Evolutionary Composition of Customised Fault Localisation Heuristics

Diogo M. de Freitas[1], Plinio S. Leitao-Junior[12],
Celso G. Camilo-Junior[1] and Rachel Harrison[2] *

1- Universidade Federal de Goias (UFG) - Instituto de Informatica (INF)
Alameda Palmeiras, Quadra D, Campus Samambaia, Goiania, Goias - Brazil

2- Oxford Brookes - Dept of Computing and Communication Technologies
Wheatley Campus, OX33 1HX, Wheatley, Oxford - United Kingdom

Abstract. Fault localisation is one of the most difficult and costly parts in software debugging. Researchers have tried to automate this process by formulating measures for assessment of code elements' suspiciousness. This paper reports an evolutionary-based approach to combine non-linearly 34 previous measures to formulate a new program oriented fault localisation heuristic. The method was evaluated with 107 single-bug programs and compared against 35 approaches – 34 spectrum-based heuristics and a previous evolutionary linear combination approach. The experiments have shown that the proposal consistently achieved competitive results compared to the others according to several effectiveness metrics.

1 Introduction

During the software development and maintenance life cycle, faults are constantly introduced and repaired. Debugging is a time-consuming process which aims to locate and fix existing faults detected in the software life cycle.

Among the essential tasks that are associated with debugging, *fault localisation* (FL) is one of the most difficult and exhausting [1] and refers to finding the actual location of the faults that are associated with the perceived deviation from the system's specification (software failure). Due to the increasing complexity in software projects, FL is increasingly becoming an onerous task [2, 3].

This paper focuses on *spectrum-based fault localisation (SFL) heuristics*, which are based on spectra data and apply formulae to find the elements that are responsible for failures [4]. A *Program spectrum* is the set of data collected at run-time that refers to the behaviour of program executions (successful/failed), i.e. it's possible to relate software elements to failure occurrence [5].

For each element n (lines or blocks of code), a SFL heuristic calculates a suspiciousness score $S(n)$, that represents the strength of association between an element's executions and failure occurrences. Once every $S(n)$ is calculated, a list can be organized in descending order so that the developer can analyze the elements from top (greater score) to bottom until all faults are located.

Although many researches have proposed SFL heuristics, individual performances of heuristics are similar and not sufficient, and there is no single heuristic

*We thank CAPES and Universidade Federal de Goias for the support and Oxford Brookes University for the facilities to develop the research.

that is good in every situation. So we follow the reasoning that a composition of different heuristics could be better than just one.

In [6], Wang *et al.* presented the seminal method for the combination of 22 existing SFL heuristics for a customised heuristic, but it is restricted to a linear combination and the formulae are trained to a set of programs as a whole.

This research is a follow-up to our previous work [7] and uses a Genetic Programming (GP) algorithm to formulate SFL heuristics by searching within all valid mathematical combinations of a group of functions and spectra variables. In the context of heuristic combination, the innovative aspects of our proposal are: (i) non linearity of the resulting formulae; (ii) suspiciousness heuristics fitted to potential faults in customised program fashion, i.e. the formulae are trained for each individual program; as well as (iii) the combination of SFL Heuristics that are commonly applied in FL [8] (different to the ones used in [6] and [7]) with basic spectrum variables, the ones used internally by the SFL Heurisitics formulae.

We evaluate empirically the performance of the proposed method over a set of baselines in terms of: *the overall ability of localising faults* (**RQ1**) and *the number of program elements investigated to find the existing faults* (**RQ2**).

The text is structured as follows. Sections 2 and 3 introduces related work and the proposed approach, respectively; the experiments are described in Section 4; Section 5 presents the results their analysis; and Section 6 concludes and shows future work.

2 Related Work

SFL heuristics are commonly equations that use spectrum variables such as: number of test cases n; code element e; number of successful [or failed] executions n_s [n_f]; number of successful [or failed] executions of element e $n_s(e)$ [or $n_f(e)$]; and number of successful [or failed] runs that don't execute the element $n_s(\overline{e})$ [or $n_f(\overline{e})$]. Thus the research field has introduced several SFL heuristics to calculate the suspiciousness score; e.g. *Tarantula* [9] and *Ochiai* [10] (Equations 1 and 2).

$$S(e)_{tarantula} = \frac{\frac{n_f(e)}{n_f}}{\frac{n_s(e)}{n_s} + \frac{n_f(e)}{n_f}} \quad (1) \qquad S(e)_{ochiai} = \frac{n_f(e)}{\sqrt{n_f \times n(e)}} \quad (2)$$

Wang *et al.* [6] proposed a Genetic Algorithm (GA) search for the 22 weights w_n in Equation 3 that correspond to each H_n SFL Heuristic (Tarantula and Ochiai along with 20 association measures). Thus, the generated composite heuristic $H_C(e)$ (Equation 3) is a linear combination of previous heuristics. The weights are the individuals and they are represented in binary form.

$$S(e)_{linear\ composition} = H_c(e) = w_1 \times H_1(e) + w_2 \times H_2(e) + \cdots + w_{22} \times H_{22}(e) \quad (3)$$

Yoo [11] introduced a Genetic Programming (GP) approach for evolving risk assessment formulae using only coverage variables (not a composition of

heuristics), and Xie *et al.* [12] performed theoretical evaluations of Yoo's GP-evolved formulae for programs with a single fault and stated that the GP can be an adequate tool for designing the risk evaluation of program elements.

3 Approach

This paper presents a development over the aforementioned efforts in the form of a Genetic Programming-based approach to combine existing SFL Heuristics. The resulting suspiciousness heuristics are not limited by linear combination (such as in [6]) as more complex formulae are allowed. Furthermore, the approach formulates *program-oriented* fault localisation heuristics, i.e. the heuristics are trained per program and hence more specialised, different from [11].

The search space is defined by all valid GP individuals composed by the functions observed in SFL Heuristics ($+$, $-$, $*$, $\div$, $\sqrt{\ }$, log and max) and the set of terminals defined by 34 SFL Heuristics (the ones listed in Table 4 of [8] along with Tarantula, Ochiai, Ochiai2) and the 4 spectrum variables in Section 2. The fitness function is the average proportion of program elements that need to be investigated to locate all faults in each program.

The proposed method is structured in two phases, training and deployment. In the training phase, an heuristic is customised to a program with a set of its known faulty versions by the GP algorithm. It's expected that the trained heuristic will be used in the debugging of many future versions of the program. In our experiments we use two thirds of the versions of each program to training and validate the results with the other one third. This process is repeated many times to account for the GP's stochastic nature. Equation (4) is an example of an obtained solution, where H_i denotes an existing SFL Heuristic.

$$\frac{\log\left(\frac{\left(\sqrt{\max\left(\max(H_{21},H_{30}),n_f(\overline{e})\right)}-\sqrt{H_5+H_{15}+H_5+H_{11}}\right)}{\sqrt{n_f(\overline{e})-\sqrt{H_{13}-H_{31}}}}\right)}{\sqrt[4]{H_7\times n(\overline{e})}} \tag{4}$$

4 Experiments

To compare the performance of the proposal we applied the 34 SFL Heuristics listed in a recent Survey [8] as they are the most used in FL studies. The linear composition approach proposed by Wang *et al.* [6] (GA-based method) is used as the evolutionary baseline for the evaluation. We named the baselines as $H_1, H_2, ..., H_{34}$ (adopting the numbering of Table 4 from [8] with Tarantula as H_{32}, Ochiai as H_{33} and Ochiai2 as H_{34}), GA and GP (our method).

Seven programs from the *Siemens Suite* were used: *printtokens, printtokens2, replace, schedule, schedule2, tcas, tot_info.* We selected 107 faulty versions and their spectra were generated with the tools *lcov*, *gcov* (a GNU standard test coverage tool) and a custom program. Another third party tool used was the Java framework JGAP, which allows the abstraction of the basic operators necessary for the implementation of evolutionary algorithms.

For both the GA and the GP, JGAP was used with the most basic settings (standard configuration), and we applied *lines of code* as the basic atomic element. We specify the number of runs as 30 aiming to reduce stochastic effects. We performed a cross validation: two groups are used for training and the remaining one used for testing, and three iterations are performed so that each group of programs is used once as the testing set.

5 Results

To analyse the number of investigated elements, we apply two evaluations: ***Accuracy* (acc@n)**, refers to the number of faults that have been localised within the top n places of the ranking (upper values are better); and ***Wasted Effort* (wef@n)**: refers to the amount of effort wasted looking at non-faulty program elements (lower values are better). We used 1, 3, and 5 as the values for n.

The overall results of the fault localisation are shown in Table 1. It presents *acc* and *wef* values of all programs for 34 metrics, GA, and GP. For $n = 1$, GP outperformed all others, which represents better results for the minimal effort to localise faults. With respect to the GA results, they were poor in relation to most of the metrics, which may be related to the linearity of the GA equations.

	acc@1	**acc@3**	**acc@5**	**wef@1**	**wef@3**	**wef@5**
$\boldsymbol{M_1}$	7	21	25	100	279	446
$\boldsymbol{M_2}$	7	23	28	100	276	438
...	...	...	...	...	...	...
$\boldsymbol{M_{34}}$	7	22	28	100	277	439
GA	0.33	6.77	9.00	106.67	313.20	509.33
GP	**8.77**	**23.27**	**35.53**	**98.23**	**272.23**	**419.67**

Table 1: Evaluation measures: acc@n and wef@n.

We have two research questions, which are discussed next.

RQ1: *How do the performances for fault localisation compare?*

We calculate the mean position of a ranking related to the *acc* and *wef* measures, referred to as *Average Rank*. For each evaluation measure (*acc*@1, *acc*@3, and so on), the average rank of Metric M is the mean position of its relative quality positions (1st, 2nd, 3rd, etc.) computed for all programs. The average rank is presented in Table 2. We highlight in bold that the average ranks for GP are better across almost all columns in Table 2. This occurs since GP values consistently perform best when looking at individual programs.

acc@1	**acc@3**	**acc@5**	**wef@1**	**wef@3**	**wef@5**
GP: 15.57	**GP: 9.57**	**GP: 5.86**	**GP: 15.57**	**GP: 7.86**	**GP: 8.14**
M_7 : 25.14	M_{16} : 16.43	M_{29} : 9.00	M_7 : 25.14	M_{11} : 14.14	M_{16} : 8.14
M_{16} : 30.00	M_{29} : 16.43	M_{16} : 9.00	M_{16} : 30.00	M_{16} : 14.14	M_{29} : 8.14
M_{29} : 30.00	M_{11} : 17.57	M_7 : 18.57	M_{29} : 30.00	M_{29} : 14.14	M_{11} : 14.14
M_8 : 31.00	M_5 : 20.57	M_{33} : 19.86	M_8 : 31.00	M_{24} : 15.86	M_{24} : 14.29

Table 2: Average rank: acc@n and wef@n.

RQ2: *How many program elements must be investigated to find the faults?*
We show the analysis of two programs: the ones next to the minimum and maximum number of faults (7 and 28 fault versions, respectively), to avoid fewest and most-trained formulae. Table 3 reveals the top-five metrics of *acc* and *wef* values for Programs *replace* and *schedule*. Each column is ordered independently (better values first). Note that GP values are all bold in the table, showing better-adapted performance than others. Furthermore, their values take the top values in 9 of the 16 evaluation measures.

P3 – replace – (28 versions)					
acc@1	**acc@3**	**acc@5**	**wef@1**	**wef@3**	**wef@5**
M_{16}: 6	**GP: 12.3**	**GP: 13.93**	M_{16}: 22	**GP: 56.6**	**GP: 84.93**
M_{29}: 6	M_{16}: 12	M_7: 13	M_{29}: 22	M_{16}: 57	M_{16}: 87
GP: 5.63	M_{29}: 12	M_{16}: 13	**GP: 22.37**	M_{29}: 57	M_{29}: 87
M_1: 5	M_2: 10	M_{29}: 13	M_1: 23	M_2: 61	M_2: 97
M_2: 5	M_5: 10	M_2: 10	M_2: 23	M_5: 61	M_5: 97
P4 – schedule – (7 versions)					
acc@1	**acc@3**	**acc@5**	**wef@1**	**wef@3**	**wef@5**
GP: 0.33	M_5: 2	**GP: 3.6**	**GP: 6.67**	M_{11}: 17	M_{11}: 26
GA: 0	M_{11}: 2	M_{11}: 3	GA: 7	M_5: 18	**GP: 26.23**
M_1: 0	**GP: 1.03**	M_{16}: 3	M_1: 7	**GP: 18.7**	M_5: 28
M_2: 0	M_1: 1	M_{29}: 3	M_2: 7	M_7: 19	M_{16}: 28
M_3: 0	M_2: 1	GA: 2	M_3: 7	M_8: 19	M_{29}: 28

Table 3: Top-five metrics of *acc@n* and *wef@n* for Programs *replace* and *schedule*.

In summary, our method is well-adapted to locating faults over the baselines, according to the results. The method is also dependent to the training since better values are reached with more of faulty versions and test cases.

Statistical Analysis. Statistical tests were carried out using the *Wilcoxon* Rank Sum Test to assess statistically significant differences, and *Vargha and Delaney's* $\hat{A}_{12}$ statistic for effect size comparison. The tests were performed comparing 30 runs for each program's deployment sets. The Wilcoxon test indicates that the differences between GP and GA have statistical significance at the 95% level in all but five deployment sets of the different programs. Overall significant statistical differences between GP and GA have been observed across all programs with GP's results significantly superior in programs *printtokens2*, *replace*, *tcas* and *tot_info* (all programs for which GP had better results in proportion of inspected lines to locate all faults). Also, GP had superior statistically significant differences in P1, which it lost to GA in average values.

Threats to Validity. We mitigated threats to *internal validity*, i.e. reducing results by chance, by using: baselines and evaluation measures used in prior studies; 30 executions to reduce the algorithm's stochastic-nature; and public open source frameworks used in a variety of applications. To deal with *external validity*, i.e. whether the results can be generalised, a benchmark used in many contexts related to software engineering was applied. Finally, to cope with threats to *construct validity*, how well the measurements are actually correlated to what they claim to do, measures are used similarly to previous studies.

6 Conclusion

We apply an evolutionary meta-heuristic – Genetic Programming (GP) – to derive non-linear equations which represent suspiciousness measures of potential faulty code elements. The work uses a program-oriented approach aiming to find better-adapted formulae for fault localisation problems.

Well-known fault-localisation measures were used as baselines, and one meta-heuristic method (a Genetic Algorithm) which derives linear equations. Despite the evolutionary perspective of the latter, its results are poor due to its limited search space. According to our results, the GP-based method allows for complex formula and better adaptation to a set of programs: the project-based training results in formulae customised to the potential fault set of a particular program.

As further work, the performance and adaptability on larger programs with real faults, different types of bugs and multi-fault programs will be investigated.

References

[1] James A. Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 467–477, New York, NY, USA, 2002. ACM.

[2] I Vessey. Expertise in debugging computer programs: An analysis of the content of verbal protocols. *IEEE Trans. Syst. Man Cybern.*, 16(5):621–637, September 1986.

[3] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

[4] Lucia, D. Lo, Lingxiao Jiang, and A. Budi. Comprehensive evaluation of association measures for fault localization. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10, Timisoara, Romania, Sept 2010. IEEE Computer Society.

[5] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J. C. van Gemund. A practical evaluation of spectrum-based fault localization. *J. Syst. Softw.*, 82(11), November 2009.

[6] Shaowei Wang, David Lo, Lingxiao Jiang, Lucia, and Hoong Chuin Lau. Search-based fault localization. In *Proceedings of the 2011 26th International Conference on Automated Software Engineering*, ASE '11, Washington, DC, USA, 2011. IEEE Computer Society.

[7] Diogo de Freitas, Plinio Leitao-Junior, Celso Camilo-Junior, Altino Dantas, and Rachel Harrison. Genetic programming-based composition of fault localization heuristics. In *CBSoft 2017 - WESB*, Fortaleza, CE, Brazil, Sept 2017.

[8] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, Aug 2016.

[9] James A. Jones, Mary Jean Harrold, and John T. Stasko. Visualization for fault localization. In *in Proceedings of ICSE 2001 Workshop on Software Visualization*, pages 71–75, Toronto, ON, Canada, 2001. ICSE International Conference on Software Engineering.

[10] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, Washington, DC, USA, 2006. IEEE.

[11] Shin Yoo. Evolving human competitive spectra-based fault localisation techniques. In *Search Based Software Engineering*, volume 7515 of *Lecture Notes in Computer Science*, pages 244–258. Springer Berlin Heidelberg, 2012.

[12] X. Xie, Fei-Ching Kuo, Tsong Yueh Chen, Shin Yoo, and Mark Harman. Provably optimal and human-competitive results in sbse for spectrum based fault localisation. In *Search Based Software Engineering*, volume 8084 of *Lecture Notes in Computer Science*, pages 224–238. Springer Berlin Heidelberg, 2013.