Abstract

Spectrum-based fault localization technique mainly utilizes testing coverage information to calculate the suspiciousness of each program element to find the faulty element. However, this technique does not fully take consideration of dependences between program elements, thus its capacity for efficient fault localization is limited. This paper combines program slicing with program spectrum technique, and proposes a program slicing spectrum-based software fault localization (PSS-SFL) technique. Firstly, PSS-SFL analyzes dependences between program elements, and deletes some elements unrelated to the failed test outputs; then it builds the program slicing spectrum model and defines a novel suspiciousness metric for each slice element; finally, the faulty element is located according to the suspiciousness metric results. Experimental results show that PSS-SFL can be effective and more precise to locate the fault than program spectrum-based Tarantula technique.

Keywords—Fault localization, program slicing spectrum, program slicing, program spectrum

I. Introduction

A software fault is an incorrect step, process, or data definition in a computer program [1]. Software fault localization is to locate the fault that cause software failure. According to Collofello’s research, in an attempt to reduce the number of delivered errors, it is estimated that most companies spend between 50% and 80% of their software development effort on testing [2], while software fault localization is one of the most complex and difficult tasks during the process of reducing the number of errors. There are two traditional ways to locate software faults: the first way is to insert print statements into the program, and then analyze the suspicious statements according to the print result; the second way is to set a breakpoint at some statement, and then single step to the next statement and determine whether the statement is faulty or not. Both ways above have to be performed manually. When the program is complex and large-scale, it is difficult to fully analyze the fault and the work will be very huge. So these two traditional ways above are not very efficient.

Program slicing technique was firstly proposed by Weiser [3]. It can abstract program’s statements according to a specific criterion, so the fault in the program can be limited to a small region, that is, a relevant slice. Program slicing technique includes static program slicing technique and dynamic program slicing technique. Software fault localization based on static program slicing [4][5] analyzes the data flow and control flow of the program statically to reduce the search domain of faults. However, because the static program slicing is overly conservative, the precision of locating fault is very low. Based on static program slicing technique, dynamic technique [6][7] introduces more precise slicing criterion and the search domain of faults can be further reduced. This paper firstly abstracts the dynamic slicing criterion according to the failed test information, then constructs a fault-related slice to locate the fault.

In recent years, program spectrum-based software fault localization [8][9] was proposed, which can be applicable to the large-scale program and easy to implement. A program spectrum characterizes, or provides a signature of a program behavior [10]. Generally speaking, the collection of program spectrum is very simple, moreover, its storage is easy, so it can be suitable for situations with limited resources. Program spectrum-based software fault localization technique first statistically analyzes program spectrum and computes the probability that each
program element may include faults, commonly known as suspiciousness; then program elements are checked according to the descending order of the suspiciousness until the fault is found. Researches show that even for the lowest quality of programs, only 20% of the program code remains to be investigated to locate the fault with the program spectrum-based technique [9][11]. However, this technique does not fully take consideration of dependences between program elements, thus its capacity for efficient fault localization is limited. This paper combines program slicing with program spectrum technique, and proposes a program slicing spectrum-based software fault localization (PSS-SFL) technique. The main contributions of this paper are as follows:

(1) We delete program elements that have no dependence with the fault program elements by program slicing technique and construct a fault-related slice according to the test information, and then propose a program slicing spectrum-based software fault localization model.

(2) Based on the popular suspiciousness metric technique, we propose a new suspiciousness metric technique by introducing the frequency and contribution of each program element.

(3) We verify the effectiveness of PSS-SFL technique with our experiment and compare PSS-SFL technique with the popular Tarantula technique based on program spectrum.

II. Preliminaries

A. Dynamic program slicing

A dynamic program slice [6][7] consists only of statements that influence the value of a variable for a specific program input. The location, variable and input are referred as the dynamic slicing criterion as Definition 1, then a dynamic program slice is abstracted from the program dependence graph as Definition 2 [6][12].

Definition 1 (Dynamic slicing criterion): Given a program P, P’s variable set V and input set I, a dynamic slicing criterion C of the program P is a triple < s, v, i >., where s ∈ P is a specific statement in P, v ∈ V denotes a subset of the program variables and i ∈ I means a specific input to the program P.

Definition 2 (Program dependence graph): Given a program P, P’s program dependence graph is a two-tuple < N, E >, where N is a set of nodes which denote the statements in P, E is a set of edges which consist of control dependence edges and data dependence edges. A control dependence edge < v_i, v_j > ∈ E shows node v_i is executed depending on the output of the predicate expression at node v_j and a data dependence edge < v_i, v_j > ∈ E indicates that the computation performed at vertex v_i directly depends on the value computed at vertex v_j.

<table>
<thead>
<tr>
<th>Table I. PSS-SFL model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>e_1</td>
</tr>
<tr>
<td>e_2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>e_n</td>
</tr>
</tbody>
</table>

B. Program spectrum-based software fault localization

Recently, many researchers have proposed various fault localization techniques based on program spectrum [8][9][10][11][13][14][15][16]. Generally speaking, these techniques mainly include three steps: (1) collect coverage information of program elements based on test executions; (2) compute the element suspiciousness based on coverage information; (3) locate the fault according to the suspiciousness in a descending order. Program spectrum is usually defined as follows:

Definition 3 (Program spectrum): Given a program P, n is the number of elements in P and T = T_F ∪ T_P is a set of test executions where T_F = {T_1, ..., T_s} is the set of failed test executions and T_P = {T_{s+1}, ..., T_m} is the set of passed ones, then program spectrum is a two dimensional matrix M_{n,m}: ∀b_{i,j} ∈ M_{n,m},

b_{i,j} = \begin{cases} 1 & T_j covered e_i \\ 0 & \text{otherwise} \end{cases}, (1 ≤ i ≤ n, 1 ≤ j ≤ m)

A visual model of program spectrum-based software fault localization is shown in the following table. Rows of the table denote corresponding elements in the program and columns T_1, ..., T_m indicate the number of test executions is m including s failed ones and m - s passed ones. b_{i,j} is the element of program spectrum matrix with value 1 or 0 which indicates whether T_j executed the element e_i or not. The last column is the suspiciousness of its corresponding element. The metric of suspiciousness is usually based on the following assumptions: the more failed tests and the less passed ones that executed the element e_i, the greater likelihood that e_i is faulty, that is, in the table, the larger value of \sum_{i=1}^{s} b_{i,j} and the smaller value of \sum_{i=s+1}^{m} b_{i,j} , the greater value of suspiciousness(e_i). So suspiciousness(e_i) is generally defined as follows [8]:

Definition 4 (Suspiciousness):

\text{suspiciousness}(e_i) = \frac{\text{failed}(e_i)\%}{\text{failed}(e_i)\% + \text{passed}(e_i)\%}

In the definition above, failed(e_i)\% is the percentage of failed tests that executed e_i in total failed tests, and passed(e_i)\% is the percentage of passed tests that executed e_i in total passed tests. In general, different researchers have different definitions to failed(e_i)\% and passed(e_i)\%, but their targets are all to locate
the faulty element more precisely by the computation of suspiciousness($e_i$). This paper defines a new suspiciousness metric based on popular Tarantula's suspiciousness metric by defining a different $\text{failed}(e_i)\%$ and $\text{passed}(e_i)\%$.

III. program slicing spectrum-based software fault localization

A. Program slicing spectrum model

Traditional program spectrum-based fault localization technique considers all elements of a test execution, but the elements are not all related to the test output, so we introduce the program slicing technique to delete the unrelated elements to the fault and improve the precision of fault localization.

To more effectively locate faults, we firstly collect fault related slice according to the failed test executions:

Definition 5 (Fault-related slice): Given a set of test executions $T = T_F \cup T_P$, where $T_F = \{T_1, \ldots, T_n\}$ is the set of failed test executions and $T_P = \{T_{n+1}, \ldots, T_m\}$ is the set of passed ones. Let $\text{Slice}(T_i)$ ($i = 1, \ldots, m$) be the dynamic slice corresponding to the test execution $T_i$, then fault-related slice is an elements set $\text{Slice} = \text{Slice}(T_1) \cup \text{Slice}(T_2) \cup \ldots \cup \text{Slice}(T_m)$.

Based on the above definition, program slicing spectrum is defined as follow:

Definition 6 (Program slicing spectrum): Given a set of test executions $T = T_F \cup T_P$, where $T_F = \{T_1, \ldots, T_n\}$ is a set of failed test execution and $T_P = \{T_{n+1}, \ldots, T_m\}$ is the set of passed ones, let the number of elements in $\text{Slice}$ be $n$, that is $n=|\text{Slice}|$, then program slicing spectrum is a two-dimensional $M_{n,m}$:

\[
M_{n,m} = \left\{ \begin{array}{ll}
f_{i,j} & \text{the frequency that } T_j \text{ executed } e_i \text{ in } \text{Slice} \\
0 & \text{otherwise}
\end{array} \right.
\]

B. suspiciousness model

Program spectrum-based software fault location technique is performed based on the suspiciousness metric. Definition 4 gives a popular suspiciousness model. This paper improves the traditional computing algorithm of $\text{failed}(e_i)\%$ and $\text{passed}(e_i)\%$ by introducing the execution frequency and contribution of each program element. The popular definition of $\text{failed}(e_i)\%$ and $\text{passed}(e_i)\%$ is proposed in Tarantula technique as follows:

\[
\text{failed}(e_i)\% = \frac{a_{11}(e_i)}{a_{11}(e_i) + a_{01}(e_i)} \times 100\% \quad \text{passed}(e_i)\% = \frac{a_{10}(e_i)}{a_{10}(e_i) + a_{00}(e_i)} \times 100\%
\]

In the formula above, $a_{11}(e_i)$ represents the number of failed tests that executed the element $e_i$, $a_{01}(e_i)$ represents the number of failed tests that didn’t execute the element $e_i$, $a_{10}(e_i)$ represents the number of passed tests that executed the element $e_i$, $a_{00}(e_i)$ represents the number of passed tests that didn’t execute the element $e_i$. Obviously, Tarantula technique computes the suspiciousness of each program element by counting the number of failed tests and passed tests that executed the element. When different tests executed the same element, the contribution of the element to each test result is considered to be equal. That is, in Table I, Tarantula technique just considers if $e_i$ was executed by each test and just counts the number of 1 of the line $e_i$. In reality, the number of program elements that each test executed is different and the frequency that each program element is executed is also different, so the contribution of a program element to different test result is unequal. For example, if a test just executed a program element, then the contribution of this program element to the test result is 100%; while this test executed two program elements and each element was executed one time, the contribution of each program element to the test result is 50%. The contribution can be computed from the column of program spectrum matrix. Below is our definition of $\text{failed}(e_i)\%$ and $\text{passed}(e_i)\%$:

\[
\text{failed}(e_i)\% = \frac{\sum_{j=1}^{m} P_{r_j} \times C_{i,j}}{\sum_{j=1}^{m} P_{r_j}} \quad \text{passed}(e_i)\% = \frac{\sum_{j=1}^{m} (1-P_{r_j}) \times C_{i,j}}{\sum_{j=1}^{m} (1-P_{r_j})}
\]

where $C_{i,j} = \frac{f_{i,j}}{\sum_{k=1}^{m} f_{i,k}}$, $P_{r_j} = \{1 \text{ } T_j \text{ is failed} \quad 0 \text{ } T_j \text{ is passed}\}$

In the formula above, $C_{i,j}$ represents the contribution of the program element $e_i$ to the test execution $T_j$ where $f_{i,j}$ is the frequency of $e_i$ that is executed by $T_j$ and details are shown in Definition 6. $P_{r_j}$ with value 1 represents that $T_j$ is a failed test execution, otherwise $P_{r_j}$ is assigned a value 0. Denominators of $\text{failed}(e_i)\%$ and $\text{passed}(e_i)\%$ represent the number of failed test executions and the number of passed ones; their numerators represent the sum of the contributions of program elements to each failed test execution and passed one. These are different from the suspiciousness metric in Tarantula technique.

When the suspiciousness of each program element is obtained, the fault can be located according to the suspiciousness in a descending way.

C. Program slicing spectrum-based software fault localization model

According to the definition of program slicing spectrum and suspiciousness model, a visual model of program slicing spectrum-based software fault localization is shown in Table II. Rows of the table denote corresponding elements in the fault-related slice $\text{Slice}$ and columns $T_1, \ldots, T_m$ indicate $m$ test executions where the number of failed ones is $s$ and passed ones is $m - s$. $f_{i,j}$ ($1 \leq i \leq n, 1 \leq j \leq m$) is the element of program slicing spectrum matrix with value $k$ or 0 which means the
frequency that \( T_j \) executed the element \( e_i \). The last column is the suspiciousness of each corresponding element by the computation in the above section. Obviously, PSS-SFL technique greatly improves traditional program spectrum-based technique. It reduces the scale of the traditional spectrum, introduces the frequencies of program elements and defines contributions for suspiciousness metric.

Program slicing technique firstly abstracts corresponding slicing criterion, and then generates the slice by traversing program dependence graph. PSS-SFL technique generates the fault-related slice according to the failed test executions information, and then constructs program spectrum according to history test executions and finally computing the suspiciousness of each element in the fault-related slice.

### IV. Experiment

In this section, we will verify the effectiveness of PSS-SFL and compare its precision of locating faults with popular Tarantula techniques.

#### A. Object of analysis

Many current researches on software fault localization use siemens suite as their object. Siemens suite consists of seven C programs and each program has several hundred lines of code. Because each program in siemens suite is mainly composed of branch structures, program spectrum-based software fault localization techniques generally have a high precision on these programs and PSS-SFL has not obvious advantages. This section uses a practical JAVA tool program JAVA Hierarchical Slicing and Applications (JHSA) as our object of analysis. JHSA program includes three packages, twenty-six classes and 11201 lines of code in all, which is used to construct a hierarchical system dependence graph for JAVA program, and to compute its hierarchical slice. We totally select 178 faulty versions for this experiment.

#### B. Experiment design and analysis

In this study, experiment data collection and analysis are conducted in four steps: first, collect program elements coverage information by the tool eclemma and count the frequencies of the program elements; second, abstract the fault-related slice by a slicing tool JHSA according to the failed test executions information, and then constructs program spectrum according to history test executions and finally compute suspiciousness as depicted in the above section and locate faults according to the suspiciousness of each program element in a descending order.

To evaluate the effectiveness of PSS-SFL, an efficiency score is used in this experiment. Given \( v \) is the number of versions in which faults have been successfully located and \( s \) is the number of program elements that have been searched for locating faults in each version we evaluate the effectiveness with following formula:

\[
Efficiency = \frac{v}{s}
\]

In the formula above, efficiency means that the number of versions in which faults have been successfully located when searching \( s \) elements in each version. Here, \( s \) can be also referred as search times. In the ideal case, if the value of \( s \) is 1 and the value of \( v \) is the number of total versions, efficiency denotes that this technique can once find faults in all versions and its efficiency is the highest.

#### C. Result and analysis

In this section, we verify the effectiveness of PSS-SFL and compare its precision of locating faults with popular Tarantula techniques with the above efficiency score.

Figure 1 shows the relation between the efficiency of PSS-SFL and search times. In the figure, with the increase of search times, the efficiency reduced. This implies PSS-SFL technique can quickly locate most faults at the beginning, and only a small number of faults need to be searched many times further, so the technique is very effective.

Figure 2 shows the relation between search times and the number of versions and compares PSS-SFL with Tarantula technique. In the figure, when the value of search times
is about 250, the number of versions in which faults have been found with PSS-SFL is nearly 80, while the value with Tarantula is about 60. In this experiment, the number of all versions is 178, so when searching 250 times, PSS-SFL can find faults in nearly half versions and Tarantula about one third. Moreover, Figure 2 shows PSS-SFL can be more effectively and precisely locate faults most of the time than Tarantula technique.

D. Threads to validity

According to the experiment above, PSS-SFL technique can effectively and precisely locate faults. However, there are some threads to the experiment.

First, the scale of the object of analysis is still relatively small. It is a real application program, so, to some extent, it can verify the effectiveness and precision of PSS-SFL. However, it still needs to be verified for the larger scale program. In addition, with the increase of sequential structures in programs, the precision of Tarantula technique will reduce. Although PSS-SFL introduce program slicing technique to reduce this degradation to some extent, but this degradation also exists in the PSS-SFL technique.

Second, if program fault is caused by the omission of some element, we can locate the fault to its prior element close to the position of omission element. Then, we can find the corresponding fault from this prior element, but this needs some manual analysis.

Third, because suspiciousness metric is based on the statistical number of failed test executions and passed test executions, the precision of suspiciousness will be reduced with the increase of the number of faults in a version. This problem exists in traditional program spectrum-based software fault location technique and also exists in PSS-SFL technique.

V. Related work

Because of the importance and difficulty of software fault localization, more and more software engineering researchers pay attention to this field. They proposed various techniques for locating faults such as techniques based on program slicing [3][4][5][17][18][19][20][21] and program spectrum [8][9][10][11][13][14][15][16]. This paper combines the program slicing and program spectrum, and proposes PSS-SFL technique, which can more effectively locate faults.

Most of program slicing based-software fault localization techniques locate faults using set operation between slices. Dicing technique [17] computes the difference between a failed slice and a passed slice to confine the fault to a small region; execution slice [18][19] is some program execution blocks according to some specific inputs, it can be abstracted more effectively than traditional slice, and this slicing technique generally computes the intersection and union between slices to locate faults. The idea of the union is first to compute the union of all passed slices, then to get the difference between a failed slice and the union; the way of the intersection is first to compute the intersection of all passed slices, then to obtain the difference between the intersection and a failed slice. Reniers proposed another difference between a failed execution slice and a passed slice that has a closest distance to the failed slice [20]. In addition to set operations, DeMillo proposed a critical slicing for locating faults based on “statement deletion” mutant operation [21]. PSS-SFL technique is different from these techniques: Although this technique abstracts dynamic program slice for fault-related slice, it constructs a program slicing spectrum model and finally computes the suspiciousness to locate faults. PSS-SFL technique is more suitable for general cases.

Program spectrum includes program frequency spectrum and program hit spectrum. Their difference is that a frequency spectrum introduces the execution frequency of each program element and a hit spectrum only considers if program elements are covered or not. Traditional program spectrum-based software fault localization is mainly based on program hit spectrum. There are usually two methods to improve the effectiveness of program hit spectrum-based software fault localization. One is to improve the suspiciousness model. The common suspiciousness models mainly have Tarantula [8], Jaccard [9], Ochiai [9], etc. In addition, Wong [13] introduced the weight of test cases into the suspiciousness metric to improve the effectiveness. The other is to optimize the test cases by deleting unrelated or redundant test cases [22][23][24]. PSS-SFL technique is different from these techniques: PSS-SFL introduces program slicing technique to delete elements that have no dependence with faults; moreover, PSS-SFL is based on program frequency spectrum and has a different sus-
piciousness model by introducing the contribution of each program element.

VI. Conclusion and future work

This paper proposed an effective fault location technique, PSS-SFL technique. First, this technique abstracts fault-related slice and deletes elements that have no dependencies with faulty elements to improve the precision of locating faults; second, it introduces the execution frequency and the contribution of each program element to improve the traditional suspiciousness metric; finally, an experiment is conducted to verify the effectiveness of this technique.

Although our experiment has verified the effectiveness of PSS-SFL on a practical program, the efficiency will be reduced with the increase of faults in a single program as other software fault localization technique, so our future work will focus on improving the efficiency of locating faults in multi-faults program.

References