Unveiling Simplicity in Complexity

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Software practitioners invest a substantial amount of their time in the arduous process of debugging their code, primarily due to the time-consuming task of pinpointing faults within the software. Despite the existence of automated fault localization techniques for many years, the software industry has shown reluctance in embracing these methods, largely attributable to their suboptimal performance in accurately identifying and isolating faults.

Effective fault localization is crucial for software development, yet existing techniques often rely on bug reports or test suites, which may not be readily available in practical scenarios. Recent research has demonstrated that combining fault localization techniques with supervised machine learning algorithms can enhance performance. However, these supervised learning approaches face challenges in generalizability and require large labeled datasets for training.

Unsupervised techniques, exemplified by [RAFL], have shown superior fault localization performance compared to state-of-the-art supervised techniques. Nonetheless, their reliance on the availability of underlying fault localization technique results poses challenges, especially in real-world industry scenarios where such information may not be accessible.

This research addresses these challenges by proposing innovative technology aimed at automating the fault localization process, thereby increasing productivity for software practitioners. The goal is to reduce reliance on manual bug localization, which is time-consuming and diverts resources from potentially more impactful activities.

II. Background

There are several families under which multiple standalone techniques have been developed. Spectrum-Based Fault Localization (SBFL) requires information about the coverage of a program during its runtime, typically measured in terms of code coverage. This coverage information is obtained by executing a set of tests on the program. The tests should be designed to cover various parts of the code, providing insights into the execution behavior of different program elements, such as statements or methods. Essentially, SBFL needs the execution data from both passed and failed test cases inorder to compute the suspiciousness values of code elements for fault localization. Mutation-Based Fault Localization (MBFL) requires information from mutation analysis rather than regular program execution. Specifically, it relies on mutated versions of the original program created by introducing changes (mutations) to expressions or statements. These mutants are then subjected to a set of test cases. MBFL analyzes how these mutations impact the results of both failed and passed test cases.

In contrast to Spectrum-Based Fault Localization (SBFL), MBFL considers whether the execution of a statement with mutations affects the test outcomes. Therefore, MBFL requires the ability to generate and execute these mutated versions of the program, along with the corresponding test cases to evaluate the impact of mutations on the program's behavior. The goal is to identify which program statements, when altered, have a higher impact on failed tests compared to passed tests. In Predicate Switching as the name suggests, synonymous with a conditional expression, dictates the execution of distinct branches. If altering the assessed result of a predicate can transform a failed test case into a successful one, that predicate is deemed critical and may be the underlying cause of the fault. Information Retrieval Fault Localization is an approach in fault localization where the input includes a bug report and the source code. The technique calculates a similarity score between the bug report (acting as a query) and different parts of the source code (considered as documents). Based on these similarity scores, the approach generates a probable list of code elements that are likely to be associated with the reported bug, aiming to pinpoint potential faults in the software. Recent research [5], [6], [8] highlight the substantial improvement in the ranked list generated when combining the ranked lists from different standalone fault localization techniques. Learning to rank is one such approach designed for ranking tasks. These techniques [5], [6] enhance ranking performance by automatically creating models that learn from suspiciousness values from various fault localization techniques along with other features like code complexity.The effectiveness of this approach hinges on a training process, the impact of which on its overall performance remains uncertain. Generating suit-

able training data presents challenges due to the demanding manual efforts involved, and the success of trained models is intricately tied to the quality of the data and features they are trained on. To avoid the dependency of training data, [8] proposed a novel method for combining different fault localization (FL) techniques, framing it as a rank aggregation (RA) problem. In this context, rank aggregation involves merging multiple ranked lists (base rankers) into a single ranked list (aggregated ranker). Their technique combines multiple ordered lists of suspicious statements from various FL techniques by minimizing the weighted sum of distances, where the importance weights and distance metric are key factors in the optimization process. While their approach has shown prominent results, their technique was to better improve automatic program repair. Our approach builds upon this by exploring algorithms like Monte-Carlo and Genetic Algorithm to generate more effective ranked list.

III. Approach

IV. Evaluation

In this work we aim to answer the following Research Questions (RQs)

Research Questions: RQ1: How effective are rank aggregation algorithms for localizing defects

- Genetic Algorithm
- Cross Entropy Monte Carlo Algorithm

RQ2: How efficient is using rank aggregation for fault localization

RQ3: How do these rank aggregation algorithms compare against state-of-the-art fault localization techniques

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

V. Related Work

A. Improving Fault localization

Automated fault localization techniques leverage both static and run-time information of a program to identify potential program elements responsible for faults. SBFL computes a suspiciousness score for each program element by utilizing test coverage information whereas the MBFL techniques utilizes mutation analysis by analyzing the impact of artificially introduced program mutations on test results . On the other hand, Deep Learning-Based Fault Localization (DLFL) employs neural networks to create a fault localization model, utilizing the trace matrix and test results.

The TRAIN [1] approach improves upon SBFL by identifying and excluding test cases that execute faulty statement(s) but lead to a correct output, thereby optimizing the trace matrix. Mutation-Spectrum Fault Localization [2] (MSFL) [2] combines mutation-based testing with SBFL. MSFL generates spectra for each mutant and the faulty program to combine them with SBFL techniques such as Tarantula, Barinel, Ochiai, and DStar to produce statement ranking sequences. Bug localization is then performed based on the similarity between the statement ranking sequence of the faulty program and mutants. However, MSFL assumes the thorough testing and fault-free nature of the original program thus it's effectiveness is compromised if no mutants are available for the faulty line, leading to a lower rank assigned to the faulty statement. WEGAT [3] represents the coverage matrix using a weighted execution graph by applying predicate weighted sequences combined with Abstract Syntax Tree (AST) information and feeds it to a Graph Attention Network for fault localization. However, the necessity for program instrumentation to collect predicate sequences incurs a time overhead, potentially hindering practitioner productivity and diverging from our goal of productivity enhancement.

Recently, the potential of using LLMs for various code related tasks, especially fault localization, has been explored.LLMAO [4] is a language model-based approach for fault localization that does not require test cases and locates buggy lines of code. It leverages large language models and bidirectional adapter layers to achieve high fault localization performance to detect general logic as well as security bugs in the code. Existing research has validated the effectiveness of integrating results from multiple fault localization (FL) techniques, revealing superior fault localization performance compared to standalone approaches. Noteworthy combination techniques, including CombineFL [5], DeepFL [6] and Fluccs [7] leverage learning to rank methodologies, notably RankSVM, to merge outputs from diverse FL techniques. However, these supervised approaches require labeled training datasets. The unavailability of such datasets, and even if one is willing to create them, introduces a substantial overhead, thereby contradicting the primary goal of enhanced practitioner productivity. Consequently, the practical application of these methodologies in real-world software industry settings is deemed unfeasible.

Recent research introduces an unsupervised fault localization technique known as SBIR [8] to combine SBFL and Blues(IRFL) results to acheive better automatic program repair. SBIR utilises RAFL (Rank Aggregation for Fault Localization) technique that employs the Cross Entropy Monte Carlo Algorithm and the Spearman footrule distance to merge top-k ranked lists of suspicious statements and notably yields superior results when compared to the RankSVM approach employed by many state-of-the-art supervised models. This unsupervised technique addresses the challenges associated with labeled training datasets, making it a more viable solution for practical implementation in real-world software industry scenarios, however, the finetuning of parameters for RAFL to achieve optimal fault localization results for practitioners remains unexplored.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when

writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks . . .”. Instead, try “R. B. G. thanks. . .”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

[1] J. Hu, “Trace matrix optimization for fault localization,” *Journal of Systems and Software*, vol. 208, p. 111900, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223002959

[2] A. Dutta and S. Godboley, “Msfl: A model for fault localization using mutation-spectra technique,” in *Lean and Agile Software Development: 5th International Conference, LASD 2021, Virtual Event, January 23, 2021, Proceedings 5*. Springer, 2021, pp. 156–173.

[3] Y. Yan, S. Jiang, Y. Zhang, and C. Zhang, “Improving fault localization via weighted execution graph and graph attention network,” *Journal of Software: Evolution and Process*, p. e2619.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.