Fine-tuning Spectrum Based Fault Localisation with Sequence Mining

Gulsher Laghari, Alessandro Murgia, and Serge Demeyer
ANSYMO — Universiteit Antwerpen België
{gulsher.laghari, alessandro.murgia, serge.demeyer}@uantwerpen.be

***Abstract*—Spectrum based fault localisation techniques merely use the coverage of the program elements to localise the fault. These techniques ignore the dependency relationships between the elements and, hence, come at the cost of their limited diagnostic accuracy. In this paper we propose a variant of spectrum based fault localisation, which leverages sequences of method calls by means of sequence mining. We demonstrate the effectiveness of the variant (we refer to it as sequenced spectrum analysis) by putting it in comparison to the state of the art with a case study on two open source java projects.**

I. Introduction

Spectrum based fault localisation [1], also known as *coverage based fault localisation*, is a lightweight automated technique to locate the faults. It localises the faults by comparing traces form failing and passing test cases. The technique monitors each element under test during program execution and calculates the hit-spectrum (sometimes also called coverage spectrum) of a program. The hit-spectrum of an element under test is tuple of four values (e_f , e_p, n_f , n_p). Where e_f and e_p are the numbers of failing and passing test cases that execute the element under test and n_f and n_p are the numbers of failing and passing test cases that do not execute the element under test. Next, the fault locator (such as Ochiai [1]) translates the hit-spectrum into suspiciousness of the element under test. This suspiciousness indicates the likelihood of the element under test to be at fault. The underlying intuition is that an element under test executed more in failing tests and less in passing tests gets a higher suspiciousness and appears atop in the ranking. Sorting the elements under test according to their suspiciousness in descending order produces the ranking. We refer to this as raw spectrum analysis.

However, this raw spectrum analysis comes at the cost of limited diagnostic accuracy. Since the raw spectrum analysis ignores the dependency relationships between elements under test, the top ranked element under test may not be the root cause of failure. In our previous work, we mined with itemset mining the patterns of method calls as dependency relationship in call traces (referred to as patterned spectrum analysis). We demonstrated that patterned spectrum analysis was more effective than raw spectrum analysis in localising the faults. Moreover, we also pointed out that the fault localising ability could be boosted if patterns comprising of the order preserving sequence of method calls which also allow for repetition of calls in the sequence are considered instead of patterns consisting of itemsets. We conjectured this because, with manual analysis, we noticed a unique call sequence of the faulty method which was only present in the failing test cases. The bug fix in Closure project revealed that the developers removed the code which produced this call sequence [2].

In this paper we are interested to evaluate the fault localisation ability of patterns in the call traces by considering sequence of method calls instead of itemsets. We refer to it as sequenced spectrum analysis. We put the sequenced spectrum analysis in comparison with both raw spectrum analysis and patterned spectrum analysis and note some observations. In our case study, we initially use two open source java projects.

II. Sequenced Spectrum Analysis

In our analysis, we choose the method as the element under test. Here, we briefly describe the steps in our sequenced spectrum analysis. We run the test cases on a faulty program and in each test case, (1) collect the traces for each executed method of the project (Section II-A), (2) mine the sequences from these traces (Section II-B), (3) calculate the hi-spectrum for the call sequences (Section II-C), and finally (4) rank the methods (Section II-D) according to their likelihood of being at fault.

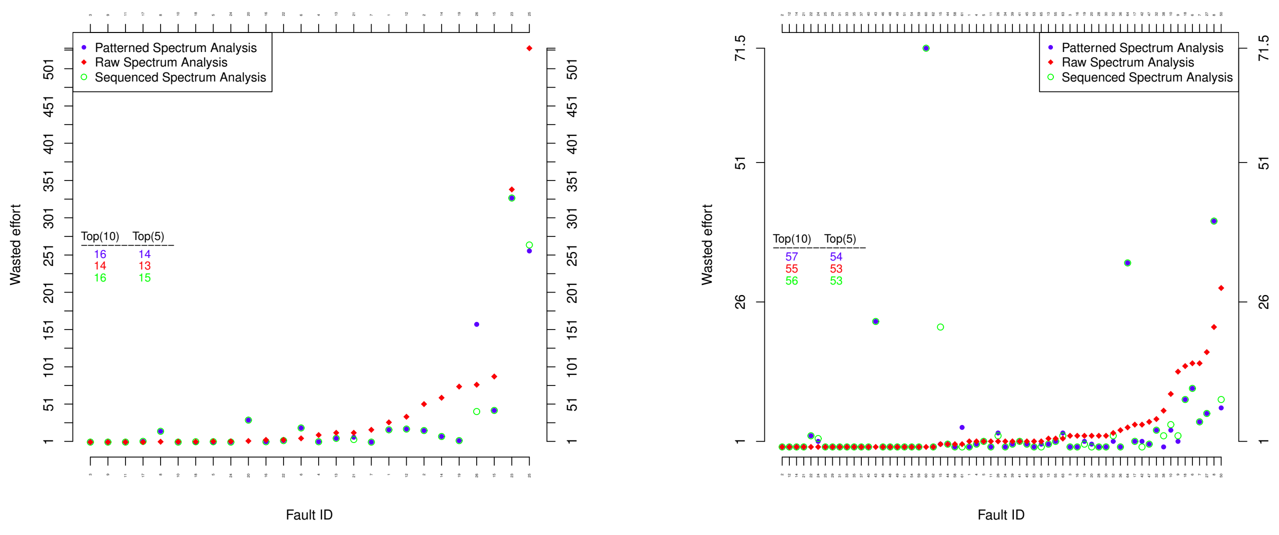
A. Collecting the Trace

In each test case, during the execution of a method, we intercept its method calls and record them into the trace. The intercepted call can be a call to a method in the project itself or the call to a method in the external library. We assign a unique integer identifier to each method. As a method can execute one or more times in a test case, we separate the call traces in each execution. The complete trace for a method `m()` in a test case $\mathcal{T}$ is represented as a set $\mathcal{T}_m = \{t_1, t_2, ..., t_n\}$. Where t_i represents the method calls invoked from method `m()` during its i^{th} execution.

B. Obtaining Call Sequences

We adopt *MARBLES algorithm* to mine the call sequences from the method call traces. The algorithm mines general, parallel, and serial sequences from a single trace sliding a window of fixed size. We only use the serial sequences [3]. We mine sequences for each call trace $ti \in \mathcal{T}_m$ of method `m()` in a test case $\mathcal{T}$.

Given an input $ti \in \mathcal{T}_m$, the algorithm produces s_i a set of call sequences as the output. The final set of the call sequences $S_{\mathcal{T}_m}$ for the method `m()` in test case $\mathcal{T}$ is the union of the set of call sequences s_i ($S_{\mathcal{T}_m} = \bigcup_{i=1}^{n} s_i$)



(a) JFreeChart (b) Apache Commons Language

Fig. 1: The comparison plots

C. *Calculating the Hit-Spectrum*

Unlike raw spectrum analysis, where there is a hit-spectrum only for element under test, sequenced spectrum analysis calculates the hit-spectrum (e_f, e_p, n_f, n_p) for each call sequence of the method.

The call sequences of a method `m()` in sequenced spectrum analysis are obtained by running the set of failing test cases (denoted as $\mathbb{T}_F$) and the set of passing test cases (denoted as $\mathbb{T}_P$). We obtain a set of call sequences $\mathcal{S}_m$ (Equation 1) for each method `m()`. The call sequences set $\mathcal{S}_m$ is the union of (i) the call sequences of a method resulting from the failing test cases ($\mathcal{S}_{\mathcal{T}_m} : \mathcal{T} \in \mathbb{T}_F$) and (ii) the call sequences resulting from the passing test cases ($\mathcal{S}_{\mathcal{T}_m} : \mathcal{T} \in \mathbb{T}_P$).

$$\mathcal{S}_m = \{s | s \in \mathcal{S}_{\mathcal{T}_m} \wedge \mathcal{T} \in \mathbb{T}_F\} \cup \{s | s \in \mathcal{S}_{\mathcal{T}_m} \wedge \mathcal{T} \in \mathbb{T}_P\} \quad (1)$$

D. *Ranking Methods*

To produce a ranked list of methods, first we assign a suspiciousness to each call sequence of a method. The fault locator function translates the hit-spectrum of the call sequence into its suspiciousness. Second, we assign a suspiciousness to the method itself. The steps are similar as in patterned spectrum analysis [2]

III. Results

As we compare sequenced spectrum analysis against patterned spectrum analysis and raw spectrum analysis, for each fault, we calculate the rankings with all three analyses. We use evaluation metric "$wasted\ effort = m + \frac{n}{2}$" [2] for the comparison of the rankings.

In our preliminary study, we use two projects (1) The Apache Commons Language with 62 faults, and (2) JFreeChart with 26 faults from Defects4J dataset [4].

For easy comparison, we show the results in a plot. The results are shown in Figures 1a and 1b. To make a plot, first we sort the rankings according to the wasted effort in raw spectrum analysis and plot them. Then for each fault, we also plot the corresponding wasted effort in patterned spectrum analysis and sequenced spectrum analysis. The plots also show the number of faults where wasted effort is in top 10 and top 5 for each analysis.

We note some observations from the case study. Like patterned spectrum analysis, sequenced spectrum analysis is also better than raw spectrum analysis. In project JFreeChart, sequenced spectrum analysis is little better in terms of wasted effort in top 10 and 5 and provides less wasted effort for 2 faults compared to patterned spectrum analysis— notable improvement for fault 26. For 1 fault the wasted effort is more, however the effect is not worst. However, there is a mixed effect in other project. There, sequenced spectrum analysis is not better in terms of wasted effort in top 10 and top 5. It provides less wasted effort for 7 faults and also more wasted effort for 7 other faults compared to patterned spectrum analysis. There is notable improvement for fault 61 and worst effect for fault 15. These preliminary findings suggest to investigate the effectiveness of sequence mining in more projects for further evaluation.

References

[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.06.035

[2] G. Laghari, A. Murgia, and S. Demeyer, "Fine-tuning spectrum based fault localisation with frequent method item sets," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016. New York, NY, USA: ACM, 2016, pp. 274–285.

[3] B. Cule, N. Tatti, and B. Goethals, "Marbles: Mining association rules buried in long event sequences," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 7, no. 2, pp. 93–110, 2014.

[4] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014. New York, NY, USA: ACM, 2014, pp. 437–440.