Exploring the Usefulness of Unlabelled Test Cases in Software Fault Localization

Xiao-Yi Zhang, Zheng Zheng, Kai-Yuan Cai

PII: S0164-1212(17)30158-9
DOI: 10.1016/j.jss.2017.07.027
Reference: JSS 10010

To appear in: The Journal of Systems & Software

Received date: 7 August 2016
Revised date: 12 July 2017
Accepted date: 18 July 2017


This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
Highlights

- The usefulness of unlabelled test cases in fault localization is explored.
- An approach using test classification is proposed to utilize unlabelled test cases.
- A classifier utilizing the domain knowledge of fault localization is designed.
- Experiments are conducted to demonstrate the usefulness of our approach.
Exploring the Usefulness of Unlabelled Test Cases in Software Fault Localization

Xiao-Yi Zhang, Zheng Zheng,∗, Kai-Yuan Cai

∗Department of Automatic Control, Beihang University, BUAA, Beijing

Abstract
In automatic software fault localization techniques, both the coverage and the testing outcomes of the provided test suite are considered to be essential information. The problem occurs when test oracles do not exist. Specifically, the test suite will contain a large number of unlabelled test cases, i.e., test cases whose output is not identified as being either correct (passing) or incorrect (failing). Such unlabelled test cases cannot be directly used, thereby leading to a degradation of localization effectiveness. In this paper, we propose an approach based on test classification to enable the use of unlabelled test cases in localizing faults. In our approach, unlabelled test cases are classified based on their execution information and are then assigned corresponding estimated labels to allow them to be utilized in fault localization. Experimental results show that with the utilization of these newly labelled test cases, the effectiveness of fault localization can indeed be improved.

Keywords: software testing, software fault localization, oracle problem, test classification, unlabelled test cases

1. Introduction
In software testing, the oracle problem, which refers to the problem of determining the correctness of a program’s behaviour under testing, has long been

∗Corresponding author
Email addresses: xiaoyizhang@buaa.edu.cn (Xiao-Yi Zhang), zhengz@buaa.edu.cn (Zheng Zheng), kycai@buaa.edu.cn (Kai-Yuan Cai)
recognized as a significant problem [1]. To date, various approaches based on
specification, derivation, and implication have been proposed to facilitate the
verification of program correctness [2, 3]. However, there are still many situations
in which test oracles are absent [4].

Once a failure is detected during testing, fault localization will be performed
to trace the root cause of the detected failure [5]. To date, various automated
approaches have been proposed to accelerate fault localization [6, 7, 8, 9]. Fault
localization is not only the first step of debugging but also a vital link in testing:
we test to see whether the program of interest contains faulty components, and
if it does, we switch from testing to fault localization to find the locations of
the detected faulty components as first step towards fixing these faults [6]. The
information generated during testing can also be utilized to support fault local-
ization; in this process, test oracles are important for deducing the possibility
that a specific set of components contains faults. Thus, the oracle problem,
which belongs to the domain of software testing, also has great influence on the
effectiveness of fault localization.

Consequently, the problem of coping with the absence of test oracles should
be further considered, from the perspective of fault localization. In software
testing, once a test case has been executed, the testers must record various pieces
of information, including the execution traces and the output correctness. Here,
we use the term “label” to refer to the identification of the output correctness,
i.e., passing or failing, for a test case. Specifically, a test case is said to be
passing or failing if its output is correct or incorrect, respectively. Then a test
case is said to be labelled if its status as either a passing or failing test case is
known; otherwise, it is said to be unlabelled. If oracles are absent, testers are
not able to determine the output correctness and labelling all the generated test
cases. However, program failures may also be detected based on relevant data
collected in various ways [10, 11, 12]. For example, if we are testing numerical
programs whose outputs cannot be verified, we can examine certain special
input values (e.g., $30^\circ, 45^\circ, 22.5^\circ, \ldots$ for the sine function) [2, 13]. As another
example, for optimization programs that are used to solve economic problems,
we may use the historical data obtained from typical data sets or experimental data from previous literature as benchmarks for verification [14, 15].

However, in fault localization, a large pool of test information is required to ensure accuracy [6]. In the above examples, the historical testing data and the special input values are very limited. As a result, although testers might be able to generate a large number of test cases from the input domain, the output correctness cannot be determined for the majority of test cases. That is to say, at the beginning of fault localization, only a few test cases are labelled; most are unlabelled. For most fault localization techniques, the correlations between the correctness of test results and the execution behaviours are very important clues to the locations of faulty components. Therefore, the unlabelled test cases cannot be directly utilized. Consequently, because of the insufficient amount of information available, these fault localization techniques may not be able to effectively identify fault locations.

Although they cannot be definitely verified, unlabelled test cases are still useful for exposing faulty components. It has been found that test cases that are adjacent in the execution space will probably exhibit similar behaviours in failure detection [16, 17], and the information contained in unlabelled test cases could be useful for fault analysis. Thus, if we want to improve the fault localization effectiveness without introducing additional data, one potential way to do so is to explore the utilization of the information associated with unlabelled test cases.

To pave the way for automatic fault localization in the absence of test oracles, we discuss how to make use of the information provided by unlabelled test cases in fault localization. Specifically, the following questions are explored:

1) Are unlabelled test cases useful for improving the effectiveness of fault localization?
2) How can we make use of unlabelled test cases during fault localization?

Our study is based on Spectra-Based Fault Localization (SBFL), one of the most famous automatic fault localization approaches. SBFL correlates program
failures with execution traces, thereby identifying the “suspicious” statements according to the difference in coverage frequency for the passing and failing test cases. Thus, the label of a test case is essential information. Although there are techniques for effectively utilizing the already labelled test cases for SBFL[9, 18], the ability to distinguish the likelihoods of fault-proneness for program statements could be insufficient. It is due to the limitation of the diversity of execution patterns contained in the already labelled test suite. Jiang et al. [19] observed that the effectiveness of SBFL deteriorates when only a small fraction of test cases can be used [19]. Therefore, it is appropriate to consider the problem of utilizing unlabelled test cases in SBFL.

We propose an approach based on test classification to solve the problem of how to utilize unlabelled test cases. The classifier, called the Suspiciousness Probability-based (SP) classifier, utilizes SBFL domain knowledge to assign an estimated label to each unlabelled test case. After classification, the newly labelled test cases are utilized in the calculation of statement suspiciousness values. By properly considering a greater number of test cases, the diversity of the investigated program execution patterns can be enriched. Consequently, more statements can be distinguished based on program spectra during fault localization. Experimental results show that our approach does help to improve the effectiveness of fault localization. Four research questions that are relevant to the performance of our approach are explored based on the results.

The reminder of the paper is organized as follows. Section 2 explains the background of Spectra-Based Fault Localization (SBFL) and then presents a motivating example for our work. Section 3 presents our approach, including the description of the proposed problem, the classifier, and the method of organizing the classification results for use in SBFL. Empirical studies and analyses of the obtained results are reported in Section 4. Section 5 analyses the threats to validity, followed by a review of related work in Section 6. Finally, we present our conclusion and summarize our contributions in Section 7.
2. Background and Motivation

2.1. Spectra-Based Fault Localization

Suppose that the program of interest, \( PG \), contains \( n \) statements, and let \( S = \{s_1, s_2, \ldots, s_n\} \) denote the set of statements. Let \( T \) denote the given test suite. Each statement \( s \in S \) can be assigned a tuple \( \langle a_{s,T}^{ef}, a_{s,T}^{ep}, a_{s,T}^{nf}, a_{s,T}^{np} \rangle \), where \( a_{s,T}^{ef} \) and \( a_{s,T}^{ep} \) represent the numbers of failing and passing test cases, respectively, in \( T \) that cover \( s \), and \( a_{s,T}^{nf} \) and \( a_{s,T}^{np} \) represent the numbers of failing and passing test cases, respectively, in \( T \) that do not cover \( s \). In addition, let \( F^T \) denote the total number of failing test cases, and let \( P^T \) denote the total number of passing test cases. During the SBFL process, each statement will be assigned a suspiciousness value based on the following two basic intuitive assumptions [20]:

1) Statements covered by more failing test cases are more likely to be faulty.
2) Statements covered by more passing test cases are less likely to be faulty.

Based on the above intuitions, the suspiciousness value of a statement \( s \) can be quantified by a metric \( R \) that is calculated using a formula involving the statistical quantities \( a_{s,T}^{ef}, a_{s,T}^{ep}, a_{s,T}^{nf}, a_{s,T}^{np}, F^T \), and \( P^T \). To date, more than 40 metrics have been proposed in accordance with the above two assumptions and based on various other heuristics. Following studies have shown that the failing test cases are more important than passing ones for localizing faulty components. Therefore, in the calculation of the suspiciousness metric \( R \), a higher weight should be assigned to \( a_{s,T}^{ef} \). Based on this conclusion, Naish et al. [8] proposed a famous metric called Op, which strictly adheres to the following rule: \( a_{s,T}^{ef} \) is first and \( a_{s,T}^{ep} \) is second. OP has been proven to be maximal in single-fault scenarios [21]. However, in multi-fault scenarios, its performance greatly decreases due to fault interference [22].

In this paper, the Ochiai metric is adopted. The Ochiai metric, proposed by Abreu et al.[23, 7], has been observed to show the best performance in many experimental studies [23, 24, 25]. Given the above statistical quantities, the
suspiciousness value of a statement \( s \) is calculated as follows:

\[
R_{Ochiai}(s, T) = \frac{\alpha_s^T}{\sqrt{(\alpha_s^T + \beta_s^T)(\alpha_s^T + \alpha_{sp}^T)}}.  
\]

(1)

\( R_{Ochiai} \) measures the suspiciousness of a statement in terms of similarity coefficients; consequently, the \((\alpha_s^T)^2\) term is proportional to \(\alpha_{sp}^T\), which has been experimentally observed to be a rational relation.

The output of SBFL is a suspiciousness list, in which statements are ranked in descending order according to their suspiciousness values. Statements with higher suspiciousness values are ranked ahead of those with lower suspiciousness values. The higher the suspiciousness value, and therefore the rank, of a faulty statement is, the sooner the testers will encounter this statement while checking through the suspiciousness list, consequently accelerating the process of fault localization.

### 2.2. Motivating Example for Exploring the Utilization of Unlabelled Test Cases

**Example: a function for UAVPP and its faulty version.** Consider a function for solving the Unmanned Aerial Vehicle Path Planning

<table>
<thead>
<tr>
<th>S</th>
<th>Original Program</th>
<th>Faulty Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (wallDis, objDir){</td>
<td>Input (wallDis, objDir)</td>
<td></td>
</tr>
<tr>
<td>double step, randDegree; void (*method)(double, double);</td>
<td>double step, randDegree; void (*method)(double, double);</td>
<td></td>
</tr>
<tr>
<td>if (wallDis &lt; 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>step = 1;</td>
<td>if (wallDis &lt; 8)</td>
<td></td>
</tr>
<tr>
<td>if (wallDis &lt; 5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>step = 2;</td>
<td>if (wallDis &lt; 5)</td>
<td></td>
</tr>
<tr>
<td>randDegree = 2;</td>
<td>randDegree = 2;</td>
<td></td>
</tr>
<tr>
<td>if (wallDis &lt; 1.5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>method = &amp;RRT;</td>
<td>method = &amp;RRT;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>else{</td>
<td>else</td>
<td></td>
</tr>
<tr>
<td>method = &amp;NN;</td>
<td>method = &amp;NN;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>if (objDir==0){</td>
<td>if (objDir==0){</td>
<td></td>
</tr>
<tr>
<td>return;</td>
<td>return;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: A function for solving the UAVPP and its faulty version.
problem, UAVPP [26]; the simplified code of this function and its faulty
version are shown in Fig. 1. The **UAVPP** function takes as input the
current state of the UAV, which can be simplified as a tuple of the form
\[(\text{objDis}, \text{wallDis})\]. Here, \text{objDis} denotes the distance from the UAV to
the target, and \text{wallDis} denotes the distance from the UAV to the closest
obstacle. Assume that these distances can be monitored in real time by
the UAV’s equipped sensors. During the path planning process, different
methods will be applied depending on the state of UAV. If the UAV is
far from any obstacles, a trained Neural Network (**NN**) will be applied
to guide the UAV towards the target along a smooth trajectory. Con-
versely, if the UAV is close to one or more obstacles, then the value of
\text{step} (representing the UAV’s velocity) will be decreased to avoid colli-
sion. In addition, stochastic methods such as QS-RRT or RRT [27, 28]
will be employed to ensure that the UAV will escape from the vicinity of
the obstacles in time. In the faulty version of the function, the statement
\text{s5} is faulty because it sets the variable \text{step} to an incorrect value when
\text{wallDis} < 8.

Suppose that we have five test cases, whose execution information and labels
are shown in Fig. 2. The labels of the first three test cases, \text{t1}, \text{t2}, and \text{t3},
are already known. Among them, \text{t2} and \text{t3} are passing test cases, under the
condition that \text{wallDis} > 8, whereas \text{t1} is a failure-revealing test case, which
covers a large number of statements and consequently cannot clearly expose
the faulty statement. If we were to implement SBFL based only on \text{t1}, \text{t2}, and
\text{t3}, we would not be able to distinguish the statements within the predicate
\text{wallDis} > 8; that is, all statements from \text{s5} to \text{s9} would be assigned the same
suspiciousness value.

However, we also have two more unlabelled test cases, \text{t4} and \text{t5}, with differ-
ent execution traces. Intuitively, if these two test cases can be properly utilized,
then the coverage diversity will be enriched and the precision of fault localization
can be improved.
From \( t_1, t_2, \) and \( t_3, \) we can observe distinct behaviours between the passing and failing test cases. Specifically, the failing test case \( t_1 \) covers all statements in the block corresponding to \( \text{wallDis} < 8 \), whereas the passing test cases \( t_2 \) and \( t_3 \) cover the statements in the block corresponding to \( \text{wallDis} > 8 \) and \( \text{objDis} < 5 \). Meanwhile, for \( t_4 \) and \( t_5, \) the coverage of their traces are most similar to that of \( t_1 \). Specifically, \( t_4 \) covers the same statements as \( t_1 \), whereas the coverage differs between \( t_5 \) and \( t_1 \) for only three statements. Therefore, these test cases probably cover the faulty statement as \( t_1 \) does, and we can assign them estimated labels of failing. Then, once \( t_4 \) and \( t_5 \) are considered along with their estimated labels, the suspiciousness value of the faulty statement \( s_5 \) will be higher. As shown, the utilization of unlabelled test cases is a feasible means of improving the effectiveness of fault localization.
3. Utilizing Unlabelled Test Cases Based on SBFL

3.1. Problem Description

Given a program of interest, \( PG \), and the corresponding set of \( n \) executable statements \( S = \{ s_1, s_2, \ldots, s_n \} \), let \( T_e \) denote the set of all executed test cases. For each test case \( t \in T_e \), let \( C^t \) denote its coverage, defined as the set of statements covered by \( t \), i.e., \( C^t = \{ s \mid \text{statement } s \text{ is covered by test case } t \} \), and let \( o^t \) denote its label. Here, \( o^t = \text{passing} \) indicates that the output of \( t \) is correct, whereas \( o^t = \text{failing} \) indicates that \( t \) reveals a failure. Otherwise, we let \( o^t = \text{null} \) if the label of \( t \) cannot be identified. Furthermore, suppose that an oracle for \( PG \) is absent, such that \( T_e \) can be divided into two subsets, \( T_l \) and \( T_u \). Here, \( T_l \) represents the set of test cases whose labels are already known, i.e., \( T_l = \{ t \in T_e | o^t = \text{passing} \lor o^t = \text{failing} \} \) and \( T_u \) represents the set of unlabelled test cases, i.e., \( T_u = \{ t \in T_e | o^t = \text{null} \} \). Moreover, we assume that \( |T_u| \gg |T_l| \).

Suppose that the testers use SBFL to perform fault localization. As described in Section 2.1, the conventional SBFL process can be represented by the function

\[
L = SBFL(T),
\]

where the input variable \( T \) denotes the adopted test suite and the output \( L \) is the suspiciousness list determined as the result of fault localization. Specifically, \( L \) is defined as

\[
L = [\lambda^L_1 \lambda^L_2 \cdots \lambda^L_n],
\]

where

\[
\lambda^L_k = (s^L_k, r^L_k),
\]

\( s^L_k \) : the statement whose rank in \( L \) is \( k \),

\( r^L_k \) : the suspiciousness value of \( s^L_k \).

In Eq. (3), \( \lambda^L_k \) denotes the \( k \)th item in \( L \), and \( s^L_k \) and \( r^L_k \) represent the related statement and its suspiciousness value, respectively. The effectiveness of \( L \) is...
measured by the expense $E^L$, which is defined as the percentage of statements in $L$ that testers must examine before finding the faulty statement $s^f$ [21], i.e.,

$$E^L = \left( \frac{i}{|L|} \right) \times 100\%, \quad s^L_i = s^f.$$  (4)

For conventional SBFL, only labelled test cases, i.e., those test cases belonging to $T_l$, can be utilized. Therefore, the actual process is represented by

$$L_l = SBFL(T_l).$$  (5)

Because the useful information contained in $T_l$ is limited, the corresponding fault localization result $L_l$ may be inaccurate.

The problem at hand is to determine how the unlabelled test cases in $T_u$ can be utilized to improve the effectiveness of fault localization. Specifically, we aim to modify the conventional SBFL process, $L = SBFL(T)$, and introduce a new process represented by the function

$$L = SBFL(T_1, T_2).$$  (6)

Unlike conventional SBFL, the process in Eq. (6) takes two test suites, $T_1$ and $T_2$, as inputs. The output is again a suspiciousness list, $L$, with the same form presented in Eq. (3), but the suspiciousness value $r^s$ for each statement $s$ is determined based on the information contained in both $T_1$ and $T_2$. Furthermore, we impose the restriction that we can use both the execution information and the label information for the test cases in $T_1$, but we can use only the execution information for the test cases in $T_2$. In this way, both the labelled test suite $T_l$ and the unlabelled test suite $T_u$ can be used. Here, we let $T_1 = T_l$ and $T_2 = T_u$. Then, the actual process is represented by

$$L_{l,u} = SBFL(T_l, T_u).$$  (7)

where $L_{l,u}$ denotes the output suspiciousness list derived based on both $T_l$ and $T_u$. Generally, solving the problem of how to utilize the unlabelled test cases is equivalent to designing a process $L_{l,u} = SBFL(T_l, T_u)$ that performs better
than conventional SBFL, represented by $L_l = SBFL(T_l)$. We summarize the
problem description as follows.

Assumptions. The entire test suite $T_e$ is divided into a labelled test suite $T_l$ and an unlabelled test suite $T_u$, which satisfy $|T_u| \gg |T_l|$. The testers use SBFL to perform fault localization.

Problem. Modify the process from $L_l = SBFL(T_l)$ to $L_{l,u} = SBFL(T_l, T_u)$ such that $E_{L_{l,u}} < E_{L_l}$.

3.2. Solution Framework

The framework of our approach is depicted in Fig. 3. Our approach, enclosed by the blue dashed line, takes as input the program of interest, $PG$, and the entire executed test suite, $T_e$, including the labelled test suite $T_l$ and the unlabelled test suite $T_u$. Similar to the case of conventional SBFL, the output $L_{l,u}$ is a suspiciousness list of statements that is consistent with Eq. (3). Specifically, our approach consists of two main procedures: test classification and suspiciousness calculation.
The core idea of our approach is test classification. As in a traditional classification problem, we have a set of categories, and each category has its own label. Let $Y$ denote the set of labels. In addition, we have a set of unlabelled samples, $X$. The goal is to identify each unlabelled sample $x \in X$; that is, we wish to find a discriminant function, $h : X \rightarrow Y$, to map the set of unlabelled samples, $X$, to the set of labels, $Y$. Here, the discriminant function $h$ should be derived through a training process based on the samples whose labels are already known. These labelled samples are usually derived from historical data or in accordance with domain knowledge.

For the test classification procedure, the set of labels $Y$ corresponds to the set $\{\text{passing, failing}\}$, and the unlabelled set $X$ corresponds to $T_u$. The discriminant function is a map from $T_u$ to $\{\text{passing, failing}\}$, i.e., $h : T_u \rightarrow \{\text{passing, failing}\}$. It should be trained based on the already labelled test suite $T_l$. Let $T'_l$ denote the output of the classification process, which produces a set of newly labelled test cases, represented by

$$
T'_l = \{t | t \in T_u, h(o^t) = \text{passing} | h(o^t) = \text{failing}\}.
$$

Here, $h(o^t)$ denotes the estimated label of $t$ that is assigned through classification. In SBFL, a test case must be labelled if it is to be used in the calculation of statement suspiciousness. Therefore, test classification is a prerequisite for the utilization of unlabelled test cases.

After classification, the suspiciousness calculation procedure is conducted, following the suitable organization of the test classification results. This procedure takes as inputs both the previously labelled test suite $T_l$ and the newly labelled test suite $T'_l$, and the output is the final suspiciousness list, denoted by $L_{l,u}$. In general, our approach to designing the process $L_{l,u} = SBFL(T_l, T_u)$ is summarized as follows:

1) Design a classifier to estimate the labels of the test cases in $T_u$ and thus to obtain a newly labelled test suite, $T'_l$.
2) Find a method of organizing the test classification results to facilitate the generation of the final suspiciousness list, $L_{l,u}$.
3.3. The Suspiciousness Probability-based Classifier

In this section, the classifier for test classification is designed. In general, for a classification process, several related features from the domain of discourse should be selected to construct a feature space, and all considered samples should be quantified in that feature space. Training a classifier means finding a score function to measure the similarity between each sample and each category. Finally, the discriminant function is applied to assign an estimated label to each target sample according to these similarity values.

For the test classification procedure, in this paper, the coverage of each statement is selected as a feature. Specifically, a test case $t$ is quantified by its coverage $C^t$. Then, the similarity between $t$ and each category, corresponding to a label of passing or failing, is measured based on $C^t$. This similarity measurement is trained based on the already labelled test suite $T_l$. Suppose that test case $t$ covers statement $s$, then the relationship between $s \in C^t$ and $o^t = failing$ (or $o^t = passing$) can be determined based on the following two considerations:

1) Does $s$ contain a fault?
2) If it does, will a program failure occur?

The first consideration is exactly what we wish to verify in fault localization. It is commonly known that the relation between the execution of faulty components and the production of program failures is complex. SBFL itself is a well-studied approach concerning this relation. During fault propagation, the faulty statement acts as the source defect, and once it is triggered, the program will be infected. Then the infection, will propagate to the output through a series of related statements. A well-designed suspiciousness metric for SBFL is also a rational expression of this process. Therefore, the rationale used for the development of suspiciousness metrics for SBFL can also be applied for the design of test classifiers.

The suspiciousness list that is produced by conventional SBFL based on $T_l$ may not be precise. This shortcoming is exactly what we intend to improve
upon in our approach. Nevertheless, this list can indeed provide an estimate of the likelihood that each program statement contains a fault. Because of the rationality of SBFL, this estimate is fairly reasonable when only the training data set $T_l$ is considered. Thus, if we wish to address the first consideration based only on $T_l$, conventional SBFL can be adopted.

The rough suspiciousness list $L_l$ that is derived through conventional SBFL, $L_l = SBFL(T_l)$, can be expressed as follows:

$$L_l = [\lambda_1^{L_l}, \lambda_2^{L_l}, \ldots, \lambda_n^{L_l}],$$

where

$$\lambda_k^{L_l} = \langle s_k^{L_l}, r_k^{L_l} \rangle,$$

$s_k^{L_l}$: the statement that is ranked $k$th,

$$r_k^{L_l} = R_{Ochiai}(s_k^{L_l}, T_l),$$

$$= \frac{a_{sf}^{k}n_{T_l}^{k}}{\sqrt{(a_{sf}^{k}n_{T_l}^{k} + a_{sf}^{k}n_{T_l}^{k}) (a_{sf}^{k}n_{T_l}^{k} + a_{sf}^{k}n_{T_l}^{k})}},$$

(10)

Then, we normalize the suspiciousness values in $L_l$ to the range of $[0, 1]$. Let $\overline{L_l}$ be a normalization of $L_l$ that is derived by recalculating each $r_k^{\overline{L_l}}$ as follows:

$$r_k^{\overline{L_l}} = r_k^{L_l} / \sum_{i=1}^{n} r_i^{L_l}.$$  

(11)

Now, for each statement $s$, we find its rank in $\overline{L_l}$. Suppose that $s$ is ranked $k$th in $\overline{L_l}$. We have

$$r_s^{\overline{T_l}} = r_k^{\overline{L_l}}.$$  

(12)

Suppose that the adopted suspiciousness metric for SBFL is competent and does reflect the likelihood of fault-proneness; then, the approximate probability that $s$ contains a fault can be estimated as

$$\hat{p}(s = s^f | T_l) = r_s^{\overline{T_l}},$$

(13)

where the equation $s = s^f$ indicates that $s$ is a faulty statement.
The second consideration when designing a classifier concerns the failure rate when the program executes a faulty statement. Unlike the first consideration, the failure rate can be simply estimated by calculating the conditional probabilities based on $T_l$. Under the assumption of $s = s^f$, the failure rate for test case $t$ can be calculated as follows:

$$p(o^t = \text{failing} | s = s^f) = \begin{cases} q_{e}^{s,T_l}, & s \in C^t \\ 0, & s \notin C^t \end{cases} \quad (14)$$

where

$$q_{e}^{s,T_l} = \frac{a_{e,T_l}^s}{a_{e,T_l}^s + a_{e,p,T_l}^s}.$$

Based on Eq. (13) and Eq. (14), the classifier can be trained. Then, given a test case $t$, the approximate failure probability can be estimated using the following total probability formula:

$$\hat{p}(o^t = \text{failing}) = \sum_{s \in C^t} \hat{p}(s = s^f | T_l) p(o^t = \text{failing} | s = s^f). \quad (15)$$

If the value of $\hat{p}(o^t = \text{failing})$ is large, it is more likely that $t$ will reveal a failure. Thus, $\hat{p}(o^t = \text{failing})$ can be treated as the similarity between test case $t$ and the failing category, whereas the similarity between $t$ and the passing category can be expressed as $1 - \hat{p}(o^t = \text{failing})$. Then, the final discriminant function is formulated as follows:

$$h(o^t) = \begin{cases} \text{failing} & \hat{p}(o^t = \text{failing}) > \theta_f \\ \text{passing} & \hat{p}(o^t = \text{failing}) < \theta_p \\ \text{null} & \text{otherwise} \end{cases} \quad (16)$$

Here, the thresholds $\theta_f$ and $\theta_p$ should be determined by balancing the benefit of identifying the label of an unlabelled sample with the risk of committing classification errors. If we set $\theta_f$ to a larger value and $\theta_p$ to a smaller value, the degree of risk of classification errors will be lower, but the ability to label unlabelled samples will also decrease. In the experiments reported in this paper, in the absence of any domain knowledge, $\theta_f$ and $\theta_p$ are set to 0.5. We reserve the optimization of $\theta_f$ and $\theta_p$ for our future work regarding the proposed approach.
Generally, as the core idea underlying the design of the classifier, the rough suspiciousness list that is derived based on $T_l$ is used to estimate the approximate probabilities of fault-proneness. In other words, the classifier is trained based on the domain knowledge of SBFL itself. Therefore, the classifier proposed in this paper is called the Suspiciousness Probability-based (SP) classifier.

Regarding the time complexity of the test classification process based on the SP classifier, the training process formulated as shown in Eq. (9) – Eq. (14) has the same complexity as that of the conventional SBFL. In addition, during the identification of the output for each test case $t$, Eq. (14) – Eq. (16) should be applied, and the coverage of each statement $s$ should be examined. Therefore, the total time complexity for classification is proportional to the size of $T_u$ and the size of $S$, i.e., $O(|T_u| \cdot |S|)$.

3.4. Organizing Classification Results for Use in the Suspiciousness Calculation

After classification, a newly labelled test suite $T_l'$ is obtained. In this section, we discuss the utilization of the test cases in $T_l'$ for the calculation of statement suspiciousness.

In the conventional SBFL process, the labels of the test cases are used only in the calculation of the statistical quantities. Specifically, based on $T_l$, the statistical quantities for each statement $s$ are $a_{nf}^{s,T_l}$, $a_{cp}^{s,T_l}$, $a_{ef}^{s,T_l}$, and $a_{np}^{s,T_l}$, and the overall statistical quantities are $F^{T_l}$ and $P^{T_l}$. They can be calculated as follows:

$$a_{ef}^{s,T_l} = |\{t \in T_l| o^t = failing, s \in C^t\}|$$

$$a_{ep}^{s,T_l} = |\{t \in T_l| o^t = passing, s \in C^t\}|$$

$$a_{nf}^{s,T_l} = |\{t \in T_l| o^t = failing, s \notin C^t\}|$$

$$a_{np}^{s,T_l} = |\{t \in T_l| o^t = passing, s \notin C^t\}|$$

$$F^{T_l} = |\{t \in T_l| o^t = failing\}|$$

$$P^{T_l} = |\{t \in T_l| o^t = passing\}|.$$

(17)
From Eq. (1), once the suspiciousness metric $R$ has been selected, how these statistical values will be plugged into the calculation of statement suspiciousness is also determined. Thus, the newly labelled test suite $T'_l$ is used in the suspiciousness calculation procedure by including $T'_l$ in the calculation of these statistical quantities. Specifically, we calculate the modified statistical quantities based on both $T_l$ and $T'_l$ as follows:

$$
a_{s_f,T_l,\alpha T'_l} = a_{s_f,T_l} + \alpha \{ t \in T'_l \mid h(o') = \text{failing}, s \in C^i \}$$
$$a_{s_p,T_l,\alpha T'_l} = a_{s_p,T_l} + \alpha \{ t \in T'_l \mid h(o') = \text{passing}, s \in C^i \}$$
$$a_{s_nf,T_l,\alpha T'_l} = a_{s_nf,T_l} + \alpha \{ t \in T'_l \mid h(o') = \text{failing}, s \notin C^i \}$$
$$a_{s_np,T_l,\alpha T'_l} = a_{s_np,T_l} + \alpha \{ t \in T'_l \mid h(o') = \text{passing}, s \notin C^i \}$$

$$F_{T_l,\alpha T'_l} = F_{T_l} + \alpha \{ t \in T'_l \mid h(o') = \text{failing} \}$$
$$P_{T_l,\alpha T'_l} = P_{T_l} + \alpha \{ t \in T'_l \mid h(o') = \text{passing} \}$$

(18)

Then, we use the statistical quantities calculated using Eq. (18) instead of the original ones to calculate the suspiciousness metric. Based on the Ochiai metric, the suspiciousness value for each statement $s$ can be calculated as follows:

$$R_{\text{Ochiai}}(s,T_l,T'_l) = \frac{a_{s,T_l,\alpha T'_l}}{\sqrt{(a_{s_f,T_l,\alpha T'_l} + a_{s_nf,T_l,\alpha T'_l})(a_{s_p,T_l,\alpha T'_l} + a_{s_np,T_l,\alpha T'_l})}}.$$  

(19)

Finally, we rank the statements according to their suspiciousness values as calculated using Eq. (19) to obtain the suspiciousness list $L_{l,u}$.

Unlike Eq. (17), Eq. (18) considers the test cases in $T'_l$ but assigns them a weight of $\alpha$. Because the test classification results may contain errors, in the calculation of statement suspiciousness, the test cases in $T'_l$ should be assigned lower weight values than those in $T_l$. In this paper, we set the value of $\alpha$ sufficiently small that for any statement $s$, we have

$$R(s_1,T_l) > R(s_2,T_l) \rightarrow R'(s_1,T_l,T'_l) > R'(s_2,T_l,T'_l),$$

(20)

which indicates that for any two statements, the qualitative relation between...
their suspiciousness values will not change if they can already be distinguished in $L_l$.

The purpose of this setting is mainly to ensure the dominant position of the test cases in $T_l$. Nevertheless, although the weight is small, the utilization of the unlabelled test suite can also exert a beneficial effect. If the information provided by the labelled test cases is not sufficient, many non-faulty statements will have the same suspiciousness values as those of a faulty statement, leading to a long tie interval in $L_l$ [21]. The consideration of $T'_l$ can distinguish the statements in this tie interval even when the weight value $\alpha$ is small. Thus, such utilization of $T'_l$ can help to refine the suspiciousness list while preventing the occurrence of detrimental effects due to classification errors.

4. Empirical Study

4.1. Research Questions

In this section, we report empirical results and evaluate the usefulness of solving the problem of how to utilize the unlabelled test cases in SBFL. In addition, several characteristics of our approach are further explored. Specifically, the following research questions are discussed.

**RQ1.** Can the effectiveness of fault localization be improved by utilizing the unlabelled test cases using our approach?

**RQ2.** How do factors related to the experimental subjects affect the performance of our approach?

**RQ3.** How much computational overhead does our approach introduce?

**RQ4.** What is the relationship between the test classification performance and the effectiveness of fault localization?

**RQ1** focuses on the overall usefulness of utilizing unlabelled test cases in fault localization and the effectiveness of our approach. Furthermore, **RQ2** explores impact factors in search of more explicitly understanding of the performance of our approach. **RQ3** focuses on the efficiency of our approach, which is important for its practicability. Finally, in **RQ4**, we attempt to explore the
Table 1: Characteristics of the programs used in the experiments

<table>
<thead>
<tr>
<th>Suite</th>
<th>Programs</th>
<th>Faulty versions</th>
<th>LOC</th>
<th>Size of test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>flex</td>
<td>53</td>
<td>10,459</td>
<td>567</td>
</tr>
<tr>
<td></td>
<td>grep</td>
<td>17</td>
<td>10,068</td>
<td>809</td>
</tr>
<tr>
<td></td>
<td>gzip</td>
<td>17</td>
<td>5,680</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>sed</td>
<td>26</td>
<td>14,427</td>
<td>370</td>
</tr>
<tr>
<td></td>
<td>print_tokens</td>
<td>5</td>
<td>472</td>
<td>1,608</td>
</tr>
<tr>
<td></td>
<td>print_tokens2</td>
<td>9</td>
<td>399</td>
<td>2,650</td>
</tr>
<tr>
<td></td>
<td>replace</td>
<td>29</td>
<td>512</td>
<td>2,10</td>
</tr>
<tr>
<td>Siemens</td>
<td>teas</td>
<td>41</td>
<td>141</td>
<td>1,052</td>
</tr>
<tr>
<td></td>
<td>schedule</td>
<td>5</td>
<td>292</td>
<td>1,430</td>
</tr>
<tr>
<td></td>
<td>schedule2</td>
<td>9</td>
<td>301</td>
<td>4,115</td>
</tr>
<tr>
<td></td>
<td>tot_info</td>
<td>23</td>
<td>440</td>
<td>5,542</td>
</tr>
<tr>
<td>Space</td>
<td>space</td>
<td>33</td>
<td>6,199</td>
<td>13,585</td>
</tr>
</tbody>
</table>

mechanism by which the test classification results influence the output of SBFL to gain more confidence in the adoption of our approach.

4.2. Experimental Setup

In our experiments, we used three suites of subject programs, i.e., a suite of four UNIX programs, the Siemens suite, and the space program. In total, 12 programs were considered in our empirical study. Among the UNIX programs, `flex` is a well-known program for lexical analysis, `grep` is a command-line utility for searching plain-text data sets for lines that match a regular expression, `gzip` is a file format application used for file compression and decompression, and `sed` is a text editor for batch processing. The `Siemens` suite includes seven programs. It was originally created to support software testing research. The `space` program functions as an interpreter for an array definition language.

Each program has several benchmark faults. These faults were manually
seeded in the UNIX programs and the Siemens suite, whereas they are real ones in the space program. In the study reported in this paper, both single-fault and multi-fault versions were examined. Although real-life programs usually contain multiple faults, it is still meaningful to discuss single-fault scenarios. In practice, if testers focus on a specific fault, they will probably filter out failures caused by other faults. For the single-fault versions, we seeded one faulty statement and eliminated others for each execution. In addition, we also tested multi-fault versions. A multi-fault version was constructed by randomly seeding two or three faulty statements simultaneously for each execution. For two-fault versions, we checked all possible combinations for each pair of faulty statements, whereas for three-fault versions, 50 combinations of faulty statements were randomly constructed for each program. The source code for the subject programs and all benchmark faults were obtained from the Software Infrastructure Repository (SIR) [29]. For each faulty version, SIR provides both the faulty statement and its correct version. During the execution of a specific faulty version, the faulty statements were compiled, and the correct ones were annotated. Table 1 lists the detailed information on each subject program, including the number of benchmark faulty versions, the number of Lines Of Code (LOC), and the size of the provided test suite.

To answer RQ 1-RQ 3, we applied our approach to utilizing unlabelled test cases in fault localization and compared the results with those derived via conventional SBFL based on only the labelled test suite to analyse the overall usefulness of our approach. Here, both single- and multi-fault versions were examined. For each comparison, t-test was applied by setting the p-value to 0.05 to determine which solution could yield the lower expense. Many researchers have recognized that if the expense value achieved with SBFL is too high, testers will turn to manual debugging [5]. In such situations, SBFL may not be a suitable fault localization technique, and discussing the improvement in these situations does not make much sense. Consequently, in our experiments, only instances with expense values lower than 0.1 were considered. Setting threshold expense values during the evaluation of SBFL can also be found in other work.
[30]. Specifically, we first performed a one-side t-test under the alternative hypothesis that the expected expense value was less than 0.1, and eliminated the faulty versions for which the t-test results did not reject the null hypothesis in both conventional SBFL and our approach.

At each faulty version, we divided the given test suite into $T_l$ and $T_u$ following a random selection strategy; that is, we randomly selected a test suite $T_u$ and assumed that the test cases in $T_u$ were unlabelled. The test suites provided by SIR are quite large; therefore, to satisfy the basic assumptions of the proposed problem, i.e., $T_l \ll T_u$, the size ratio of $|T_l|$ to $|T_e|$ was set to a quite low value of 0.05. Ratios of 0.1 and 0.15 were also tested when analysing the influence of the training samples size. Because of the randomness inherent in the generation of $|T_u|$, we repeated 20 times for each faulty version to mitigate the effect of noise.

To answer RQ 4, we independently performed test classification under the assumption that the size ratio of $|T_l|$ to $|T_e|$ was 0.05. To evaluate the test classification performance, the F-measure, which is frequently used in the evaluation of various classification problems, was adopted.

During test classification, failing and passing test cases were treated as “negative” and “positive” samples, respectively. Thus, we could adopt the following definitions.

- True Positives (TP): Number of passing test cases correctly identified as passing.
- False Positives (FP): Number of failing test cases incorrectly identified as passing.
- True Negatives (TN): Number of failing test cases correctly identified as failing.
- False Negatives (FN): Number of passing test cases incorrectly identified as failing.

Based on the above definitions, the Precision and Recall are defined as fol-
lows:

\[ \text{Precision} = \frac{TP}{TP + FP} \]  

(21)

\[ \text{Recall} = \frac{TP}{TP + FN} \]  

(22)

In turn, the F-measure can be calculated as:

\[ \text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \]  

(23)

In addition, we considered a perfect classifier, namely, one that can correctly determine the output correctness for all the unlabelled test cases. This case was considered to further explore the relationship between the quality of test classification and the effectiveness of fault localization.

4.3. Results

In this section, we present the experimental results and a discussion based on them. Table 2 presents the t-test results for the comparison of the fault localization performance with and without utilizing the unlabelled test cases, when the value of \(|T_l|/|T_u|\) was set to 0.05. The first column indicates whether the considered instances contained single or multiple faulty statements, and in the second column, notations “B” and “W” indicate the percentages of faulty versions for which our approach performs better and worse, respectively, than conventional SBFL. Each column from Column 3 to Column 8 displays the comparison results for each program suite. For example, the data presented in Column 4, Row 2, indicate that, for the grep program, the percentage of faulty versions for which our approach outperforms conventional SBFL is 41.1%.

Then, in the ninth column, the total data are presented. Here, we consider our approach to outperform conventional SBFL for a specific faulty version if it can produce a suspiciousness list with a lower expense value for this faulty version according to the t-test results. In addition, Table 3 and Table 4 present the t-test results for the cases in which \(|T_l|/|T_u|\) is equal to 0.1 and 0.15, respectively.
Table 2: The t-test results for our approach with $|T_l|/|T_e| = 0.05$

<table>
<thead>
<tr>
<th></th>
<th>flex</th>
<th>grep</th>
<th>gzip</th>
<th>sed</th>
<th>Siemens</th>
<th>space</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Fault</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(%)</td>
<td>18.8</td>
<td>41.1</td>
<td>53.3</td>
<td>37.5</td>
<td>6.6</td>
<td>10.7</td>
<td>17.4</td>
</tr>
<tr>
<td>W(%)</td>
<td>7.5</td>
<td>11.7</td>
<td>20</td>
<td>16.7</td>
<td>6.6</td>
<td>0</td>
<td>8.1</td>
</tr>
<tr>
<td>Multiple Faults</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(%)</td>
<td>25.9</td>
<td>41.6</td>
<td>58.6</td>
<td>42.3</td>
<td>5.1</td>
<td>10.0</td>
<td>19.7</td>
</tr>
<tr>
<td>W(%)</td>
<td>5.6</td>
<td>15.3</td>
<td>17</td>
<td>6.5</td>
<td>4.2</td>
<td>3.2</td>
<td>6.6</td>
</tr>
</tbody>
</table>

Table 3: The t-test results for our approach with $|T_l|/|T_e| = 0.1$

<table>
<thead>
<tr>
<th></th>
<th>flex</th>
<th>grep</th>
<th>gzip</th>
<th>sed</th>
<th>Siemens</th>
<th>space</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Fault</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(%)</td>
<td>18.8</td>
<td>29.4</td>
<td>53.3</td>
<td>25</td>
<td>5.8</td>
<td>3.6</td>
<td>14.3</td>
</tr>
<tr>
<td>W(%)</td>
<td>9.4</td>
<td>17.7</td>
<td>13.3</td>
<td>8.3</td>
<td>4.1</td>
<td>3.6</td>
<td>6.9</td>
</tr>
<tr>
<td>Multiple Faults</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(%)</td>
<td>22.9</td>
<td>26.8</td>
<td>56.6</td>
<td>34.6</td>
<td>4.3</td>
<td>2.0</td>
<td>15.3</td>
</tr>
<tr>
<td>W(%)</td>
<td>7.8</td>
<td>11.7</td>
<td>14.3</td>
<td>6.1</td>
<td>4.2</td>
<td>1.2</td>
<td>6.2</td>
</tr>
</tbody>
</table>

Table 4: The t-test results for our approach with $|T_l|/|T_e| = 0.15$

<table>
<thead>
<tr>
<th></th>
<th>flex</th>
<th>grep</th>
<th>gzip</th>
<th>sed</th>
<th>Siemens</th>
<th>space</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Fault</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(%)</td>
<td>15.1</td>
<td>17.7</td>
<td>53.3</td>
<td>29.1</td>
<td>2.8</td>
<td>0</td>
<td>11.2</td>
</tr>
<tr>
<td>W(%)</td>
<td>9.4</td>
<td>17.7</td>
<td>13.3</td>
<td>4.2</td>
<td>4.1</td>
<td>3.6</td>
<td>6.5</td>
</tr>
<tr>
<td>Multiple Faults</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(%)</td>
<td>20.1</td>
<td>20.6</td>
<td>56.0</td>
<td>35.4</td>
<td>2.5</td>
<td>0.6</td>
<td>12.0</td>
</tr>
<tr>
<td>W(%)</td>
<td>9.6</td>
<td>18.4</td>
<td>10.4</td>
<td>3.1</td>
<td>4.8</td>
<td>2.1</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Figure 4: Relationships between the number of instances and the expense values derived using different solutions, where the x axis represents the expense value and the y axis represents the number of faulty versions for which the fault localization expense is lower than the value associated with the abscissa. For example, a point \((a, b)\) indicates that there are \(a\) instances with expense values lower than \(b\).
Table 5: Computation times for implementing our approach and conventional SBFL

<table>
<thead>
<tr>
<th>Computation Time(s)</th>
<th>flex</th>
<th>grep</th>
<th>gzip</th>
<th>sed</th>
<th>Siemens</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional SBFL</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.01</td>
<td>0.0</td>
<td>0.05</td>
</tr>
<tr>
<td>Our approach</td>
<td>0.37</td>
<td>0.38</td>
<td>0.08</td>
<td>0.11</td>
<td>0.02</td>
<td>7.06</td>
</tr>
<tr>
<td>Classification</td>
<td>0.33</td>
<td>0.31</td>
<td>0.07</td>
<td>0.10</td>
<td>0.02</td>
<td>6.74</td>
</tr>
</tbody>
</table>

Fig. 4 illustrates the performance of our approach to utilizing unlabelled test cases. Each sub-figure presents the results for a specific program suite, where the x axis represents the expense value and the y axis represents the number of faulty versions for which the expense of fault localization is lower than the value associated with the abscissa. For example, if a curve passes through the point (0.01, 20), it indicates that there are 20 faulty versions with expense values lower than 1%. These plots can be interpreted as the number of faults that can be localized after checking a certain percentage of the code. In each sub-figure, the black solid curve illustrates the performance achieved using our approach, whereas the green dot-dashed curve illustrates the performance of conventional SBFL. In addition, we also assume a “perfect” classifier that can correctly identify the output of all unlabelled test cases; the performance of the “perfect” classifier is illustrated by the cyan dashed curves.

Table 5 presents the computational overheads of the different fault localization procedures. Specifically, the second and third rows record the computation times required to implement conventional SBFL and our approach, respectively. We also record the computation time for test classification, in the fourth row. Each column presents the data for a specific program suite.

Fig. 5 shows the effectiveness of test classification in terms of the F-measure values. The results for both the single- and multi-fault versions are calculated together. The x axis represents the ranges of F-measure values from (0, 0.1] to (0.9, 1], whereas the y axis represents the percentage of instances that fall into each range.
Figure 5: The distributions of F-measure values for test classification.
4.4. Analyses

In this section, we analyse the experimental results and answer research questions RQ1-RQ4.

4.4.1. Answer to RQ1:

As seen from Table 2, for the single-fault program versions, our approach performs better than conventional SBFL for 17.4% (45) of the faulty versions, more than twice the percentage of faulty versions for which our approach performs worse (which is 8.1%). For the multi-fault versions, the advantage of our approach is more obvious. For 19.7% (790) of the faulty versions, our approach performs better, whereas it performs worse for only 6.6% (272). The number of faulty versions for which the performance of our approach is better is nearly three times that for which it is worse. Although, for some instances, the better percentage is equal to or even much smaller than the worse percentage (e.g., Siemens programs in single-fault versions and space programs when $|T_f|/|T_e| = 0.15$), the overall results obviously demonstrate the advantage of applying our approach. Generally, the number of faulty versions for which our approach is beneficial for fault localization is much larger than the number of faulty versions for which our approach has a detrimental effect. Considering all instances with different relative sizes of $T_f$, there are 2025 in which our approach performs better, far more than the 882 instances in which it performs worse.

Moreover, from Fig. 4, we can intuitively observe that for most programs, the curves representing our approach (i.e., the black solid curves) lie above those representing conventional SBFL (i.e., the green dot-dashed curves). This means that, given a specific limitation in terms of fault localization expense, our approach can allow the faulty statements to be found in more instances than when conventional SBFL is applied.

Generally speaking, compared with conventional SBFL, in which only labelled test cases are considered, our approach can indeed improve the effectiveness of fault localization. That is to say, the usefulness of properly utilizing unlabelled test cases in fault localization has been demonstrated.
4.4.2. Answer to RQ2:

To answer research question RQ2, we explore the influences of factors related to the experimental subjects on the performance of our approach, including different numbers of labelled test cases, different subject programs, and different numbers of faults.

The proportion of labelled test cases.

A comparison of Table 2 – Table 4 for the different programs reveals different influences depending on the values of $|T_l|/|T_e|$. For example, as $|T_l|/|T_e|$ increases from 0.05 to 0.1, our approach shows better performance for the sed and gzip programs and similar performance for the flex and Siemens programs but worse performance for the grep and space programs. Then when $|T_l|/|T_e|$ further increase from 0.1 to 0.15, our approach performs much better for the sed program and worse for the other programs.

The results indicate that the relationship between the performance of our approach and the size of the “training” samples is quite complex. On the one hand, if the information contained in $T_l$ is insufficient to support effective fault localization, the performance of our approach will improve as the size of $T_l$ increases because we can better utilize the unlabelled test cases with a more competent training set. On the other hand, for some instances, if the fault localization effectiveness has already reached a bottleneck solely on the labelled test cases in $T_l$, our approach cannot exert a beneficial effect. In fact, such situations do not satisfy one of the basic assumptions of the problem considered in this paper, i.e., that the information provided by the labelled test cases is insufficient. For some of the programs considered in this experiment, the labelled test suites are quite large even though they account for only a small proportion of the provided test suites. Nevertheless, in practice, if test oracles are absent, it is very likely that the labelled test suites will be unable to support automatic fault localization, and our approach will be useful. Of course, the question of how to identify the sufficiency of a labelled test suite needs to be further explored. In fact, relevant studies concerning different coverage criteria have
been conducted by other researchers [19, 30], and there is potential to combine these studies with our approach.

**Different programs.**

From Table 2, we can observe that our approach shows distinct performances for different programs. For the `flex`, `gzip`, and `grep` programs, the improvements achieved using our approach are obvious. The number of instances for which our approach performs better is two times greater than the number for which it performs worse. By contrast, for the `Siemens` programs, the single-fault versions, our approach performs better for 8 instances and worse for 8 instances. Such phenomenon can also be found in the `space` program. Although, for these programs, our approach cannot achieve obvious improvements. Nevertheless, from Fig. 4, we can observe that the performances of our approach and conventional SBFL are close for the `Siemens` and `space` programs, which indicates that our approach also introduce few detrimental effects.

**Single- and multi-fault scenarios.**

Examination of the single- and multi-fault scenarios in Table 2 reveals that our approach shows better performance on multi-fault scenarios than on single-fault scenarios. In total, among the multi-fault versions, the faulty versions for which our approach performs better are at least three times more numerous than those for which our approach performs worse, whereas in the single-fault scenarios, our approach achieve better performance for only twice as many faulty versions as the number for which it performs worse. We infer that for single-fault cases, the faulty statements are more easily to be exposed, meaning that the already labelled test cases are more likely to be sufficient. In most real-life situations, a program will contain multiple faulty statements, which support the practicability of our approach.

From the above discussion of **RQ2**, we find that the performance of our approach may be influenced by several factors related to the subject of an experiment. In most instances, the effectiveness of fault localization using our approach shows an overall improvement under various settings. In addition, we observe that our approach exerts few detrimental effects even when its per-
formance is influenced by these factors. The discussion also demonstrates the
importance of combining our approach with other current fault localization tech-
niques to cope with these influences.

4.4.3. Answer to RQ3:

From Table 5, we can observe that our approach requires more computa-
tion time than conventional SBFL does, and the test classification procedure
consumes the most computation time. However, in practice, SBFL is a only
semi-automatic fault localization technique, meaning that its output must be
further checked by testers. Thus, compared with the overhead of the fault lo-
calization process as a whole, the additional computation time required by our
approach is not much of a cost. For example, for the space program, our ap-
proach requires less than ten seconds to classify more than ten thousands of test
cases. Therefore, the computation time required to implement our approach is
acceptable.

4.4.4. Answer to RQ4:

The effectiveness of test classification using the SP classifier designed in this
paper is shown in Fig. 5. Over 90% of instances have F-measure values over 0.95.
Specifically, for the grep and gzip programs, the percentages of instances with
F-measure values larger than 0.95 are greater than 80%. For the flex programs,
approximately 65% of instances have F-measure values greater than 0.95; this
percentage is markedly smaller than those for the grep and gzip programs. For
the sed program, most instances have F-measure values over 0.6, and more than
20% of instances have F-measure values between 0.5 and 0.7. For the Siemens
and space programs, nearly all instances have the F-measure values larger than
0.95. When checking the details of the classification process, we find that, for
these two program suites, the quantities of passing test cases are much larger
than those of failure-revealing test cases, which causes the SP classifier to assign
all the estimated labels as passing. Nevertheless, in these faulty versions, the
quantity of passing test cases is dominant and the failing test cases are very
few. Therefore, although the failing test cases are difficult to be identified, the classification results will also have high accuracy.

When analysing Fig. 4 and Fig. 5 together, we observe that the accuracy of test classification and the effectiveness of fault localization are indeed positively correlated. First, for the flex, grep, and gzip programs, the classifier yields good results, and our approach also achieve obvious improvements in fault localization. In addition, from the cyan dashed curves in Fig. 4, we can observe that the improvement would be more significant if a perfect classifier could be designed and adopted.

In addition, although the SP classifier does not perform as well for Siemens, sed and space as for flex, grep and gzip, the fault localization effectiveness nevertheless shows improvement. In fact, for the sed program, the improvement is more obvious than for any other program. Meanwhile, for Siemens programs, the improvement in fault localization is not obvious; however, compared with the performance of conventional SBFL, test classification procedure introduces few detrimental effects.

The results can be analysed from two perspectives. At one hand, we find that the SP classifier performs well in most instances and that test classification is indeed a suitable means of utilizing the unlabelled test cases to facilitate fault localization. In addition, an obvious gap remains between the performance of our approach and that of the perfect classifier on improving fault localization. This observation further demonstrates that the improvement of fault localization depends on the classification accuracy. That is to say, the effectiveness of fault localization can be markedly improved if we could provide much more accurate classification results. This observation also indicates that our approach still has considerable room for improvement in the future.

On the other hand, for instances that our classifier does not perform very well, test classification brings few negative effects to fault localization. Considering the reasons, in SBFL, the statement suspiciousness is calculated based on the executions of all the test cases, including the originally labelled test cases and the test cases with correctly estimated labels. Thus, a small inaccuracy of
output correctness could not make obvious changes to the suspiciousness list. The other possible reason is that, in our approach, we assign a small weight value to the classification results for use in fault localization. This assignment further limits the extent to which the basic effectiveness of fault localization can be modified. Note that the large amount of test cases whose labels are correctly estimated can indeed exert a beneficial effect on refining the suspiciousness list.

For example, when the extent of coincidental correctness is severe (e.g., the Siemens programs), much misleading information is included when training the classifier. Consequently, the classifier is very likely to assign a failure-revealing test cases a label of passing. Nevertheless, in these cases, the passing test cases in $T_l$ already contain a great deal misleading information, which could affect both test classification and the original suspiciousness list $L_l$. As a result, the classification inaccuracy does not induce large changes to the suspiciousness list based on $T_l$. For sed, the test cases in $T_l$ are insufficient. Therefore, despite the occurrence of classification inaccuracy, the benefits derived through classification are nevertheless significant.

5. Threats to Validity

One external threat to validity is that we used three groups of subject programs in our experiments to examine the effectiveness of our approach. They range from being small in scale, with hundreds of statements, to quite large in scale, with more than ten thousand statements. These programs have also been used in other studies concerning software fault localization and SBFL [7, 9, 6, 19, 30, 31]. Nevertheless, considering the variety of modern software, additional attempts based on different programs, different programming languages, and various failure patterns should be further explored.

Another important threat pertains to the faulty versions considered. In this paper, all faulty versions are obtained from SIR. The multi-fault versions are constructed based on the combinations of independent faults. These faulty versions are not representative of all the behaviours of real-life faults. More
complex faults and methods of constructing multi-fault program versions should be tested in the future. Additional possible research directions include studying the characteristics of program behaviours under particular faults and designing specialized classifiers.

In addition, in our experiments, we assumed that the labelled test cases, which support the classification process in our approach, were selected randomly. However, for some specific testing strategies, such as boundary testing, the distributions of execution profiles of the selected test cases could be different. Therefore, whether a set of labelled test cases could be used to train accurate classifiers and how to utilize the unlabelled test cases based on various distributions of test information should be further studied.

The main internal threat concerns the proposed SP classifier. In the field of machine learning, there are many other classifiers with solid theoretical foundations for use in various situations. These classifiers, such as SVM and neural-network-based classifiers, also deserve analysis and exploration. In fact, we studied these classifiers during our experiments and found that it is difficult to choose appropriate parameters for them based on an analysis of the domain knowledge of software testing. In addition, more comprehensive analyses of the SP classifier and explorations of the utilization of additional information, such as program specifications, could be pursued.

The second potential internal threat is that only the Ochiai metric was examined; no other SBFL metrics were studied in this work. The Ochiai metric has been observed to exhibit good performance in both single- and multi-fault scenarios [23, 7, 25, 22]. In addition, for Ochiai metric, the suspiciousness values are calculated using a continuous function. Thus, they are suitable to be treated as probability values when training the SP classifier. Other effective suspiciousness metrics, such as Op for single-fault scenarios, are also worth exploring.
6. Related Work

Fault localization is an activity aimed at finding the locations of detected faults [32]. Spectra-Based Fault Localization (SBFL) is a promising technique aimed at locating the fault-like components by utilizing program spectra with different coverage granularity levels (such as statements [23], predicates [33], or a mixture of multiple levels [34]). After necessary information is collected for each program component, different metrics are applied to evaluate each component’s suspiciousness in terms of its possibility of containing faults. One of the earliest metrics developed was the Tarantula metric [35]. More comprehensive metrics have since been proposed, including the Ochiai metric [23], which is based on similarity coefficients. Naish et al. [8] collected many of these metrics and grouped them into several equivalent categories. This work was later extended by theoretical analyses of more than thirty metrics in single-fault scenarios [21]. Based on the analyses, Op1 and Op2 have been identified as maximal metrics.

In addition, because the pool of test information is essential to fault localization and because the quality of the available test suite has a large impact on the effectiveness of fault localization, another focus of research concerns test suites based on SBFL. Yu et al. [36] demonstrated that fault localization effectiveness varies with the test reduction strategy and attempted to find a balance between the testing effort and the effectiveness of fault localization. Yoo et al. [6] first introduced the fault localization prioritization problem and devised a solution method borrowing from the concept of information entropy. Jiang et al. [19, 30] applied several popular test case prioritization strategies to perform fault localization. They found that the adequacy criterion and the percentage of a prioritized test suite that is utilized are important factors affecting the effectiveness of SBFL, and they also found that Random Testing (RT) and Adaptive Random Testing (ART) [37] are effective test strategies. Furthermore, they found that SBFL can be more effective if the test suite satisfies the MC/DC-adequacy criterion.

Orso and Rothermel [1] produced a research travelogue about software test-
ing approaches and techniques, and they noted that the oracle problem has always been a significant challenge. An explicit and comprehensive survey of trends in the study of test oracles has been conducted by Harman et al. [3]. They divided work on test oracles into four categories: specified oracles, derived oracles, implicit oracles and no oracles (coping with the absence of oracles). Our work is related to the domain of coping with the absence of oracles. Another famous approach called Metamorphic Testing (MT) [38], which considers test case generation and selection methods using domain knowledge, is also used to cope with a lack of oracles. Xie et al. [39] studied fault localization issues based on MT. Their work is an application of MT in fault localization based on domain knowledge related to the program of interest.

In the domain of machine learning, classification is a major topic. To date, the underlying concepts of various classification approaches have been borrowed to identify coincidental correctness in fault localization [40, 41, 42, 43]. These studies are based on an entire labelled test suite and focus on finding and addressing passing test cases that have detrimental effects on fault localization. In this paper, we use test classification to support the utilization of unlabelled test cases due to the absence of oracles. This study represents a potential research area involving the adoption of machine learning approaches in different procedures for fault localization.

7. Conclusion and Future Work

This paper discusses the influence of the test oracle problem on software fault localization, considering that if test oracles are absent, most test cases in an executed test suite may be unlabelled, which will decrease the effectiveness of fault localization. Our work attempts to address this problem by enabling proper utilization of the unlabelled test cases during fault localization. Specifically, our approach is based on test classification, in which the test cases whose labels are already known are treated as training data. Then, a classifier is trained to determine the output of the remaining test cases and to assign them
with estimated labels. Subsequently, the newly labelled test cases can be incorporated into the data considered in automated fault localization techniques. As more test cases are considered, the diversity of the test information pool can be improved, and accordingly, we can achieved better fault localization.

Our approach is based on the technique of Spectra-Based Fault Localization (SBFL). As part of the proposed approach, a Suspiciousness Probability-based (SP) classifier is proposed for use in test classification. It utilizes SBFL domain knowledge and treats the normalized suspiciousness values derived through conventional SBFL as approximate probabilities of faulty-proneness during the training process. After classification, the newly labelled test cases are utilized with small weight values. Empirical results show that in most instances, failure to use the unlabelled test cases will lead to a reduction in fault localization effectiveness, whereas when our approach is applied, the effectiveness of fault localization can indeed be improved.

The primary contributions of this paper are summarized as follows.

1) The usefulness of unlabelled test cases in mitigating the influence of an absence of test oracles on fault localization is particularly discussed. We believe that this is an important and practical issue in the “test-find-fix” cycle.

2) The idea of test classification is proposed to allow unlabelled test cases to be used to support fault localization. A Suspiciousness Probability-based classifier is developed and applied, demonstrating the feasibility of the idea and achieving obvious improvements in effectiveness based on SBFL.

3) An approach of utilizing unlabelled test cases based on SBFL is proposed. Issues concerning the effectiveness and performance of the proposed approach are experimentally analysed.

4) Both the design of new fault-localization-oriented classifiers and expanded empirical studies can provide us with new research directions.

In future work, the initial task will be to test more classifiers to obtain more general conclusions. We also wish to design better classifiers for unlabelled test cases, ones that consider more issues regarding test profiles and program specifications. Furthermore, because different faults exhibit different behaviours
and present different characteristics, another promising direction of research would be to identify failure patterns or execution patterns that are specific to different program structures and fault types for use in test classification. In addition, except for fault localization, we will extend our approach to the utilization of unlabelled test cases from various aspects, aiming to pave the way to many activities in software engineering, such as test case generation [44] and self-healing systems [45]. For example, we can improve the feedback procedure of adaptive testing [46] or design a better rejuvenation model for complex systems with aging problems [47].

References


