    search unselected test case from validPassList;
    add it to newPassedList;
end
return newPassedList;

```

ALGORITHM 1: Selected best passed test cases (PNF).

where Susp_{ti} and Susp_{tj} denote the suspiciousness of statement i and statement j after increasing passed test cases. We discuss three cases based on the relation of the suspiciousness of the faulty statement and nonfaulty statement in the initial test suite.

According to (1) and $N_f^i = F$, (2) can be expressed as follows:

$$\begin{aligned}
 &\Rightarrow \frac{1}{1 + N_p^i/KP} - \frac{N_f^j/F}{N_f^j/F + N_p^j/KP} \\
 &\Rightarrow \frac{N_f^j}{F} + \frac{N_p^j}{KP} - \frac{N_f^j}{F} - \frac{N_f^j}{F} \times \frac{N_p^i}{KP}
 \end{aligned}$$

$$\begin{aligned}
 &\Rightarrow \frac{N_p^j}{KP} - \frac{N_f^j}{F} \times \frac{N_p^i}{KP} \\
 &\Rightarrow \frac{1}{KP} \times \left(N_p^j - \frac{N_f^j \times N_p^i}{F} \right).
 \end{aligned} \tag{3}$$

Here, we proof that PNF strategy is positive from three cases, respectively: $\text{Susp}_i > \text{Susp}_j$, $\text{Susp}_i = \text{Susp}_j$, and $\text{Susp}_i < \text{Susp}_j$.

Case 1 ($\text{Susp}_i > \text{Susp}_j$). Because the suspiciousness of statement i is greater than the suspiciousness of statement j , then we can express it as follows:

$$T = \text{Tarantula}(i) - \text{Tarantula}(j) > 0. \tag{4}$$

That is,

$$\frac{N_f^i/F}{N_f^i/F + N_p^i/P} - \frac{N_f^j/F}{N_f^j/F + N_p^j/P} > 0. \tag{5}$$

Since $N_f^i = F$, (5) can be simplified as follows:

$$\begin{aligned}
 &\Rightarrow \frac{1}{1 + N_p^i/P} - \frac{N_f^j/F}{N_f^j/F + N_p^j/P} > 0 \\
 &\Rightarrow \frac{N_p^j}{P} - \frac{N_f^j}{F} \times \frac{N_p^i}{P} > 0 \\
 &\Rightarrow F \times N_p^j - N_f^j \times N_p^i > 0 \\
 &\Rightarrow N_p^i < \frac{N_p^j \times F}{N_f^j}.
 \end{aligned} \tag{6}$$

To ensure that our strategy is positive, we need to have $\text{Susp}_{ti} - \text{Susp}_{tj} > 0$.

According to previous calculation, we know

$$\begin{aligned}
 &\text{Susp}_{ti} - \text{Susp}_{tj} > 0 \Rightarrow \\
 &\frac{1}{KP} \times \left(N_p^j - \frac{N_f^j \times N_p^i}{F} \right) > 0 \\
 &\therefore \frac{1}{KP} > 0, \\
 &\therefore N_p^j - \frac{N_f^j \times N_p^i}{F} > 0 \Rightarrow \\
 &N_p^i - \frac{N_p^j \times F}{N_f^j} < 0.
 \end{aligned} \tag{7}$$

According to the above analysis, this problem can be simplified to the following proof:

The following is given: $N_p^i < (N_p^j \times F)/N_f^j$.

The following is proved: $N_p^{tj} < (N_f^j \times N_p^i)/F$.

Proof. For the faulty statement i , because the increased passed test cases do not execute this statement, $N_p^{ti} = N_p^i$. For any statement j ($j \neq i$), because we increased $(K - 1) \times P$ passed test cases, the value range of N_p^{tj} is

$$[N_p^j, N_p^j + (K - 1) \times P], \quad (8)$$

where N_p^{tj} denotes the number of passed test cases which executed the statement j . Consider

$$\begin{aligned} \because N_p^i &< \frac{N_p^j \times F}{N_f^j} \\ \therefore N_p^i &< \frac{[N_p^j, N_p^j + (k - 1) \times P] \times F}{N_f^j} \\ &= \frac{N_p^{tj} \times F}{N_f^j} \Rightarrow \\ N_p^i - \frac{N_p^{tj} \times F}{N_f^j} &< 0, \\ \therefore N_p^{ti} &= N_p^i \Rightarrow \\ N_p^{ti} - \frac{N_p^{tj} \times F}{N_f^j} &< 0 \Rightarrow \\ \text{Susp}_{ti} &> \text{Susp}_{tj}. \end{aligned} \quad (9)$$

□

Based on the proof, if $\text{Susp}_i > \text{Susp}_j$, we can get $\text{Susp}_{ti} > \text{Susp}_{tj}$ when we use the strategy to select passed test cases. That is to say, for the faulty statement i whose suspiciousness is higher than the suspiciousness of the statement j before increasing passed test cases, its suspiciousness is still higher than the suspiciousness of statement j after increasing passed test cases. It shows that the rank of the faulty statement i will not decrease. Therefore, the strategy of increasing passed test cases is a positive approach to select passed test cases in this condition.

Case 2 ($\text{Susp}_i = \text{Susp}_j$). Similarly with the first case, when we increase passed test cases to K ($K > 1$) times than before, the value range N_p^{tj} of is $[N_p^j, N_p^j + (K - 1) \times P]$:

(i) If $N_p^{tj} = N_p^j$,

$\therefore \text{Susp}_i = \text{Susp}_j \Rightarrow$

$$N_p^i = \frac{N_p^j \times F}{N_f^j} \quad (\because N_p^{ti} = N_p^i) \Rightarrow$$

$$N_p^{ti} = \frac{N_p^{tj} \times F}{N_f^j} \Rightarrow$$

$$\text{Susp}_{ti} = \text{Susp}_{tj}. \quad (10)$$

(ii) If $N_p^{tj} > N_p^j$,

$\therefore \text{Susp}_i > \text{Susp}_j \Rightarrow$

$$N_p^i < \frac{N_p^j \times F}{N_f^j} \quad (\because N_p^{ti} = N_p^i) \Rightarrow \quad (11)$$

$$N_p^{ti} < \frac{N_p^{tj} \times F}{N_f^j} \Rightarrow$$

$$\text{Susp}_{ti} > \text{Susp}_{tj}.$$

Based on the above analysis, if $\text{Susp}_i = \text{Susp}_j$, we can get $\text{Susp}_{ti} \geq \text{Susp}_{tj}$ when we use the strategy to select passed test cases. It implies that the rank of the faulty statement i will not decrease while it maybe increases. When we increase passed test cases, if we select passed test cases which execute as many nonfaulty statements ($N_p^{tj} > N_p^j$) as possible, it will enhance the rank of the faulty statement. The strategy of increasing passed test cases in this way will be a positive approach to enlarge test suite.

Case 3 ($\text{Susp}_i < \text{Susp}_j$). The object equation is the same as previous analysis:

$$\text{Susp}_{ti} - \text{Susp}_{tj} = \frac{1}{KP} \times \left(N_p^{tj} - \frac{N_p^{tj} \times N_p^{ti}}{F} \right). \quad (12)$$

Since $1/KP > 0$, $F > 0$, we only focus on the numerator:

$$\Rightarrow N_p^{tj} \times F - N_p^{ti} \times N_p^{tj}$$

$$\because N_p^{ti} = N_p^i \quad (13)$$

$$\Rightarrow N_p^{tj} \times F - N_p^i \times N_p^{tj}.$$

When $N_p^i > (N_p^j \times F)/N_f^j$ ($\text{Susp}_i < \text{Susp}_j$), because $N_p^{tj} \geq N_p^j$, the result of $(N_p^i \times N_p^{tj} - N_p^{tj} \times F)$ could be one of the three cases: > 0 , $= 0$, and < 0 . But as we can know, the following condition can effectively reduce the negative effects:

$$N_p^{ti} < \frac{N_p^{tj} \times F}{N_f^j} \quad (\text{Susp}_{ti} > \text{Susp}_{tj}). \quad (14)$$

Therefore, we should make the value of N_p^{tj} as large as possible; namely, we should select those passed test cases which execute as many nonfaulty statements as possible.

2.3. *Theoretical Analysis in Nashi2.* The suspiciousness formula of Nashi2 is

$$\text{Susp}_s = N_f^s - \frac{N_p^s}{P+1}. \quad (15)$$

The symbols in (15) are the same with the formula of Tarantula. We use the similar analysis process with Tarantula. Therefore, we still try to calculate the difference between Susp_i of the faulty statement and Susp_j of the statement and expect that the rank of faulty statement in the report of SBFL is not impaired. The difference between Susp_{fi} and Susp_{fj} can be expressed as follows:

$$N_f^i - \frac{N_p^i}{KP+1} - \left(N_f^j - \frac{N_p^j}{KP+1} \right). \quad (16)$$

We discuss three cases based on the relation of Susp_i (suspiciousness of the faulty statement) and Susp_j (suspiciousness of the statement) in the original test suite as Section 2.2.

Case 1 ($\text{Susp}_i > \text{Susp}_j$). According to $\text{Susp}_i > \text{Susp}_j$, we have

$$\begin{aligned} N_f^i - \frac{N_p^i}{P+1} &> \left(N_f^j - \frac{N_p^j}{P+1} \right) \Rightarrow \\ F - \frac{N_p^i}{P+1} &> \left(N_f^j - \frac{N_p^j}{P+1} \right) \Rightarrow \\ N_p^i &< (F - N_f^j)(P+1) + N_p^j. \end{aligned} \quad (17)$$

In order to ensure $\text{Susp}_{fi} > \text{Susp}_{fj}$, the object equation of (16) can be transformed into the following:

$$\begin{aligned} F - \frac{N_p^i}{KP+1} - \left(N_f^j - \frac{N_p^j}{KP+1} \right) &> 0 \Rightarrow \\ N_p^i &< (F - N_f^j)(KP+1) + N_p^j \Rightarrow \\ N_p^i &< (F - N_f^j)(KP+1) + N_p^j. \end{aligned} \quad (18)$$

According to the above analysis, this problem can be simplified to the following proof:

The following is given: $N_p^i < (F - N_f^j)(P+1) + N_p^j$.

The following is proved: $N_p^i < (F - N_f^j)(KP+1) + N_p^j$.

Proof. Consider the following:

$$\because K > 1$$

$$\therefore KP+1 > P+1$$

$$\therefore N_p^i < (F - N_f^j)(P+1) + N_p^j \Rightarrow$$

$$N_p^i < (F - N_f^j)(P+1) + N_p^j \Rightarrow$$

$$N_p^i < (F - N_f^j)(KP+1) + N_p^j$$

$$\therefore N_p^i > N_p^j \Rightarrow$$

$$N_p^i < (F - N_f^j)(KP+1) + N_p^j.$$

(19)

□

This proof shows that the suspiciousness of the faulty statement i is still higher than the suspiciousness of statement j after passed test cases are increased. It means that the rank of the faulty statement does not reduce.

Case 2 ($\text{Susp}_i = \text{Susp}_j$). According to this condition, we have

$$N_p^i = (F - N_f^j)(P+1) + N_p^j. \quad (20)$$

In order to ensure $\text{Susp}_{fi} \geq \text{Susp}_{fj}$, the object equation of (16) can be transformed into the following:

$$F - \frac{N_p^i}{KP+1} - \left(N_f^j - \frac{N_p^j}{KP+1} \right) \geq 0. \quad (21)$$

Proof. Because $KP+1 > 0$, we can only focus on the numerator and simplify it into

$$\begin{aligned} &\Rightarrow (F - N_f^j)(KP+1) + N_p^i - N_p^j \\ &\Rightarrow (F - N_f^j)(KP+1) + N_p^i - N_p^j \\ &\because N_p^i = (F - N_f^j)(P+1) + N_p^j \\ &\Rightarrow (F - N_f^j)(KP+1) + N_p^j - [(F - N_f^j)(P+1) + N_p^j] \\ &\Rightarrow P(K-1)(F - N_f^j) + (N_p^j - N_p^j) \\ &\because K > 1, P > 0, F - N_f^j \geq 0, N_p^j - N_p^j \geq 0 \\ &\therefore P(K-1)(F - N_f^j) + (N_p^j - N_p^j) \geq 0 \\ &\Rightarrow F - \frac{N_p^i}{KP+1} - \left(N_f^j - \frac{N_p^j}{KP+1} \right) \geq 0. \end{aligned} \quad (22)$$

□

The above inequality implies that the suspiciousness of the faulty statement i must be equal to or higher than the suspiciousness of statement j after passed test cases are increased. Moreover, we should make the value of N_p^i as large as possible, and it means that we should select the passed test cases which execute as many nonfaulty statements as possible.

Case 3 ($\text{Susp}_i < \text{Susp}_j$). According to this condition, we have

$$N_p^i > (F - N_f^j)(P+1) + N_p^j. \quad (23)$$

We need to calculate the relation between N_p^{ti} and $(F - N_f^j)(KP + 1) + N_p^{tj}$.

Because $(F - N_f^j)(KP + 1) > (F - N_f^j)(P + 1)$ and $N_p^{tj} > N_p^j$, the relation between N_p^{ti} and $(F - N_f^j)(KP + 1) + N_p^{tj}$ can be one of the three cases: higher, equal, and lower.

When $N_p^{ti} < (F - N_f^j)(KP + 1) + N_p^{tj}$, it shows $\text{Susp}_{ti} > \text{Susp}_{tj}$. This case can effectively reduce the negative effects on the rank of the faulty statement by increasing passed test cases. Therefore, we should make the value of N_p^{tj} as large as possible; namely, we should select those passed test cases which execute as many nonfaulty statements as possible.

2.4. Summary of Strategy. According to the analysis and proof about Tarantula and Nashi2 in Sections 2.2 and 2.3, we draw a conclusion that the accuracy of SBFL would not be affected by the class ratio of failed to passed test cases when we change the class ratio using a nonrandom PNF strategy. Here, PNF strategy means holding the number of failed test cases unchanged and modify the class ratio by increasing passed test cases. When increasing passed test cases, the selected passed test cases are expected not to execute the faulty statement and to execute as many nonfaulty statements as possible. According to the analysis, we can build up a balanced test suite with the PNF strategy. The new test suite has the following advantages:

- (1) A smaller size than the original test suite.
- (2) A more balanced ratio of failed to passed test cases than the original one.
- (3) Keeping the same statement coverage as the original test suite.
- (4) At least keeping the same fault localization accuracy of SBFL as the original test suite.

Despite the knowledge about the location of a fault in PNF strategy, it can be replaced by other information, such as the top N suspicious statements according to suspiciousness calculation. However, in order to get more exact result, the knowledge about the location of a fault is required in this paper. This PNF strategy is not applicable to the regression testing in which target program has to be modified. However, it can be used to construct a new test suite for the regression testing in which target program is unchanged. For example, regression testing is often required for all developed products in a company when OS is updated, package is applied, or platform is changed.

3. Experiment

Because the effectiveness of our PNF strategy has been analysed and proven from a theoretical perspective in Section 2, we took two small programs (tcas and totinfo) from Siemens program suite and one large program (Space) as samples to verify our strategy from an experimental perspective in this section.

TABLE 1: Information of three target programs.

Program	LOC	Faulty versions	Total test cases
tcas	174	41	1608
totinfo	407	23	1052
Space	9127	38	13585

3.1. Experiment Setup. Siemens and Space (<http://sir.unl.edu/portal/index.php>) dataset have been widely used in the research work of fault localization, and the original information of the three target programs is listed in Table 1. All the programs are written in C. The average fault localization accuracy of all available versions for a target program is the final accuracy of the program.

There are several faults in some faulty versions of tcas and Space, and we call these versions multiple faults versions. In experiment, we simplified the interactions and interferences between multiple faults presented in [19] and took one multiple faults version as several single fault versions. For these multiple versions, we supposed that only the most suspicious fault could be localized in every iteration, then fixed it, and entered it into the next iteration to find out another fault. Although this way is not very efficient, it is similar to the fault localization process of real software testing in a certain degree. In addition, we used 20 of 38 versions of Space program in our experiment and excluded other versions in which there were compile errors or no failed test cases.

Symbol R refers to different test case class ratio and symbol V denotes different program versions. In experiment, we generated 7 different class ratios for each program suite, as $R_1, R_2, R_3, R_4, R_5, R_6, R_7 = 1, 2, 4, 8, 16, 32, 64$, where $R_7 = 64$ means that the number of passed cases is 64 times the number of failed cases in test suite. Symbol $E = \langle R_i, V_j \rangle$ ($i = 1, \dots, m; j = 1, \dots, k$) refers to one test, and it means experiment on version V_j with test case class ratio R_i .

To evaluate the influence of class ratio in fault localization activity, we performed 4 basic fault localization techniques, Tarantula, Nashi2, Jaccard, and Ochiai, on each faulty program version with different class ratio. Because of the time and space constraints, our case studies did not use other advanced fault localization techniques, such as RBF, DStar, and others proposed in [4, 6, 20–22].

Since failed test cases are more contributive to fault localization, we generated test case suites by remaining all the failed cases and increasing passed cases in accordance with the different class ratio. When we constructed test suites, we used random strategy and nonrandom PNF strategy, respectively. The random strategy selects passed cases randomly from the original passed test cases to generate a new test suite, while the nonrandom PNF strategy selects passed cases according to Algorithm 1.

3.2. Results Measured by Score. In this section, we used suspiciousness Score to measure the accuracy of fault localization,

FIGURE 1: Score with different class ratio (tcas).

which has been widely used in software fault localization [1, 2, 23]:

$$\text{Score} = 1 - \frac{\text{Rank}}{\text{Total}}. \quad (24)$$

In (24), Total denotes the total number of statements, and Rank denotes the rank of the faulty statement. Rank/Total presents the percentage of code that needs to be examined before the faults are identified. A higher score means higher efficiency of fault localization. When calculating score, we used First-Line strategy to deal with the same suspiciousness situations in which we assigned all statements sharing the same suspiciousness with the first ranking number of them. And we took the average of the score of each faulty version of the same program as the final score of the program. For multiple faults version, we assigned the highest score among several iterations as the score of the version.

Figures 1, 2, and 3 present our experiment results on tcas, totinfo, and Space respectively. In the horizontal axis of these figures, “orig” denotes the original test suite and “1 : N” means that the class ratio of failed test cases to passed test cases is 1 : N.

The three figures show that class ratio has effect on fault localization performance when using random strategy to generate test suite for all four SBFL techniques. For tcas and totinfo, we can get better accuracy of fault localization with lower class ratio, which implies that a balanced test suite is more efficient for SBFL by selecting passed test cases in a random strategy, which is also consistent with the result of literature [16]. But when we use the nonrandom PNF strategy to enlarge passed test cases, the class ratio can do barely nothing to fault localization performance, which conformed with our theoretical analysis. Consequently, whether the class ratio has the effect on the accuracy of SBFL is closely related

to the strategy for generating the new test cases. From the results, it may be observed that the scores of the new test suites constructed by PNF strategy are higher than the scores of original test suite for the three target programs, while the strong point does not always hold when using random strategy to build up a new test suite.

As shown in Figure 3, there is a small difference between Space and tcas/totinfo. Figure 3(b) about Space program indicates that the four classical SBFL techniques could get better fault localization accuracy with a more balanced test suite using random strategy. It is the same with tcas and totinfo, while the fault localization accuracy becomes better when the distribution of test cases becomes more unbalanced using PNF strategy. It is a different trend compared with tcas and totinfo. But no matter using PNF strategy or random strategy, we can still achieve higher score than the original test suite. The experiment evidence of our PNF strategy is effective for building up a relative balanced test suite. In this figure, 1 : 2 is the best class ratio considering both the accuracy of fault localization and the size of test suite.

3.3. Results Measured by NScore. This section evaluates our theoretical analysis with NScore measurement conforming with literature [16]. NScore could be calculated by the following equation:

$$\text{NScore} = \frac{N_{\text{detected}}}{N_{\text{total}}} \times 100. \quad (25)$$

In (25), N_{detected} is the number of faulty program versions in which the accuracy of fault localization is higher than a threshold θ and N_{total} is the total number of program versions.

The experimental results of tcas and totinfo using different threshold θ (0.8/0.9/0.95) are presented in Figures 4 and 5.

FIGURE 2: Score with different class ratio (totinfo).

FIGURE 3: Score with different class ratio (Space).

For tcas program, the most left top point in Figure 4(a) means that the scores in 50% of versions are higher than 0.8. Namely, if the threshold θ of accuracy is 0.8 and the class ratio is 1:1, then 50% of faults can be localized correctly with Tarantula. The following can be observed: (1) The performance of fault localization is more stable using PNF strategy than random strategy to generate test case; (2) higher fault localization accuracy can be achieved using PNF strategy instead of random strategy. The totinfo program has the same tendency with the tcas.

Figure 6 illustrates the experiment result of Space program. Similar to the results in Section 3.2, there is a small difference between the Siemens and Space program. Figures 6(c) and 6(d) indicate the following: No matter using PNF or random strategy for Nashi2, NScore of Space does not change with the variation of class ratio, while NScore has an increased tendency along with the growth of unbalanced class ratio for Tarantula presented in Figures 6(a) and 6(b). Although a thorough analysis about the reason causing the difference has not been conducted yet, the PNF strategy can reduce

(a) PNF (Tarantula)

(b) Random (Tarantula)

(c) PNF (Nashi2)

(d) Random (Nashi2)

FIGURE 4: NScore with different class ratio (tcas).

the original test suite to a more balanced test suite and get better fault localization accuracy.

3.4. Experiment Summary and Discussion. The experiments using Score and NScore imply that class ratio really has influence on the accuracy of faulty localization. From the experiment results, we concluded that we can build up a balanced test suite with PNF strategy and ensure the accuracy of fault localization with the new balanced test suite. Also, the size of new test suite is smaller than original one. In

experiments, the test case reduction rate was over 95% in tcas program, 90% in totinfo program, and 90% in Space program.

In experiment, we also found that the score of different faulty versions has a significant difference with the same SBFL method, while the results of some version have same tendency. The average score will make us ignore the difference. Table 2 is the score of different tcas versions with several SBFL methods, and it is sorted by Tarantula's score in descending order. From this table, we observed that the maximum score of tcas is 0.9859, but the minimum score is only 0.1268. Why are the scores of some versions so different? Nashi2, Jaccard,

(a) PNF (Tarantula)

(b) Random (Tarantula)

(c) PNF (Nashi2)

(d) Random (Nashi2)

FIGURE 5: NScore with different class ratio (totinfo).

and Ochiai have the same problem. It is worth doing more studies from this perspective.

4. Conclusions

Every suspiciousness calculation of SBFL method is closely related to the number of passed and failed test cases. Previous studies have shown a balanced test suite, which means that the class ratio of the number of failed to passed test cases is similar

and is more beneficial to improve the accuracy of SBFL. In this paper, we proposed a PNF strategy to building up a balanced test suite according to the theoretical analysis and evaluated it by experiments using different SBFL methods.

In the PNF strategy, in order to construct a new balanced test suite, we kept the failed test cases unchanged and selected passed test cases from the original test suite according to certain rules: the selected passed test cases should not execute the faulty statement and should execute

(a) PNF (Tarantula)

(b) Random (Tarantula)

(c) PNF (Nash2)

(d) Random (Nash2)

FIGURE 6: NScore with different class ratio (space).

as many nonfaulty statements as possible with the consideration of statement coverage. The experiments indicated the PNF strategy is effective for SBFL. Based on the original test suite, it can generate a new more balanced test suite, which has smaller test suite size (average 90% test cases are reduced), the same accuracy of SBFL, and the same statement coverage.

However, PNF strategy still has some limitations. For example, when there are only few failed test cases in the original test suite, the selected passed test cases by PNF

strategy must be not enough for normal testing. And the process about the multiple faulty program is not enough. The work can be improved in following directions: (1) To reveal the other factors which have the effect on the accuracy of SBFL besides the class ratio; (2) To combine the PNF strategy with test suite reduction techniques. These studies will improve the efficiency of testing and the accuracy of fault localization. In addition, how to build up a balanced test suite without the knowledge about the location of a fault is one of our future work.

TABLE 2: Score of different versions (tcas).

Version	Tarantula	Nashi2	Jaccard	Ochiai
v36	0.9859	0.9859	0.6761	0.9859
v25	0.9718	0.9718	0.5775	0.9718
v39	0.9718	0.9718	0.9577	0.9718
v1	0.9577	0.9577	0.8521	0.9577
v4	0.9577	0.9577	0.2113	0.9577
v37	0.9437	0.9577	0.9859	0.9437
v6	0.8732	0.8732	0.1408	0.8732
v35	0.6761	0.7606	0.2113	0.6901
v28	0.6761	0.7606	0.1408	0.6901
v2	0.6761	0.6901	0.9577	0.6901
v29	0.6620	0.6761	0.6761	0.662
v20	0.5915	0.5915	0.2254	0.5915
v21	0.5915	0.5915	0.5915	0.5915
v22	0.5775	0.5775	0.5915	0.5775
v24	0.5775	0.5775	0.5775	0.5775
v9	0.5775	0.5775	0.1690	0.5775
v14	0.5915	0.5915	0.2113	0.5915
v16	0.2254	0.2254	0.5915	0.2254
v7	0.2254	0.3099	0.8732	0.2254
v12	0.2113	0.3099	0.5775	0.2254
v13	0.2113	0.2676	0.2113	0.2254
v26	0.1831	0.2676	0.9718	0.2113
v8	0.1690	0.169	0.2254	0.169
v5	0.1408	0.3099	0.9577	0.2817
v18	0.1268	0.3099	0.2254	0.2676

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (61402370, 61502392), Fundamental Research Funds for the Central Universities (3102015JSJ0004, 3102014JSJ0013), and Aerospace Science and Technology Innovation Foundation of China (2014H03FK011).

References

- [1] 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 (ASE '05)*, pp. 273–282, ACM, November 2005.
- [2] T. Y. Chen, X. Xie, F.-C. Kuo, and B. Xu, "A revisit of a theoretical analysis on spectrum-based fault localization," in *Proceedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC '15)*, vol. 1, pp. 17–22, IEEE, Taichung, Taiwan, July 2015.
- [3] W. E. Wong and V. Debroy, "A survey of software fault localization," Tech. Rep., The University of Texas at Dallas, Richardson, Tex, USA, 2009.
- [4] W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, "Effective software fault localization using an RBF neural network," *IEEE Transactions on Reliability*, vol. 61, no. 1, pp. 149–169, 2012.
- [5] W. Wen, "Software fault localization based on program slicing spectrum," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 1511–1514, Zurich, Switzerland, June 2012.
- [6] X. Ju, S. Jiang, X. Chen, X. Wang, Y. Zhang, and H. Cao, "HSFal: effective fault localization using hybrid spectrum of full slices and execution slices," *Journal of Systems and Software*, vol. 90, no. 1, pp. 3–17, 2014.
- [7] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan, "Identifying bug signatures using discriminative graph mining," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA '09)*, pp. 141–151, Association for Computing Machinery, Chicago, Ill, USA, July 2009.
- [8] D. Gopinath, R. N. Zaeem, and S. Khurshid, "Improving the effectiveness of spectra-based fault localization using specifications," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12)*, pp. 40–49, IEEE, September 2012.
- [9] 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 (ICSE '02)*, pp. 467–477, May 2002.
- [10] R. Abreu, P. Zoetewij, and A. J. C. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing (PRDC '06)*, pp. 39–46, Riverside, Calif, USA, December 2006.
- [11] T. Y. Chen, X. Xie, F.-C. Kuo, and B. Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 4, article 40, 2013.
- [12] D. Hao, L. Zhang, Y. Pan, H. Mei, and J. Sun, "On similarity-awareness in testing-based fault localization," *Automated Software Engineering*, vol. 15, no. 2, pp. 207–249, 2008.
- [13] D. Hao, T. Xie, L. Zhang, X. Wang, J. Sun, and H. Mei, "Test input reduction for result inspection to facilitate fault localization," *Automated Software Engineering*, vol. 17, no. 1, pp. 5–31, 2010.
- [14] G. Dandan, W. Tiantian, S. Xiaohong, and M. Peijun, "A test-suite reduction approach to improving fault-localization effectiveness," *Computer Languages, Systems and Structures*, vol. 39, no. 3, pp. 95–108, 2013.
- [15] J. Xuan and M. Monperrus, "Test case purification for improving fault localization," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 52–63, ACM, November 2014.
- [16] C. Gong, Z. Zheng, W. Li, and P. Hao, "Effects of class imbalance in test suites: an empirical study of spectrum-based fault localization," in *Proceedings of the 36th Annual IEEE International Computer Software and Applications Conference Workshops (COMPSACW '12)*, pp. 470–475, IEEE, Izmir, Turkey, July 2012.
- [17] Y. Gao, Z. Zhang, L. Zhang, C. Gong, and Z. Zheng, "A theoretical study: the impact of cloning failed test cases on the effectiveness of fault localization," in *Proceedings of the 13th International Conference on Quality Software (QSIC '13)*, pp. 288–291, Nanning, China, July 2013.
- [18] P. Rao, Z. Zheng, T. Y. Chen, N. Wang, and K. Cai, "Impacts of test suite's class imbalance on spectrum-based fault localization

- techniques,” in *Proceedings of the 13th International Conference on Quality Software (QSIC '13)*, pp. 260–267, Najing, China, July 2013.
- [19] V. Debroy and W. E. Wong, “Insights on fault interference for programs with multiple bugs,” in *Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE '09)*, pp. 165–174, IEEE, Karnataka, India, November 2009.
- [20] W. Eric Wong, V. Debroy, and B. Choi, “A family of code coverage-based heuristics for effective fault localization,” *Journal of Systems and Software*, vol. 83, no. 2, pp. 188–208, 2010.
- [21] 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, 2012.
- [22] W. Eric Wong, V. Debroy, R. Gao, and Y. Li, “The dstar method for effective software fault localization,” *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 290–308, 2014.
- [23] W. E. Wong and Y. Qi, “BP neural network-based effective fault localization,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 4, pp. 573–597, 2009.