SBFL Hybridization

team 10

20246131 Angela Hu
20246119 Keith Chua Dian Xun
20246118 Chun Zhi Heng Davis
20234922 Wenqing Sun

davisczh/ASTProject_SBFL (github.com) ← the link of project

Ⅰ.problem to be solved

Background:

- Existing SBFL techniques may have limitations when applied to a single test dataset, making it difficult to accurately and comprehensively localize faults.
- Different test datasets may produce different coverage results, impacting fault localization effectiveness.

What we try to do:

Our objective is to test whether splitting our test suite for SBFL gives us more accurate fault localisation or not. We use 'Lang-5b', 'Lang-6b', and 'Lang-7b' (three versions for Defects4J) these three as databases to test.

II. Methodology

Tools:

GZoltar:

- A automated fault localization tool
- Utilizes test coverage data and statistical analysis
- Generate fault reports.Generate fault localization reports

Defects4J:

- Provides existing, validated defect datasets for us

Description of implementation:

1. Take an entire D4J dataset ready to run through Gzoltar.
2. Run GZoltar and generate fault localization reports.
3. Check how accurate the fault localisation report is based on the actual faulty programs provided by D4J themselves.

4. Repeat the above steps but for multiple smaller splits of the dataset, making sure to keep the failing tests.
5. Make observations to see if splitting the input is better.

Detail description:

Determining the accuracy of bug localization in the target project: (evalUnifiedSFL.py)

To determine the accuracy of the SBFL report, we compare the report result to the ground truth.
The line that causes failure in the test suite is underlined in the *failing_tests* file. By identifying the line number and comparing it with the SBFL report, we are able to establish the accuracy.

The formula used to determine the accuracy of the SBFL report is the following:

$$accuracy = \frac{suspiciousness\ value}{rank}$$

As you can see, the rank is also included. The presence of another line evaluated as more buggy than the ground truth undermines the accuracy of the whole report, therefore it's important to also consider the ranking.
The SBFL report is a CSV file processed using *pandas*. Our goal is to evaluate the accuracy of the merged report and the report of the whole test set, in order to determine which one is better.

Based on the line number, we search in the table the corresponding suspiciousness value and using the formula above, we calculate the accuracy(Figure 2.1).

```
44    buggy_rank = float(ranking.loc[ranking['name'].str.contains(buggy_line), 'rank'].values[0])
45    buggy_sus = float(ranking.loc[ranking['name'].str.contains(buggy_line), 'suspiciousness_value'].values[0])
46    accuracy = buggy_sus/buggy_rank
47
48    print(accuracy)
49
50    accuracies = {}
51    accuracies[project_name] = accuracy
52
53    print(accuracies)
```

Figure 2.1 The calculation for accuracy

Use GZoltar to perform tests and collect coverage data:(run.sh)

Using GZoltar's CLI tool, list all test methods from the specified test class file directory and output this information to a file. Set the classpath for Java. Use GZoltar's Java agent to run the tests while collecting test coverage data. This data is stored into a serialized file. Through GZoltar's CLI tool, specify the test method file and necessary parameters such as classes and methods to include and exclude. Finally, using the collected coverage data, the GZoltar tool is

invoked to generate defect localization reports that use specified formulas to evaluate the suspiciousness of each line of code and output to a text file.(Figure 2.2)

```
ser_file="$work_dir/$PID-${BID}b/gzoltar.ser"
java -XX:MaxPermSize=4096M -javaagent:$GZOLTAR_AGENT_JAR=destfile=$ser_file,buildlocation=$src_classes_dir,includes=$classes_to_debug,excludes="",inclnolocationclasses=false,output="FILE" \
  -cp "$src_classes_dir:$D4J_HOME/framework/projects/lib/junit-4.11.jar:$test_classpath:$GZOLTAR_CLI_JAR" \
  com.gzoltar.cli.Main runTestMethods \
    --testMethods "$unit_tests_file" \
    --collectCoverage
```

Figure 2.2 Using GZoltar to run test methods and collect coverage data

Final result:zhiqianzuoy

The original suspiciousness value and the average suspiciousness value after different test segments were calculated. The results(Figure 2.3) show that the suspicion of the originally identified buggy line is improved after some test segmentation, suggesting that the segmentation strategy may help locate the error more accurately.

```
{0: 0.5773502691896258, 1: 1.0, 2: 0.5773502691896258, 3: 1.0}
Original sus: 0.5773502691896258
Average split sus: 0.859116756396542
Splitting finds buggy line more suspicious
```

figure 2.3

III. Experimental Comparison

Experimental Setup

-Environment Configuration:

Set up Defects4J and GZoltar environment variables to ensure the tools function correctly.

-Project Checkout and Compilation:

Use the Defects4J framework to check out the Lang project with the specified bug ID and compile the project.

-Metadata Collection:

Collect necessary metadata, including test classpath, source classes directory, and test classes directory, to facilitate the subsequent steps.

-Listing Test Methods:

Use GZoltar to list all test methods and store them in a file named unit_tests.txt.

Before and After Comparison

Before Implementation:

One program can just run one set of test cases and get one spectrum report by GZoltar.

After Implementation:

After our project,we can reach the conclusion that splitting our test suite for SBFL gives us more accurate fault localisation. In the case that splitting does indeed work better, we can then use coincidental correctness to justify that result.

Ⅳ.Conclusion:

Summary for project:

This project explored the effectiveness of splitting the test suite for Spectrum-Based Fault Localization (SBFL) using GZoltar and Defects4J. Our investigation confirmed that segmenting the test suite improves the accuracy of fault localization. This method provides a more precise identification of buggy lines. Our results indicate that test suite segmentation could be a viable strategy to enhance fault localization techniques in software testing.

Benefit for future:

This method is able to tell us splitting the test suite gives us better fault localisation results, and check against the possibility of coincidental correctness.