Rank Aggregation for Effective and Efficient Fault Localization

Ashish Ramrakhiani
School of EECS
Oregon State University
Corvallis, United States
ramrakha@oregonstate.edu

Lakshita Malhotra
School of EECS
Oregon State University
Corvallis, United States
malhotrl@oregonstate.edu

***Abstract*—**

***Index Terms*—**

I. Introduction

A. Problem statement

[[What is the problem statement that you plan to solve?]]

Fault localization families utilize diverse bug information sources to localize bugs in the code. Although these techniques individually provide satisfactory results, research has shown that none of the family of techniques are the best and their effectiveness is limited by the nature of bug information source they use for fault localization [1]. These limitations lead to the unavailability of a one-size-fits-all solution. Recent studies have shown that combining multiple fault localization techniques using supervised learning improves fault localization [2], but these approaches require the manual creation of annotated datasets to combine FL results which is difficult to create and leads to a substantial manual overhead for software practitioners.

Instead, we ask the question, is supervised learning really required? Can unsupervised methods, which require no training data for combining ranked lists, perform as well as these supervised methods and improve practitioner productivity by localizing bugs in their code?

B. Motivation

[[Why should anyone care about solving that problem?]]

Software practitioners dedicate a considerable portion of their time to the intricate process of debugging their code, primarily due to the laborious task of localizing faults within the software [3]. Consequently, there arises a demand for an automated software fault localization technique aimed at localizing bugs in the code, thus streamlining the debugging process and enhancing productivity. Over the years, numerous families of software fault localization techniques have emerged to automatically detect bugs in the code, each utilizing diverse bug information sources for fault localization [4] [5] [6] [7]. While these techniques individually yield satisfactory results, research indicates that no single family of techniques stands out as the optimal solution, with their efficacy being constrained by the nature of the bug information sources they rely on [2]. These limitations underscore the absence of an one-size-fits-all solution, motivating the pursuit of a comprehensive approach. Recent research on FL improvement has demonstrated that combining FL techniques, preferably ones that use different kinds of bug information sources, and using machine learning to combine the ranking outputs into a single ranking can significantly improve FL performance [2]. Such approaches require a training dataset of program statements annotated with suspicion scores from multiple FL techniques and manually labeled ground truth. The manual effort required for creating this training data poses a significant challenge. Consequently, these existing challenges impede practitioner productivity, as they expend considerable time on manual debugging and also face the overhead of generating annotated training datasets manually [3]. In light of these prevailing issues, our research endeavors to develop an unsupervised fault localization technique that combines the results of various fault localization techniques, thereby facilitating practical and scalable automated fault localization.

C. Relation with software engineering research

[[How is the problem related to software engineering research? What is already known about that problem space and what is still unknown that you are interested in solving?]]

Many software fault localization techniques have been developed over the years to automatically localize bugs in the code. These bug localization techniques use varied bug information sources to localize faults. Some families of techniques such as Spectrum-based fault localization compute a suspicion score for each program element using program spectra of passing and failing test cases [4] [5]. Other families apply information retrieval methods for fault localization based on the similarity of bug reports and source code referred to as Information-Retrieval-based Fault localization [6]. Additionally, there are Mutation based Fault localization techniques that utilize test results from mutating the program [7] [8], Dynamic program slicing techniques that utilize dynamic program dependencies [9], Stack trace analysis that utilizes crash reports [10], History based fault localization that utilize development history [11], LLM based fault localization that leverage LLM's program synthesis capabilities to localize faults [12] etc. However, each of fault these localization techniques has its own limitations depending upon the bug information source that it utilizes to localize bugs [1]. Studies have shown that none of them are the best and they work well with localizing the type of bugs that directly correspond to a particular bug information source they use as input [2]. Therefore, there is no one-solution-fits-all technique currently available. Recent research has demonstrated that combining multiple fault localization techniques can improve fault localization [2]. These combining techniques take multiple fault localization results as inputs, which are typically a ranked list of program statements likely to be buggy and produce an optimal rank list. However, most of these combining approaches use supervised learning for rank aggregation [13] [2] [14] which requires extensive training data as well as poses a manual overhead of creating an annotated dataset which is impractical for real-world scenarios. Unsupervised techniques, exemplified by SBIR [15], have shown effective fault localization performance to improve automated program repair. However, the technique evaluates effectiveness for fault localization for Top-25, Top-50, and Top-100 metrics as automated program repair techniques are capable of evaluating hundreds of program elements, however, software practitioners in real life, most likely, will not go beyond 5 program statements to localize bugs in their code [16]. Doing so is not efficient in the real world as well as it hampers software practitioner's productivity tremendously.

Hence, the exploration of whether unsupervised methods can be utilized to aggregate results from diverse fault localization techniques employing different bug information sources to enhance fault localization is yet to be undertaken. We are keenly interested in addressing the exploration of whether this unsupervised approach to fault localization, which involves merging fault localization results, can enhance the productivity of software practitioners by automatically identifying bugs in their code.

D. Key Insight or Idea

[[What is the high-level approach that you would like to explore to the solve the problem? Why you feel you can succeed with that approach in 5 weeks?]]

Fault localization (FL) techniques are commonly used to assign suspiciousness scores to numerous program statements. The challenge arises when attempting to combine multiple ranked lists generated by these techniques, as they often exhibit inconsistencies. Achieving a result that closely aligns with individual lists, based on a given distance metric, can become computationally infeasible. In response to this issue, we propose our unsupervised technique Bugsleuth.

BugSleuth will operate as follows. Let L_1, L_2, ..., L_m represent m ordered lists of suspicious statements ordered by suspicion scores obtained through various FL techniques. BugSleuth aims to construct an ordered list δ of length k (where k $\geq$ 1) by minimizing the sum of distances between δ and the individual input lists using a rank aggregation algorithm. Formally, BugSleuth will minimize the objective function $f(\delta) = \sum_{i=1}^{m} \mathrm{d}(\delta, L_i)$, where d is a distance metric and δ is the optimal list using a rank aggregation algorithm.

E. Assumptions

[[What kind of assumptions will you need to make for your choice of solution?]]

In our project, we are making the following assumptions:

1) Availability of input rank lists ordered by suspicion score obtained via various fault localization techniques, preferably that use a different bug information source.

2) Existence of an efficient rank aggregation algorithm to generate an optimal ordered rank list which is as similar as possible to all individual input lists. Identification of an optimal aggregation algorithm that balances the varying reliability and relevance of program statements from different sources is of upmost importance.

3) Existence of a distance metric which serves as a robust measure of the positional differences between rankings capturing both the order and magnitude of displacements in program statements to evaluate the similarity / dissimilarity between input lists and the combined list.

F. Research questions

[[What research question(s) will you answer?]]

We will answer the following research questions:

RQ1: How effective is BugSleuth (unsupervised rank aggregation) for localizing defects ?

RQ2: How efficient is using BugSleuth (unsupervised rank aggregation) for fault localization ?

RQ3: How does BugSleuth (unsupervised rank aggregation) compare against state-of-the-art fault localization techniques ?

G. Evaluation Dataset

[[What kind of dataset will you evaluate your solution on? Where and how will you get that dataset?]]

We will use the Defects4J (v2.0) [17] benchmark to evaluate our FL technique. Defects4J targets Java 8 and consists of 835 reproducible defects from 17 large open-source Java projects. Each defect comes with (1) one buggy and one developer-repaired version of the project code (2) a set of developer written tests, all of which pass on the developer-repaired version and at least one of which evidences the defect by failing on the buggy version; and (3) defect information. Defects4J (v2.0) is available at [17].

The ground truth to evaluate the technique will be created by documenting the developer added/modified/deleted lines for 815 defects in Defects4J (v2.0) [17]. We will finetune our technique using 111 defects of the JacksonDatabind project in Defects4J (v2.0) [17].

H. Evaluation metrics

[[How will you know that you have solved the problem successfully? In other words, how will you evaluate your solution?]]

We will use two metrics, common to FL evaluations [2] : Top-N and EXAM.

- **Top-N:** Top-N measures the number of faults with at least one faulty element located within the first N positions (N=1, 3, 5). Developers only examine a small amount of the most-likely buggy elements within a ranked list [3], with particular attention paid to the top-5 elements [16]. To compare against state-of-the-art techniques, we adopt Top-N.
- **EXAM:** EXAM represents the proportion of ranked statements that need examination to identify a buggy statement.

The Top-N metric assesses the utility of a fault localization (FL) technique for a software practitioner by considering the top N statements. On the other hand, EXAM tells us how high buggy statements are ranked thereby helping practitioners localize bugs efficiently.

II. Background and Motivation

III. Related Work

IV. Approach

V. Evaluation

A. Dataset

B. Metrics

C. Experiment Procedure

D. Results

VI. Discussion and Threats to Validity

VII. Contributions

References

[1] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating & improving fault localization techniques," *University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-16-08-03*, p. 27, 2016.

[2] 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, 2019.

[3] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 international symposium on software testing and analysis*, 2011, pp. 199–209.

[4] 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 Transactions on software engineering and methodology (TOSEM)*, vol. 22, no. 4, pp. 1–40, 2013.

[5] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "On the accuracy of spectrum-based fault localization," in *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 2007, pp. 89–98.

[6] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[7] M. Papadakis and Y. Le Traon, "Metallaxis-fl: mutation-based fault localization," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 605–628, 2015.

[8] S. Moon, Y. Kim, M. Kim, and S. Yoo, "Ask the mutants: Mutating faulty programs for fault localization," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE, 2014, pp. 153–162.

[9] H. Agrawal, J. R. Horgan, S. London, and W. E. Wong, "Fault localization using execution slices and dataflow tests," in *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*. IEEE, 1995, pp. 143–151.

[10] R. Wu, H. Zhang, S.-C. Cheung, and S. Kim, "Crashlocator: Locating crashing faults based on crash stacks," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 204–214.

[11] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007, pp. 489–498.

[12] A. Z. Yang, C. Le Goues, R. Martins, and V. Hellendoorn, "Large language models for test-free fault localization," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[13] J. Sohn and S. Yoo, "Fluccs: Using code and change metrics to improve fault localization," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017, pp. 273–283.

[14] X. Li, W. Li, Y. Zhang, and L. Zhang, "Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, 2019, pp. 169–180.

[15] M. Motwani and Y. Brun, "Better automatic program repair by using bug reports and tests together," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1225–1237.

[16] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th international symposium on software testing and analysis*, 2016, pp. 165–176.

[17] G. Gay and R. Just, "Defects4j as a challenge case for the search-based software engineering community," in *Search-Based Software Engineering: 12th International Symposium, SSBSE 2020, Bari, Italy, October 7–8, 2020, Proceedings 12*. Springer, 2020, pp. 255–261.