Impacts of Test Suite's Class Imbalance on Spectrum-Based Fault Localization Techniques

CONFERENCE PAPER · JANUARY 2013
DOI: 10.1109/QSIC.2013.18

CITATION
1

DOWNLOADS
12

VIEWS
31

5 AUTHORS, INCLUDING:

Zheng Zheng
Beihang University (BUAA)
51 PUBLICATIONS  181 CITATIONS

Available from: Zheng Zheng
Retrieved on: 03 July 2015
Impacts of Test Suite’s Class Imbalance on Spectrum-Based Fault Localization Techniques

Peifeng Rao
School of Automation Science and Electrical Engineering
Beihang University, Beijing, 100191, China
peifengrao@163.com

Tsong Yueh Chen
Swinburne University of Technology
Hawthorn, Victoria, 3122, Australia
tchen@swin.edu.cn

Zheng Zheng
School of Automation Science and Electrical Engineering
Beihang University, Beijing, 100191, China
zhengz@buaa.edu.cn

Nan Wang, Kaiyuan Cai
School of Automation Science and Electrical Engineering
Beihang University, Beijing, 100191, China
wangnan1990@asee.buaa.edu.cn, kycai@buaa.edu.cn

Abstract—Spectrum-based fault localization (SBFL) uses the execution results of test cases to debug. There are two types of SBFL techniques: one using conventional slices, and the other using metamorphic slices. This paper investigates the ratio between non-violated and violated metamorphic test groups of test suites for SBFL techniques using metamorphic slices. We have observed that the higher the ratio of passed metamorphic test groups to failed metamorphic test groups, the less effective the SBFL techniques using metamorphic slices. This observation is consistent with what has been observed in SBFL techniques using conventional slices. Besides, a new real-life fault in schedule2 of Siemens Suite is identified in our experiments.

Keywords—spectrum-based fault localization, test oracle, class imbalance, metamorphic testing, metamorphic slices

I. INTRODUCTION

Recently much attention has been paid on spectrum-based fault localization (SBFL) for its simple principle and high fault localization effectiveness. There are two types of SBFL techniques: one using conventional slices, which require a test oracle to determine whether a test case is passed or failed [1-4]; and the other using metamorphic slices, which can be applied irrespective of having a test oracle or not [5-7].

The phenomenon of class imbalance, which exists in many disciplines, means that some classes have far more samples than other classes [8]. Actually, it also exists in the test suites for fault localization, in which we need failed test cases and passed test cases in order to predict the location of the faulty statement. However, in practice, it is much easier to find passed test cases than failed test cases. Thus, it is very likely to have an imbalance between the numbers of passed and failed test cases in a test suite for SBFL, and hence we are interested to investigate the impact of such an imbalance. Previous studies [9-13] have proved that the fault localization effectiveness of SBFL techniques depends on the characterizations of test suites. However, most of these researches focus on the impacts of the size of a test suite, rather than the composition of a test suite. Our previous work [14] has investigated the impact of the ratio between the numbers of passed and failed test cases in a test suite for SBFL using conventional slices. However, this study focuses on SBFL using metamorphic slices. In this case, the class proportion of test suites is defined as the ratio of non-violated metamorphic test groups to violated metamorphic test groups (details given in Section II-B). In this paper, the investigated class imbalance only covers the scenario of having more non-violated metamorphic test groups than violated metamorphic test groups in a test suite, because this scenario reflects the realistic situation.

The remainder of this paper is organized as follows: Section II explains the background and motivation is provided in Section III. Section IV describes the experimental setup. Experimental results and their analyses are presented in Section V. Section VI discusses the threats to validity. Conclusions are given in Section VII.

II. BACKGROUND

A. Spectrum-based fault localization

SBFL techniques evaluate the fault localization effectiveness by risk evaluation formulas. In this study, ten popular risk evaluation formulas [14] listed in Table I are selected to be our experimental subjects. With a given test suite, for any statement, $N_{np}$ and $N_{nf}$ represents the numbers of passed and failed test cases executing it, respectively; $N_{np}$ and $N_{nf}$ represents the numbers of passed and failed test cases not executing it, respectively.

B. Metamorphic testing and metamorphic slices

Metamorphic testing (MT) [5] is proposed to alleviate the oracle problem. It uses the specific properties of an algorithm, namely metamorphic relations (MRs), to verify the relationship between multiple but related test cases and their outputs. In applying MT, firstly the testers need to identify an MR of the software under test, and choose a test
<table>
<thead>
<tr>
<th>Name</th>
<th>Formula expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarantula</td>
<td>( \frac{N_v}{N_n + N_v} )</td>
</tr>
<tr>
<td>Jaccard</td>
<td>( \frac{N_v}{(N_n + N_v)^2} ) ( \min ) ( \frac{N_v}{(N_n + N_v)^2} )</td>
</tr>
<tr>
<td>Euclid</td>
<td>( \sqrt{N_n + N_v} )</td>
</tr>
<tr>
<td>Hamann</td>
<td>( N_n + N_v )</td>
</tr>
<tr>
<td>Ochiai</td>
<td>( N_v \sqrt{N_n + N_v(N_n + N_v)} )</td>
</tr>
<tr>
<td>Ochiai2</td>
<td>( \frac{N_v N_n}{(N_n + N_v)(N_n + N_v + N_n + N_v)} )</td>
</tr>
</tbody>
</table>
| Wong2      | \( N_n - \left\{ \begin{array}{l} N_v \\text{if } 2 \leq N_v \leq 10 \\
\quad 2.8 + 0.001(N_v - 10) \text{ if } N_v > 10 \end{array} \right. \) |
| Wong3      | \( \frac{N_v}{(N_n + N_v + N_v + 1000N_v N_v / N_v)} \)                           |
| Zoltar     | \( \frac{N_v - N_n}{N_n(N_n + N_v)(N_n + N_v + N_v + N_v) + (N_n + N_v)N_v(N_v + N_v)} \) |
| Harmonic   | \( \frac{N_n N_v}{(N_n N_v - N_n N_n)(N_v + N_v)(N_v + N_v + N_v) + (N_n + N_v)N_v(N_v + N_v)} \) |

TABLE I. INVESTIGATED TEN FORMULAS OF SBFL TECHNIQUES

To evaluate the risk of each program component using a risk source test cases, we refer to “execution metamorphic slice” to refer to “execution metamorphic slice”. Failed or passed in normal testing, respectively. We will use the corresponding statement in e_mslice(MR, T) as an individual test case, and the metamorphic test results (violated or non-violated) correspond to the test results (failed or passed). After replacements of the corresponding components, the procedure for SBFL using conventional slices can be similarly applied to SBFL using metamorphic slices. Details can be found in [5].

III. MOTIVATION AND RESEARCH QUESTIONS

Class imbalanced test suites mean far more test cases of one class than the test cases of the other class. Previous study [14] has shown that class imbalance commonly occurs in test suites. Table II shows the mean ratio of non-violated to violated test groups in Siemens and grep program suites. This table shows that the class imbalance phenomenon is as serious as test suites of test cases as shown in [14]. Moreover, there is a huge difference on the class imbalance degrees of formula. However, for programs without a test oracle, the test results are not available. SBFL using metamorphic slices is proposed to alleviate this problem. Its relevant matrix and vector slices are constructed in a similar way as for SBFL using conventional slices (Figure 1).

\[
S := (s_1, s_2, \ldots, s_n) \left[ \begin{array}{cccc}
\vdots & \vdots & \vdots & \vdots \\
1/0 & 1/0 & \ldots & 1/0 \\| v/n \\
1/0 & 1/0 & \ldots & 1/0 \\| v/n \\
\vdots & \vdots & \vdots & \vdots \\
1/0 & 1/0 & \ldots & 1/0 \\| v/n \\
\end{array} \right]
\]

\[
M := \left[ \begin{array}{c}
\vdots \\
p \\
\vdots \\
m \\
\end{array} \right] \left[ \begin{array}{cccc}
\vdots & \vdots & \vdots & \vdots \\
1/0 & 1/0 & \ldots & 1/0 \\| v/n \\
1/0 & 1/0 & \ldots & 1/0 \\| v/n \\
\vdots & \vdots & \vdots & \vdots \\
1/0 & 1/0 & \ldots & 1/0 \\| v/n \\
\end{array} \right]
\]

Figure 1. SBFL techniques using e_mslice

In Figure 1, the vector M is the test suite containing k test groups and vector S represents n executable statements of the program. Matrix P represents the program execution information which contains program spectrum in the first n columns and test results in the last column. The program spectrum is constructed by using e_mslice, and the binary value 1 or 0 denotes the membership or non-membership of the corresponding statement in e_mslice(MR, T). The expression “v/n” records the metamorphic test result for m, in which “v” indicates violated and “n” indicates non-violated.

Comparing SBFL using metamorphic slices with that using conventional slices, we find the correspondences between their key components. For example, e_mslice corresponds to e_slice, a test group T'=(T', T') corresponds to an individual test case t, and the metamorphic test results (violated or non-violated) correspond to the test results (failed or passed). After replacements of the corresponding components, the procedure for SBFL using conventional slices can be similarly applied to SBFL using metamorphic slices. Details can be found in [5].

TABLE II. THE CLASS IMBALANCE PHENOMENON OF TEST SUITES IN SIEMENS SUITE AND GREP

<table>
<thead>
<tr>
<th>Program</th>
<th>Non-violated: Violated</th>
<th>Program</th>
<th>Non-violated: Violated</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_tokens</td>
<td>190.08:1</td>
<td>schedule2</td>
<td>84.33:1</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>21.20:1</td>
<td>tot_info</td>
<td>110.96:1</td>
</tr>
<tr>
<td>replace</td>
<td>237.00:1</td>
<td>tcas</td>
<td>30.85:1</td>
</tr>
<tr>
<td>schedule</td>
<td>195.89:1</td>
<td>grep</td>
<td>385.28:1</td>
</tr>
</tbody>
</table>
different programs.

Inspired by the results presented in [14], two questions arise:

Q1: What are the effects that class proportions between non-violated and violated test groups have on SBFL using metamorphic slices?

Q2: Is there any significant difference of the impacts due to class imbalance on conventional and metamorphic slices?

These questions are important because: (1) No previous study has been conducted on the class imbalance of test suites of metamorphic test groups. (2) The answers may provide further information and insight to design effective test suites for SBFL. Following [14], we conduct the experiments using various class proportions and sizes of test suites to study their impacts on the effectiveness of SBFL using metamorphic slices.

IV. EXPERIMENTAL SETUP

A. Subject Programs

Siemens and grep program suite have been widely used [12, 15-17]. So we select them as programs in our research. Table III shows some statistics and relevant information of Siemens Suite and grep. The versions of all the 7 programs in Siemens Suite used in our experiments are 2.0. The source code for grep used in our experiments is from v1 of grep1.2. All the programs were downloaded from SIR website [18].

<table>
<thead>
<tr>
<th>Program</th>
<th>Loc</th>
<th>Test Cases</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_tokens</td>
<td>472</td>
<td>4130</td>
<td>Lexical analyzer</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>399</td>
<td>4115</td>
<td>Lexical analyzer</td>
</tr>
<tr>
<td>replace</td>
<td>512</td>
<td>5542</td>
<td>Pattern recognition</td>
</tr>
<tr>
<td>schedule</td>
<td>292</td>
<td>2650</td>
<td>Priority scheduler</td>
</tr>
<tr>
<td>schedule2</td>
<td>301</td>
<td>2710</td>
<td>Priority scheduler</td>
</tr>
<tr>
<td>tas info</td>
<td>440</td>
<td>1608</td>
<td>Altitude separation</td>
</tr>
<tr>
<td>grep</td>
<td>15633</td>
<td>2982(MR1)</td>
<td>5003(MR2)</td>
</tr>
</tbody>
</table>

B. Definitions of MRs

For each program, we use the three MRs as used in [5]. We will only explain the MRs for grep in this paper.

The used MRs of grep are all related to the regular expression analyzer which is one of its key components. Given a regular expression and some input files, grep searches these files line by line for the strings which match the regular expression, and then prints all the matched lines. Although an MR may involve more than one source test case and more than one follow-up test case, in this study, all test groups only involve one source test case and one follow-up test case. The source and follow-up test cases are denoted as $I_s$ and $I_f$, respectively, and their outputs are denoted as $O_s$ and $O_f$, respectively. All the three MRs involve $I_f$ which is equivalent to $I_s$. So a correct program should satisfy the relation that $O_f$ is the same as $O_s$. The three MRs are briefly described as follows.

(1) MR1: Completely decomposing the bracketed sub-expression.

MR1 considers how to completely decompose a bracket sub-expression like “[x-y]” where $x$ and $y$ are digits (or characters) and $x<y$. For the regular expression of “[x-y]”, it means that any digit or character within the region of $x$ to $y$ should match. Thus, if $I_s$ contains such an expression, $I_f$ can be constructed by completely decomposing the bracket in $I_s$, using “|” which means “or”. For example, an expression “[1-4]” in $I_s$ can be transformed to “1|2|3|4” or “1|3|2|4” in $I_f$. In this case, $O_f$ should be the same as $O_s$. Note that, in our experiments, we consider the default C locale, in which characters are sorted according to their ASCII codes.

(2) MR2: Splitting the bracketed structure.

Similar to MR1, MR2 also considers the decomposition of bracketed structures. The difference of MR2 from MR1 is that MR2 keeps the bracket structure. It uses “|” to split a bracket structure into two bracket structures. Let us take the regular expression “[1-4]” in $I_s$ as an example. It can be transformed to “[1][2][3][4]” in $I_f$. According to the specification of grep, there should be no differences between $O_f$ and $O_s$.

(3) MR3: Bracketing simple characters.

MR3 considers the replacement of a simple character with itself enclosed by the brackets. If some simple characters in $I_s$ are not reserved words with special meanings, $I_f$ can be constructed by bracketing some of these characters. Suppose $I_s$ contains “abcd”, which can be replaced by “[a][b][c][d]” in $I_f$. In this case, $O_f$ and $O_s$ should be identical.

C. Source test suites

For all the MRs used in this experiment, each of their test groups only involves a source test case and a follow-up test case. After specifying a test suite for all source test cases, we can use the defined MR to generate a test suite for the corresponding follow-up test cases. For the 7 programs in Siemens Suite, we use the test cases provided by SIR as the source test cases. For grep, the 807 test cases provided by SIR are not used in these experiments, as only a small portion of them are “eligible source test cases” with respect to our MRs. Similar to [5], we use a test pool with 171634 random test cases of which 2982 test cases are eligible source test cases for MR1, 5003 for MR2, and 2084 for MR3. The numbers of source test cases of each program are listed in Table III. All MRs for Siemens Suite use the same source test cases.

D. Mutant generation

Although SIR has several mutants for Siemens Suite and grep, only a few mutants are suitable for this study, because many of them are equivalent with respect to the used MRs. In order to have sufficient non-equivalent mutants for experimentation, we randomly generate 405 mutants for Siemens Suite and 234 mutants for grep. The numbers of useful mutants (that is, having violated test groups) are listed.
in Table IV. We also exclude mutants that could not be compiled, do not have any violated test group or have exceptional exists.

<table>
<thead>
<tr>
<th>Program</th>
<th>MR1</th>
<th>MR2</th>
<th>MR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>print tokens</td>
<td>11</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>print tokens2</td>
<td>4</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>replace</td>
<td>44</td>
<td>45</td>
<td>40</td>
</tr>
<tr>
<td>schedule</td>
<td>19</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>schedule2</td>
<td>17</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>tcas</td>
<td>7</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>tot info</td>
<td>27</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>grep</td>
<td>58</td>
<td>51</td>
<td>79</td>
</tr>
</tbody>
</table>

Our experiments only consider first-order faults (that is, single-fault mutant) excluding the omission faults. Also, only two types of mutant operators are used; namely, statement mutation and operator mutation. For statement mutation, there are mainly two operations which are the replacement of a “continue” statement with a “break” statement (and vice versa), and the replacement of the label of a “goto” statement with another valid label. For operator mutation, an arithmetic (or logical) operator is replaced by another arithmetic (or logical) operator. For more information, readers may refer to [5].

E. Test suite generation

This section presents the test suite generation strategies for our experiments. To study the impacts of class imbalanced test suite on SBFL using metamorphic slices, a large number of test suites with different class proportions levels need to be generated. Prior to presenting test suite generation strategy, let us define some notations first in the following:

- $L$ denotes a list of class proportion levels, and $L=(l_1, l_2, ..., l_m)$, in which $l_i$ ($1 \leq i \leq m$) is the ratio of non-violated test groups to violated test groups in a test suite and $m$ is the total number of levels.
- $Y$ and $V$ denote the subsets of all non-violated test groups and violated test groups of a test suite for a particular faulty version, respectively. The test suite is denoted as $TS = V \cup Y$.

Our test suite generation strategies are to generate a list of test suites $\tilde{\mathcal{T}}=(\tilde{T}_1, ..., \tilde{T}_m)$, such that for any test suite $\tilde{T}_i$ of $\tilde{\mathcal{T}}$, its ratio of non-violated test groups to violated test groups is equal to $l_i$. Two test suite generation strategies are used in our experiments. The first strategy (referred to as Strategy 1) generates test suites of constant size but varying class portions. That is, all test suites of $\tilde{\mathcal{T}}$ are of the same size, say $c$.

**Test suite generation strategy 1: Strategy 1(c, $TS$)**

Step 1: Split $TS$ into $Y$ and $N$, which are the subsets of non-violated test groups and violated test groups, respectively.

Step 2: For $i=1$ to $m$

Step 2.1: Violated test group selection. If $|V| \geq c/(1+l_i)$, then randomly select $\lceil c/(1+l_i) \rceil$ test groups from $V$ to form $V'$; else let $T_i = \Phi$ and continue;

Step 2.2: Non-violated test group selection. If $|Y| \geq d*|l_i|$, then randomly select $\lceil d*|l_i| \rceil$ test groups from $Y$ to form $Y'$; else let $T_i = \Phi$ and continue;

Step 2.3: New test suite generation. The new test suite $T_i$ is defined as $T_i = V' \cup Y'$;

Step 3: $\tilde{\mathcal{T}}=(T_1, ..., T_m)$, which is the output for Strategy 1(c, $TS$).

Note that, $|V|$ and $|Y|$ denote the numbers of test groups in $V$ and $Y$, respectively. $\lfloor \cdot \rfloor$ function returns the value of a real number rounded downwards to the nearest integer. If $T_i = \Phi$, it means that we cannot select enough test groups from the test suite $TS$ to satisfy the setting of $l_i$ and $c$.

The second strategy (referred to as Strategy 2) generates test suites with different class proportions and a fixed number of violated test groups. That is, for any test suite $T_i$ of $\tilde{\mathcal{T}}$ ($1 \leq i \leq m$), the number of violated test groups in $T_i$ is a constant ($d$). Strategies 1 and 2 are very similar with differences only in Steps 2.1 and 2.2 which are detailed as follows.

**Test suite generation strategy 2: Strategy 2(d, $TS$)**

Step 2.1: Violated test groups selection. If $|V| \geq d_1$, then randomly select $d_1$ test groups from $V$ to form $V'$; else let $T_i = \Phi$ and continue;

Step 2.2: Non-violated test groups selection. If $|Y| \geq d*|l_i|$, then randomly select $d*|l_i|$ test groups from $Y$ to form $Y'$; else let $T_i = \Phi$ and continue;

Note that, $T_i = \Phi$ means impossible to construct a test suite from the original $TS$, which satisfies the setting of $l_i$ and $d$.

F. Experimental setups

In this section, the detailed experimental steps are presented. Some notations are first introduced in the following:

- $S$ denotes a list of test suite sizes, that is, $S=(s_1, ..., s_n)$, where $s_j$ ($1 \leq j \leq n$) is the total number of test groups in the $j$th test suite of a list of test suites with length $n$.
- $C$ denotes a list of the numbers of violated test groups, that is, $C=(c_1, ..., c_h)$, where $c_k$ ($1 \leq k \leq h$) is the number of violated test groups in the $k$th test suite of a list of test suites with length $h$.

In the following, two experiments are designed. For a given pair of program $P$ and a risk evaluation formula $F$ as stated in Table I, the first experiment uses different settings of class proportions with a fixed size of test suites.
TABLE V. THE EXPENSE OF TEN FORMULAS IN DIFFERENT CLASS PROPORTION LEVELS WHILE $s_i(i=2, 4, 7)$ (%) 

<table>
<thead>
<tr>
<th>Formulas</th>
<th>program</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
<th>$I_4$</th>
<th>$I_5$</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarantula</td>
<td>Siemens</td>
<td>41.50</td>
<td>42.80</td>
<td>42.59</td>
<td>40.71</td>
<td>42.45</td>
<td>44.07</td>
</tr>
<tr>
<td>Jaccard</td>
<td>Siemens</td>
<td>28.48</td>
<td>28.54</td>
<td>27.80</td>
<td>33.64</td>
<td>33.65</td>
<td>34.50</td>
</tr>
<tr>
<td>Ochiai</td>
<td>Siemens</td>
<td>4.98</td>
<td>9.59</td>
<td>10.80</td>
<td>10.76</td>
<td>10.56</td>
<td>11.80</td>
</tr>
<tr>
<td>Euclid</td>
<td>Siemens</td>
<td>34.03</td>
<td>33.33</td>
<td>29.04</td>
<td>78.82</td>
<td>78.57</td>
<td>83.07</td>
</tr>
<tr>
<td>Wong2</td>
<td>Siemens</td>
<td>4.39</td>
<td>42.29</td>
<td>43.20</td>
<td>61.00</td>
<td>63.01</td>
<td>62.52</td>
</tr>
<tr>
<td>Ochiai2</td>
<td>Siemens</td>
<td>46.01</td>
<td>47.03</td>
<td>45.94</td>
<td>46.80</td>
<td>48.28</td>
<td>49.39</td>
</tr>
<tr>
<td>Harmonic</td>
<td>Siemens</td>
<td>38.09</td>
<td>37.61</td>
<td>37.88</td>
<td>39.89</td>
<td>40.60</td>
<td>42.39</td>
</tr>
<tr>
<td>grep</td>
<td>--</td>
<td>8.60</td>
<td>8.63</td>
<td>8.74</td>
<td>8.60</td>
<td>8.10</td>
<td>9.23</td>
</tr>
<tr>
<td>grep</td>
<td>--</td>
<td>33.83</td>
<td>32.92</td>
<td>33.16</td>
<td>31.31</td>
<td>24.13</td>
<td>32.55</td>
</tr>
</tbody>
</table>

**Experiment 1:**

Step 1: Execute all faulty versions of $P$ and identify the non-violated test groups and violated test groups of the test suite used for each faulty version.

Step 2: Set $S=(s_1, s_2, \ldots, s_n)$;

Step 3: For each faulty version $P'$ of program $P$,

Step 3.1: Denote $V'$ and $Y'$ as the violated and non-violated test groups for faulty version $P'$, and $TS'=V' \cup Y'$;

Step 3.2: For $j=1$ to $n$ Construct $T_j=Strategy_1(s_j, TS')$;

Step 3.3: For $j=1$ to $m$,

For $i=1$ to $m$

If $T_i = \Phi$, then $e^{P'}_{i,j} = -1$;
else (1) Use the formula $F$ to compute the suspiciousness of each statement using $T_i$;
(2) Calculate the expense of looking for the faulty statement $s_f$, denoted as $e^{P'}_{i,j}$;

Step 4: $E^P_{i,j} = \left( \sum_{P' \neq P} e^{P'}_{i,j} \right) / r$, 

where $r$ is the number of faulty versions which satisfy $e^{P'}_{i,j} \neq -1$.

In Step 4, $E^P_{i,j}$ is the output of this experiment process, and it represents the mean expense [19] of looking for a faulty statement under the setting $I_i$ and $s_j$. It is obvious that $r$ is different for various settings, because $T_i$ may be $\Phi$ according to Strategy1 or Strategy2. Intuitively speaking, if a method has a smaller expense value than the other, it is better with respect to fault localization. Our experiments set a threshold $\ell$ (=70) to $r$ for each program. If $r$ is a small number (i.e. less than the threshold $\ell$), we cannot generate enough test suites to measure the fault localization effectiveness objectively. Thus, we set $E^P_{i,j}$ to be a special value -1, and mark the result as symbol “--” in Tables V and VI. In the experiment shown in Section V-A,
we set \( L = \{1, 2, 4, 8, 16, 32, 64, 128\} \) and \( S = \{50, 100, 200, 300, 400, 500, 800\} \). Besides, to eliminate any bias in the test suite generation strategy, the experimental process had been randomly repeated for ten times to calculate the mean values.

The second experiment is to get the fault localization results of the formula \( F \) on the program \( P \) under different setting of class proportions with a fixed number of violated test groups. The basic steps of the second experiment are similar to those of Experiment 1. The differences between the two experiments only come from Step 2 and Step 3.2.

### Experiment 2:

**Step 2:** Set \( C = (c_1, c_2, \ldots, c_h) \);

**Step 3.2:** For \( k = 1 \) to \( h \) Construct \( \tilde{T}_f = \text{Strategy}(2(c_1, TS)) \).

In the experiment shown in Section V-B, we set \( L = \{1, 2, 4, 8, 16, 32, 64, 128\} \) as the first experiment and \( C = \{1, 2, 4, 8, 16, 32, 64, 128\} \). Similar to the first experiment, the experimental process had been randomly repeated for ten times.

### V. Experimental Results

#### A. Results of Experiment 1

The results of Experiment 1 are shown in Figure 2 and Table V. Figure 2 (a-d) illustrates the mean expenses of four “risk evaluation formulas” (or simply “\( F^r \)” by examining different number of statements, where horizontal axis represents class proportions with their level index in \( L \), vertical axis represents the expense \( E_f(i, j) \) (Due to page limit, only parts of the results are shown in the figures). The plots show three representative trends of the mean expenses).

Table V shows the mean expenses of ten \( F \) in different class proportion levels and test suite sizes. For each cell in the table, its value is \( E_f(i, j) \) as defined by Equation (1). Here, \( P \) can be Siemens or grep. The last column of Table V shows the impacts of class imbalanced test suites on the effectiveness of SBFL formulas, where “\( N \)” represents negative impacts, “\( P \)” represents positive impacts and “-” represents no obvious impacts. Consider the cell corresponding to the setting of \( I_l \) and \( s_2 \), and Wong3 with Siemens. Its value of 20.72% means that the mean expense of Wong3 detecting a fault is 20.72% at \( I_l=1 \) and \( s_2=100 \).
Figure 2 shows that the expense of fault localization for most formulas will increase with the increase of class imbalance. From Table V, most of the ten formulas show the same trend, that is, if \( l_i < l_j \), in general fault localization effectiveness in the test suites with class proportion level \( l_i \) is better than that in the test suites with class proportion level \( l_j \). For example, given the size of test suite \( s_j = 100 \), with class proportions varying from \( l_i \) to \( l_j \), the mean expense of Wong3 detecting a fault varying from 20.72%, 22.69%, 26.30%, to 41.79%. This increasing trend can also be observed in most of the other settings.

The results of Experiment 1 clearly show that the class imbalanced test suites have negative impacts on the effectiveness of SBFL using metamorphic slices. For test suites of the same size, the less imbalanced test suites always contain more violated test groups than the more imbalanced ones. Since the number of violated test groups may affect the results, Experiment 2 is designed to serve this purpose.

B. Results of Experiment 2

The results of Experiment 2 are illustrated by Figure 3 and Table VI, and the settings for the figures and table are similar to those of Experiment 1. As observed from these results, a significant increase of the fault suite size has a slight improvement of the fault localization effectiveness at the best. For example, in Figure 3(a), although the number of test groups has increased more than 64 times from \( l_1 \) to \( l_8 \), the expense are increased from 49.50% to 61.28%. In Figure 3(c), the number of test groups has increased more than 64 times from \( l_1 \) to \( l_8 \), but the expense almost have no difference. Nevertheless, the results show the negative effects of class imbalance.

Table VI shows that five formulas have an obvious increase of expenses with an increase of class imbalance levels (Jaccard, Hamann, Euclid, Ochiai and Wong2). The other five formulas show a fluctuated trend in a small region of expenses or a slight decrease of expenses from \( l_1 \) to \( l_8 \) (Tarantula, Zoltar, Wong3, Ochai2 and Harmonic).

C. Finding a Real-life fault in schedule2

A surprising but pleasant result of our experiments is the identification of one real-life fault in the schedule2 of Siemens Suite. In our experiment, we first checked the validity of the selected MRs for all subject programs by executing the original programs with some groups of source and follow-up test cases, and then verifying whether the relevant MRs were satisfied or not. We found violations of the third MR for schedule2. Hence, we carried out an extensive validation of the relevant MR, test cases, outputs and original program for schedule2. This has led us to reveal a fault in schedule2.

The program schedule2 performs priority scheduling. The program and the relevant description clearly show that the valid range of priority values is from 1 to 3 inclusive. Also, there are codes explicitly showing how to deal with inputs containing invalid priority values smaller than 0 and greater than 3. However, the invalid priority of 0 is not properly handled by the current version. Thus, a fault occurs. This fault was fixed by inserting “if (command ==1 && prio ==0) continue;” before line 62 “schedule (command, prio, ratio);”.

In fact, there are two other programs of Siemens Suite containing real-life faults, which were detected by MT [5]. Hence, MT is not only able to alleviate the oracle problem, but also can be used as a new test case selection strategy, which complements existing test case selection strategies.

VI. Threats to Validity

In experiments, we used Siemens Suite and grep as subject programs. Although they are widely used in fault localization community, we still need more programs to further validate the effectiveness of our conclusions. Another internal threat comes from the type of mutants used in this work. Although these mutants are randomly generated, each mutant, however, only contains one fault and the types of faults are limited.

There are also threats to the validity of the study, including the implementation of SBFL formulas, the generation of test groups, and the examination of test groups’ execution outputs. We expect that such threats are unlikely, because we have carefully implemented the experiments and conducted a thorough testing at each implementation step. Other threats come from the evaluation metric. However, the used metric Expense has been extensively used by the community [5, 12, 14, 19].

VII. Conclusion

This paper takes an empirical study of the class imbalance problem in software fault localization community. By analyzing experimental results in Siemens Suite and grep, we can get the following conclusions: (1) Comparing with the results in [14], the impacts of class imbalance are similar for SBFL using conventional slices [14] or using metamorphic slices. The higher the ratio of non-violated metamorphic test groups (or passed test cases) to violated metamorphic test groups (or failed test cases) is, the less effective the SBFL techniques are. (2) Our empirical study provides evidence on the exchangeability between the role of test cases and the role of metamorphic test groups; the exchangeability between the role of execution slices and the role of metamorphic slices; and the exchangeability between the role of pass/fail of test cases and the role of satisfaction/violation of metamorphic test groups. (3) A new real-life fault in schedule2 program is identified. It shows that MT is not only useful to alleviate the oracle problem, but also can be used as a new test case selection strategy.

Acknowledgment

This project is partially supported by Australian Research Council Discovery Project (DP120104773), and the National Natural Science Foundation of China (60904066).

References


