New Ranking Formulas to Improve Spectrum Based Fault Localization Via Systematic Search

Qusay Idrees Sarhan[1, 2], Tamás Gergely[1], and Árpád Beszédes[1]
[1] Department of Software Engineering, University of Szeged, Szeged, Hungary
[2] Department of Computer Science, University of Duhok, Duhok, Iraq
{sarhan, gertom, beszedes}@inf.u-szeged.hu

***Abstract*—In Spectrum-Based Fault Localization (SBFL), when some failing test cases indicate a bug, a suspicion score for each program element (e.g., statement, method, or class) is calculated using a risk evaluation formula based on basic statistics (e.g., covering/not covering program element in passing/failing test) extracted from test coverage and test results. The elements are then ranked from most suspicious to least suspicious based on their scores. The elements with the highest rank are believed to have the highest probability of being faulty, thus, this light-weight automated technique aids developers to find the bug earlier. Several SBFL formulas were proposed in the literature, but the number of possible formulas is infinite. Previously, experiments were conducted to automatically search new formulas (e.g., using genetic algorithms). However, no systematic search for new formulas were reported in the literature. In this paper, we do so by examining existing formulas, defining formula structure templates, generating formulas automatically (including already proposed ones), and comparing them to each other. Experiments to evaluate the generated formulas were conducted on Defects4J.**

***Index Terms*—Debugging, automated fault localization, spectrum-based fault localization, formulas, systematic search.**

I. Introduction

Various automated fault localization techniques have been proposed over the last few decades including Spectrum-Bbased Fault Localization (SBFL). In SBFL, the probability of each program element (e.g., statements) of being faulty is calculated based on program spectra obtained by executing a number of test cases and a formula that uses basic statistics of this spectra. However, automated fault localization using this technique is not yet widely used in the industry because it poses a number of issues and challenges [1]. One of such issues is that in SBFL different formulas produce different results. In SBFL program elements are ranked in order of their suspicion scores (calculated by the used formula) from the most suspicious to the least. To decide whether an element is faulty or not, programmers examine each element starting from the top of the ranking list. To help developers discover the faulty element early in the examination process and with minimal effort, the faulty element should be put near to the highest place in the ranking. Often, SBFL formulas do not put faulty program elements higher in the ranking list to be examined first [1].

In this paper we propose to systematically search for SBFL formulas, based on formula templates. First, we examine existing formulas reported in the literature to define formula templates which describe the structure of the formula. Then, we systematically generate all possible formulas for these templates and examine them. To eliminate redundancy in the work, we sort out *useless* (constant or duplicate) formulas, determine *equivalent* (that produce the same rankings), *inverse* (that produce exactly the opposite rankings), and *mostly equivalent* ones (that only differ from each other in special cases).

To illustrate the concept, we performed a preliminary search with a simple formula template. We used the generated formulas on the Defects4J dataset to compute rankings and determine their effectiveness. While most of our preliminary expectations about the performance of the formulas are supported by the results, some numbers indicated that handling special cases can have strong influence on the effectiveness of similar formulas. In the future, we plan to analyze more formula templates and examine how different handling of special cases influences the performance of formulas.

The main contributions of this paper are as follows:

1) The idea of finding new SBFL formulas via a systematic search based on formula templates.
2) A formula template based on existing formulas and an evaluation of formulas derived from it.

II. Background on SBFL

At the beginning of the SBFL process, the execution of test cases on program elements is recorded to extract the spectra for the program under test. Program spectra is a two-dimensional matrix used to represent the relationship between the test cases and the program elements. Its rows demonstrate the test cases and its columns represent the program elements. A cell of the matrix indicates whether the corresponding program element (column) is covered by the corresponding test case (row). Test results (passed or failed) are also stored in the matrix (in an extra column).

The following four basic statistical numbers are calculated from the spectra for each program element e: (a) ep: number of passed test cases covering e; (b) ef: number of failed test cases covering e; (c) np: number of passed test cases not covering e; (d) nf: number of failed test cases not covering e.

Then, these four basic statistics can be used by an SBFL formula, e. g., Barinel ($\frac{ef}{ef+ep}$) [2] or Wong II ($ef - ep$) [2], to compute the suspicion score for each program element.

Finally, a ranking list based on the scores is produced as an output. An element ranked the highest in the list is the most suspicious of containing a fault. Thus, SBFL can help developers to find the faulty element in a target program easier.

III. Related Works and Goals

This section briefly presents the most relevant works of improving SBFL by targeting its formulas.

The first approach is to "guess" new SBFL formulas that outperform the existing ones. For example, the authors in [3] proposed a new SBFL formula called "DStar". The proposed formula has been compared with several widely used formulas and it showed good performance compared to others.

The second approach is formula modification. The authors in [4] modified three well-known SBFL formulas based on the idea that some failed test cases may provide more testing information than other failed test cases. Therefore, for the three used formulas, different weights for failed test cases were assigned and then applied with multi-coverage spectra.

Another way is to combine existing formulas. The authors in [5] proposed a new SBFL formula by combining 40 different formulas using different voting systems. The results of the experiments have shown that the formula generated by their method is better than several existing ones. The authors of [6] also created a hybrid formula which combines the advantages of other existing formulas.

The authors in [7] used genetic programming (GP) to evolve new formulas from a hybrid dataset (i.e., from different bug benchmarks). They were able to produce several formulas that outperformed many existing ones. However, their approach, by the random nature of genetic programming, is not systematic in the sense that it might miss obvious candidates, thus it does not guarantee that even a simple formula is examined.

Finally, involving new information to existing SBFL formulas can also lead to improvements. For example, the authors in [8] utilized the method calls frequency during the execution of failed tests to add new contextual information to existing formulas. Thus, the ef of each formula was changed to the frequency ef. Their experiments improved SBFL effectiveness.

All the aforementioned studies improved the performance of SBFL formulas in different ways. However, using systematic search (in contrast to ad-hoc, intuitive or heuristic methods) for introducing new formulas is a novel approach which has not been investigated previously. We are doing this by defining formula templates (based on existing formulas) and instantiate them in all possible ways. This approach also has limitations and arise problems. First, the more generic a template is, the more combinatorial explosion we will face and more formula instances we will get. Second, as Naish et al. [9] and Xie et al. [10] have shown, in theory, several formulas can produce the same rankings. We can utilize this result when a large number of new formulas are generated: it is enough to (and practically we have to) choose a single representative of the equivalent ones. Our goal is not to show formula equivalences, but to find completely new formulas.

In this paper, we introduce our preliminary results with a relatively restricted formula template. Our goal is to find out how different the generated formulas will be in terms of their ranking ability, and how we can limit the combinatorial space.

IV. Systematic analysis

Several researchers have been trying to find new and better formulas and numerous formulas have been proposed in the literature in the past decades, but no research is known to us that carried out a systematic search to find new possible formulas. On the one hand, this is understandable since the number of possible formulas is infinite.

On the other hand, by examining the reported SBFL formulas, we found that groups of them have similar structures, so there could be built using patterns, in other words, *formula templates*. Using a single formula template, the number of formulas that can be generated is finite, thus, the formulas can be systematically produced and examined. Utilizing this property, we examined the set of already reported formulas and defined a single linear/reciprocal formula template for this study that cover several existing formulas. In this template, BSN will denote the basic statistical numbers, i.e. $\mathrm{BSN} = \{ef, ep, nf, np\}$. The following template can literally yield in, for example, the Barinel, Jaccard, Kulczynski, Wong (I-II) formulas, but also covers the ranking of e.g. SBI or OP2 [2].

$$\frac{\sum_{t\in \mathrm{BSN}\cup\{1\}} n_t t}{\sum_{t\in \mathrm{BSN}\cup\{1\}} d_t t} = \frac{n_{ef}\,ef + n_{ep}\,ep + n_{nf}\,nf + n_{np}\,np + n_1}{d_{ef}\,ef + d_{ep}\,ep + d_{nf}\,nf + d_{np}\,np + d_1},$$

where $\{n, d\}_{\{ef,ep,nf,np,1\}} \in \{-1, 0, 1\}$ are the coefficients in the numerator and denominator. This formula yields $29,040$ valid and usable formulas (division by zero and constant formulas could be excluded, and $\frac{a}{b} = \frac{-a}{-b}$ are the same). Among these formulas, there are several groups, within which all the formulas have the same ranking effect; the most obvious examples are $\frac{a}{b} \equiv \frac{a+b}{b} \equiv \frac{a-b}{b} \equiv \frac{a+1}{b} \equiv \frac{a-1}{b}$.

A. Formula template for the experiments

In this paper we illustrate the method using a simplified version of the first formula template:

$$\frac{\sum_{t\in\{ef,ep\}} n_t t}{\sum_{t\in\{ef,ep\}} d_t t} = \frac{n_{ef}\,ef + n_{ep}\,ep}{d_{ef}\,ef + d_{ep}\,ep},$$

where $\{n, d\}_{\{ef,ep\}} \in \{-1, 0, 1\}$ are the coefficients in the numerator and denominator, and $0ef + 0ep$ is treated as constant 1. This template yields 81 formulas. 32 of these formulas are mathematical duplicates ($\frac{-a}{-a} = \frac{a}{a}$), 9 of the rest are constants (in forms $\frac{1}{1}$, $\frac{a}{a}$ and $\frac{-a}{a}$). These can be excluded from the examination. The remaining 40 formulas are 20 *normal* and *inverse* pairs ($\frac{a}{b}$ and $\frac{-a}{b}$), which pairs will produce exactly the reverse rankings.

It is theoretically possible that both formulas of a pair produce good results for different subsets of bugs, thus, we cannot leave out either of them from the measurement. However, a normal and its inverse formula have similar attributes and relations to other formulas (Note, that it is arbitrary which element of a pair is treated as normal or inverse).

Formulas in the form $\frac{a}{1}$ are similar in their rankings with their $\frac{1}{-a}$ versions (the latter are denoted by orange background in Table I). Differences are due to program elements for which the denominator of the used formula equals zero. This gives 8 formulas (plus 8 inverses).

The remaining 12 formulas have 4 denominators, 3 formulas sharing each one. The 3 formulas with the same denominator are equivalent with each other regarding the rankings they produce (their computed suspicious scores differ only by a constant from each other), thus, any two of them can be eliminated from the measurements. Furthermore, the 4 remaining formulas (and their inverses, denoted by green background) are also mostly equivalent with each other as they produce similar rankings except for elements with zero denominators.

Thus, we have $8+4$ formulas and their 12 inverses. The 24 formulas to be measured are shown in Table I, which includes well known formulas Barinel (F10), Wong I (F2), and Wong II (F16). Note, if we had no program elements for which ep or ef is zero or $ep = ef$, then the formulas in the same row would produce the same rankings, thus, it would be enough to measure only 5 different formulas and their 5 inverses.

TABLE I
VARIANTS OF FORMULAS. FORMULAS IN THE SAME ROW ARE EQUIVALENT TO EACH OTHER UNDER THE CONDITION IN THE COLUMN HEADER. FORMULAS F13-F24 ARE THE INVERSES OF FORMULAS F1-F12.

conditions →	$ep \neq 0$	$ef \neq 0$	$ep + ef \neq 0$	$ep \neq ef$
F1 = ep	F5 = $\frac{1}{-ep}$			
F2 = ef		F7 = $\frac{1}{-ef}$		
F3 = $ep + ef$			F9 = $\frac{1}{-ep-ef}$	
F4 = $ep - ef$				F11 = $\frac{1}{-ep+ef}$
	F6 = $\frac{ef}{ep}$	F8 = $\frac{-ep}{ef}$	F10 = $\frac{ef}{ep+ef}$	F12 = $\frac{ef}{ep-ef}$
F13 = $-ep$	F17 = $\frac{1}{ep}$			
F14 = $-ef$		F19 = $\frac{1}{ef}$		
F15 = $-ep - ef$			F21 = $\frac{1}{ep+ef}$	
F16 = $-ep + ef$				F23 = $\frac{1}{ep-ef}$
	F18 = $\frac{-ef}{ep}$	F20 = $\frac{ep}{ef}$	F22 = $\frac{-ef}{ep+ef}$	F24 = $\frac{-ef}{ep-ef}$

In the analysis above, we examined the formulas to find equivalences and similarities. Finding equivalences was not our goal, but reduced the number of formulas to be evaluated. However, for a larger number of automatically generated formulas, this analysis should be performed automatically as well. This issue seems to be a non-trivial mathematical problem which we are planning to investigate in future.

V. MEASUREMENTS

An appropriate dataset is required to examine fault localization. In this study, we used the faulty programs of version v1.5.0 of Defects4J [11]; where 6 open-source Java programs have 438 real single and multiple faults. We excluded 27 faults in this study due to instrumentation errors or unreliable test results. Thus, a total of 411 faults were included in our final dataset. Table II presents each program and its main properties.

TABLE II
SUBJECT PROGRAMS

Project	Number of bugs	Size (KLOC)	Number of tests	Number of methods
Chart	25	96	2.2k	5.2k
Closure	168	91	7.9k	8.4k
Lang	61	22	2.3k	2.4k
Math	104	84	4.4k	6.4k
Mockito	27	11	1.3k	1.4k
Time	26	28	4.0k	3.6k
All	411	332	22.1k	27.4k

We employed method-level granularity as a program spectra/coverage type. Compared to statement-level granularity, it has several advantages, e. g. it scales well and provides users with a more understandable level of abstraction [12]. Nevertheless, there is no theoretical obstacle to investigate lower levels of granularity in the future.

In this paper, we compare all the generated SBFL formulas to each other, presented in Table I, in order to measure the effectiveness of each generated formula.

VI. EXPERIMENTAL RESULTS AND DISCUSSION

A. Achieved improvements in the average ranks

Here, we use the average rank approach, which assigns to all of the tied elements (those having the same suspicion score) their average rank. Thus, if there are E pieces of tied elements having consecutive ranks (in random order) starting from rank S in the ranked list, we assign the same rank $R = S + (E - 1)/2$ to all of these elements.

For handling the exceptional case when the denominator of the formula is 0 for a program element (the div/0 problem), we assign the 0 suspicion score to the given element.

Table III presents the average rank of each generated SBFL formula. It can be noticed that F10 (Barinel) performs the best, and formulas F2 (Wong I), F6, and F19 also produce much better average ranks than all other formulas. It is worth mentioning that according to [9], F6 is equivalent to F10 in ranking. However, the difference in results shows that div/0 conditions ($ef \neq 0$ and $ef + nf \neq 0$) makes difference in practice even for theoretically equivalent formulas. Also, the result of F19 is a bit surprising at first glance, as it is similar to F14, which is the inverse of F2 (Wong I). However, F19 fails to handle program elements that does not fail ($ef = 0$). We assign 0 suspicion score to these elements, thus ranking them lowest in the list, while F2 (Wong I) also ranks them lowest due to their natural 0 score. This shows how handling special cases, like div/0, can influence the performance of otherwise similar formulas.

TABLE III
AVERAGE RANKS OF THE GENERATED SBFL FORMULAS

Name	Average rank	Name	Average rank
F1	1956.0	F13	4129,55
F2 (Wong I)	156.61	F14	6052.76
F3	1751.38	F15	4381.29
F4	2198.84	F16 (Wong II)	3833.54
F5	2963.85	F17	3119.95
F6	216.16	F18	5899.8
F7	6012.12	F19	205.77
F8	5655.43	F20	497.87
F9	3193.56	F21	2940.67
F10 (Barinel)	38.5	F22	6160.15
F11	2674.05	F23	3373.45
F12	614.03	F24	5393.14

B. Achieved improvements in the Top-N categories

The performance of SBFL can also be evaluated by focusing on the Top-N rank positions, as follows: (a) Top-N: When the rank of a faulty program element is less or equal to N. (b) Other: When the rank of a faulty program element is more than the highest N value used in the categorizations (it is 10 in our experiments). Table IV presents the number of bugs in the Top-N categories and their percentages for the dataset.

TABLE IV
TOP-N CATEGORIES

	Top-1		Top-3		Top-5		Top-10		Other	
	#	%	#	%	#	%	#	%	#	%
F1	0	0	0	0	0	0	0	0	411	100
F2	10	2	57	14	72	18	110	27	301	73
F3	0	0	0	0	0	0	0	0	411	100
F4	0	0	0	0	0	0	0	0	411	100
F5	0	0	0	0	0	0	0	0	411	100
F6	64	16	142	35	180	44	223	54	188	46
F7	0	0	0	0	0	0	0	0	411	100
F8	0	0	0	0	0	0	0	0	411	100
F9	0	0	0	0	0	0	0	0	411	100
F10	65	16	165	40	201	49	248	60	163	40
F11	15	4	31	8	35	9	38	9	373	91
F12	57	14	124	30	158	38	199	48	212	52
F13	0	0	0	0	0	0	0	0	411	100
F14	0	0	0	0	0	0	0	0	411	100
F15	0	0	0	0	0	0	0	0	411	100
F16	17	4	34	8	36	9	39	9	372	91
F17	0	0	0	0	0	0	0	0	411	100
F18	0	0	0	0	0	0	0	0	411	100
F19	6	0	55	13	71	17	107	26	304	74
F20	13	3	37	9	51	12	74	18	337	82
F21	0	0	0	0	0	0	0	0	411	100
F22	0	0	0	0	0	0	0	0	411	100
F23	0	0	0	0	0	0	0	0	411	100
F24	16	4	32	8	34	8	38	9	373	91

This evaluation also shows that F10 (Barinel), F6 and F12 are the best three formulas from this set. Surprisingly, at least at first glance, the similar F8 performs very bad. The reason is again the handling of program elements with $ef = 0$ value, to which elements we assign the 0 score. As the other scores are negative (at most 0), these not failing elements will be ranked in the top of the list. Other interesting results are those of F12 and F24. These are inverses of each other, yet F24 also ranks 4% of the buggy elements in the Top-10. We think the main reason for this (besides the zero denominator, i. e. when $ep = ef$) is that when $ef > ep$ (i. e. more failing tests cover the element than passing ones), F12 turns the score into negative, ranking these elements low, while F24 will result in a positive score (negative numerator divided by negative denominator), ranking them in the top of the list. The F12, F24 pair is a practical proof that inverse rankings can perform well on different spectra, thus both of them needs to be examined.

VII. Implications and Future Plans

This paper presented formula template based systematic search for new SBFL formulas. The results of our preliminary research indicate that the proposed approach deserves further investigation. Presently, we did not find an automatically generated formula which is not published in the literature but it outperforms existing ones.

However, since the template we used was very simple, this is not surprising. We hope that by extending the template to e.g. polynomial, exponential, and/or logarithmic, we will be able to identify new formulas which outperform existing ones. We also intend to perform the following studies in future work:

- Comparing the effectiveness of the formulas generated for all the identified templates with each other.
- Combining the best formulas from different templates into a single formula that has the advantages of others.
- Performing a theoretical analysis of how formula equivalences can be automatically detected.
- Involving other benchmark datasets to measure how much impact do they have on finding good formulas.
- Finding new ways other than formula equivalences used in this paper to limit the size of our search space.
- Studying how handling exceptional cases (e.g., division-by-zero) influences the performance of SBFL formulas.

VIII. Conclusions

In this paper we proposed a systematic approach to search for new SBFL formulas using only the four basic statistical numbers from the spectra. For this purpose, formula templates are determined and the possible formulas are generated automatically. As a demonstration, we used a formula template to systematically generate all formulas for that template, then these were analyzed and their effectiveness was evaluated on the Defects4J dataset. Interestingly, the analysis has shown that in theory several formulas generated from the same template are equivalent to or should similarly rank elements to each other, while the handling of special cases (like division-by-zero) can significantly influence the practical performance of the formulas and thus the relations among them. Summing up, leveraging systematic search for finding new formulas automatically is worth investigating in the future.

References

[1] Q. I. Sarhan and A. Beszedes, “A survey of challenges in spectrum-based software fault localization,” *IEEE Access*, vol. 10, pp. 10 618–10 639, 2022.

[2] Neelofar, “Spectrum-based Fault Localization Using Machine Learning,” 2017. [Online]. Available: https://findanexpert.unimelb.edu.au/scholarlywork/1475533-spectrum-based-fault-localization-using-machine-learning

[3] W. E. Wong, V. Debroy, R. Gao, and Y. Li, “The dstar method for effective software fault localization,” *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 290–308, 2014.

[4] Y.-S. You, C.-Y. Huang, K.-L. Peng, and C.-J. Hsu, “Evaluation and analysis of spectrum-based fault localization with modified similarity coefficients for software debugging,” in *2013 IEEE 37th Annual Computer Software and Applications Conference*, 2013, pp. 180–189.

[5] B. Bagheri, M. Rezaalipour, and M. Vahidi-Asl, “An approach to generate effective fault localization methods for programs,” in *International Conference on Fundamentals of Software Engineering*, 2019, pp. 244–259.

[6] J. Kim, J. Park, and E. Lee, “A new hybrid algorithm for software fault localization,” in *The 9th International Conference on Ubiquitous Information Management and Communication*, 2015, pp. 1–8.

[7] A. A. Ajibode, T. Shu, and Z. Ding, “Evolving suspiciousness metrics from hybrid data set for boosting a spectrum based fault localization,” *IEEE Access*, vol. 8, pp. 198 451–198 467, 2020.

[8] B. Vancsics, F. Horvath, A. Szatmari, and A. Beszedes, “Call frequency-based fault localization,” in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 365–376.

[9] L. Naish, H. J. Lee, and K. Ramamohanarao, “A model for spectra-based software diagnosis,” *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, aug 2011. [Online]. Available: https://doi.org/10.1145/2000791.2000795

[10] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu, “A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization,” *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, pp. 31:1–31:40, 2013.

[11] R. Just, D. Jalali, and M. D. Ernst, “Defects4J: a database of existing faults to enable controlled testing studies for Java programs,” in *International Symposium on Software Testing and Analysis (ISSTA)*. ACM Press, 2014, pp. 437–440.

[12] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang, “An empirical study of fault localization families and their combinations,” *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 332–347, 2021.