Optimizing Regression Testing Efficiency Through Advanced Test Case Prioritization Techniques Using Execution Trace Diversity

Appari Pavan Kalyan[1], Dr. Harsh Pratap Singh[2], Dr. B. Kavitha Rani[3]

[1]Research Scholar, Dept. of Computer Science and Engineering Sri Satya Sai University of Technology and Medical Sciences, Sehore, Bhopal, Madhya Pradesh, India.

[2]Research Guide, Dept. of Computer Science and Engineering Sri Satya Sai University of Technology and Medical Sciences, Sehore, Bhopal, Madhya Pradesh, India.

[3]Research Co-Guide, Professor, Department of Computer Science and Engineering, CMR Technical Campus, Hyderabad, Telangana, India

How to Cite

Pavan Kalyan, A. ., Pratap Singh, H. ., & Kavitha Rani, . B. . (2023). Optimizing Regression Testing Efficiency Through Advanced Test Case Prioritization Techniques Using Execution Trace Diversity. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *12*(2), 2560–2568. https://doi.org/10.61841/turcomat.v12i2.14741

Abstract

Spectrum-based fault localization (SBFL), which utilizes spectrum information of test cases to calculate the suspiciousness of each statement in a program, can reduce developers' effort. However, applying redundant test cases from a test suite to fault localization incurs a heavy burden, especially in a restricted resource environment, and it is expensive and infeasible to inspect the results of each test input. Prioritizing/selecting appropriate test cases is important to enable the practical application of the SBFL technique. In addition, we must ensure that applying the selected tests to SBFL can achieve approximately the effectiveness of fault localization with whole tests. This paper presents a test case prioritization/selection strategy, namely the Diversity-Aware Test Optimization (DATO). The DATO strategy prioritizes/selects test cases using information on the diversity of the execution trace of each test case. We implemented and applied the DATO strategy to 233 faulty versions of the Siemens and UNIX programs from the Software-artifact Infrastructure Repository.

Keywords: Test Case Prioritization, Test Case Selection, Execution Trace Diversity, Fault Localization, Software Testing, Siemens Program, UNIX Program, Software-artifact Infrastructure Repository

1. Introduction

In the ever-evolving landscape of software development, maintaining the integrity of software systems amidst frequent changes is paramount. As software undergoes continuous updates and modifications [1], ensuring that these changes do not introduce new defects or adversely affect existing functionality is a significant challenge. To address this challenge, regression testing has emerged as a crucial practice, aimed at verifying that recent changes to the codebase do not disrupt previously validated features [2]. According to recent industry reports, regression testing accounts for approximately 60% to 70% of the total testing effort in large-scale software projects, highlighting its critical role in maintaining software quality [3]. Regression testing is a type of software testing that focuses on verifying that recent changes or additions to the software do not adversely affect the existing functionality of the application [4]. This process involves re-running previously executed test cases to ensure that new code changes have not introduced new bugs or issues. Regression testing is essential whenever software is updated, whether through bug fixes [5], enhancements, or new feature integrations. By systematically evaluating the software against a suite of test cases, regression testing helps ensure that the software remains stable and reliable over time, despite the ongoing changes [6].

The need for regression testing stems from the necessity to maintain software quality and functionality throughout its lifecycle. As software systems evolve [7], they often become more complex, with new features and bug fixes interacting in unforeseen ways. Without regression testing, there is a significant risk that new changes may inadvertently break existing functionalities, leading to a degradation in user experience and potential system failures. Effective regression testing helps identify these issues early in the development cycle, reducing the likelihood of costly post-release defects and ensuring that software updates do not compromise the overall system integrity [8]. Despite its importance, regression testing faces several existing problems that impact its effectiveness and efficiency. One major issue is the redundancy of test cases, where the same tests are executed repeatedly without adding value to the fault detection process. This redundancy not only increases the testing effort but also consumes valuable resources, leading to longer testing cycles and higher costs [9]. Additionally, the sheer volume of test cases in large test suites can make it challenging to prioritize and select the most relevant tests, potentially resulting in missed defects or ineffective fault localization. Other problems include the difficulty in managing and updating test cases as the software evolves and the lack of automation in some regression testing processes, which further exacerbates the challenge of maintaining test efficiency and coverage [10].

2. Literature survey

In [10], Xia et al. introduced a method for diversity maximization to enhance fault localization in both single-fault and multi-fault programs. Their approach focuses on accelerating the process of identifying diverse test cases that effectively pinpoint faults. The study demonstrates significant improvements in fault localization efficiency. In [11], Wu et al. conducted a theoretical and empirical analysis on the effects of coincidental correct test cases in multiple fault localization. They explored how these coincidental cases can impact the accuracy and reliability of fault localization methods. Their findings offer insights into mitigating the influence of such cases in fault detection. In [12], Renieris and Reiss presented a fault localization technique based on nearest neighbor queries. Their approach leverages proximity-based analysis to identify faults by comparing the test results of similar test cases. This method provides an efficient way to locate faults using query-based techniques. In [13], Bajaj and Sangwan performed a systematic literature review on test case prioritization methods utilizing genetic algorithms. They reviewed various strategies and outcomes of applying genetic algorithms to prioritize test cases effectively. Their work provides a comprehensive overview of how genetic algorithms can enhance test case prioritization. In [14], Zakari et al. conducted a systematic literature review on multiple fault localization in software programs. Their review covers various methodologies and approaches used to locate multiple faults within software. The study highlights trends and gaps in the current fault localization research.

In [15], GCC is identified as the GNU Compiler Collection, a widely-used compiler system. It provides essential tools for compiling and optimizing code, crucial for software development and testing. The collection includes compilers for various programming languages, enhancing development efficiency. In [16], Nayak et al. proposed a regression test case prioritization technique based on the analytic hierarchy process. Their method aims to improve the fault detection rate by systematically prioritizing test cases. The technique enhances the effectiveness of regression testing through a structured prioritization approach. In [17], Lei et al. introduced Feature-FL, a feature-based fault localization approach. Their method utilizes feature-based analysis to improve the accuracy of fault localization. Feature-FL enhances the identification of faults by focusing on relevant features within the software. In [18], Xie et al. provided a theoretical analysis of risk evaluation formulas used in spectrum-based fault localization. They examined the validity and effectiveness of various risk evaluation methods. Their analysis contributes to refining spectrum-based approaches for more accurate fault localization. In [19], Lucia et al. conducted an extended study on association measures for fault localization. Their research explores various measures and their effectiveness in fault localization. The study aims to improve the understanding and application of association measures in fault detection. In [20], Campos et al. proposed an entropy-based test generation method to enhance fault localization. Their approach uses entropy metrics to generate tests that improve the accuracy of fault localization. The method focuses on creating tests that provide more information about potential faults.

3. Proposed methodology

The DATO procedure is a sophisticated approach designed to enhance regression testing by effectively prioritizing and selecting test cases. It focuses on improving the efficiency of fault localization through strategic test case selection based on the diversity of execution traces. The core principle of DATO lies in leveraging the concept of statement coverage and the resultant test case diversity to optimize the fault localization process. Initially, the DATO procedure divides statements into two categories based on coverage from an initial failed test case: Gcf (statements covered by the failed test case) and Guf (statements not covered). Gcf represents the statements that are most relevant to the fault localization process, as it includes statements where the fault might reside. On the other hand, Guf is considered less useful in the fault localization context due to its low likelihood of containing the fault. The initial set of statements, Gcf, is used to form the base group for the DATO strategy.

To organize and prioritize test cases, DATO employs a division process that categorizes statements into different groups based on their coverage by selected test cases. This process is described through a set of division operations: Do(1,1), Do(1,0), Do(0,1), and Do(0,0). These operations represent different combinations of coverage between the current and the next selected test case. For example, Do(1,1) indicates that both the current and the next test case cover the same statement, while Do(0,0) indicates that neither test case covers the statement. This division helps in creating a hierarchical structure of groups as presented in Figure 1, where each group is assigned a unique identifier based on its level in the tree.

The DATO tree is a binary tree structure where each node represents a group of statements, and branches represent the division of groups based on test case coverage. The goal of the DATO procedure is to select test cases that minimize the aggregate diversity across the groups at each level of the tree. This is achieved by calculating a metric called Mi for each test case, which measures how well the test case can divide the statements of a group into subgroups. The test case that results in the smallest difference between the number of statements in the resulting subgroups is prioritized. This process continues iteratively, with each level of the tree being processed until all selected test cases have the same Mi value, indicating convergence. By employing this strategic approach, DATO aims to improve the effectiveness of regression testing by ensuring that the selected test cases provide a balanced and comprehensive coverage of the statements relevant to fault localization. This method not only enhances the efficiency of fault detection but also optimizes the resource utilization in test case execution, making it a valuable technique in the context of regression testing.

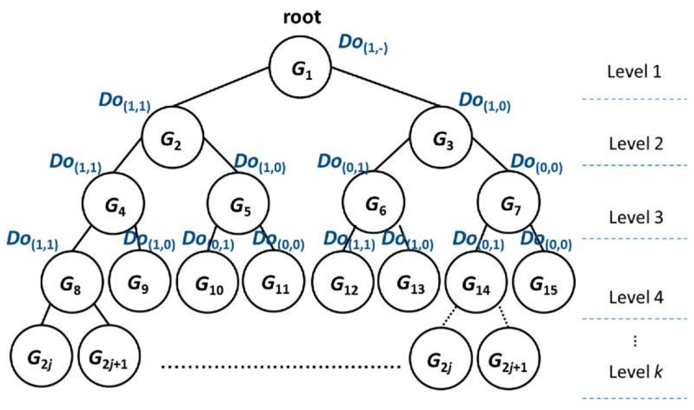


Figure 1. DATO Tree.

DATO Algorithm

The DATO algorithm iteratively selects test cases to maximize the diversity of statement coverage, ensuring that the statements in each group are evenly divided across the test cases. This approach aims to improve the efficiency of fault localization by prioritizing test cases that provide the most balanced coverage of the statements.

Step 1: Initialize Groups and Test Cases: Define groups of statements and their identifiers. Identify the statements covered by the initial failed test case and create the initial group from these statements. Start with an empty set for selected test cases and add the initial failed test case to it.

Step 2: Prepare Test Suite: Remove the initial failed test case from the test suite, leaving the remaining test cases to be evaluated.

Step 3: Begin Selection Process: Start at the first level of the DATO tree and initialize the process.

Step 4: Evaluate Test Cases: For each test case in the remaining test suite, assess how well it can divide the current group into smaller subgroups.

Step 5: Divide the Group: Use each test case to split the current group into two subgroups based on how the test case interacts with the statements.

Step 6: Select the Best Test Case: Choose the test case that provides the most balanced division of the group. Add this test case to the set of selected test cases.

Step 7: Update Test Suite: Remove the selected test case from the remaining test suite.

Step 8: Move to Next Level: Increment the level in the DATO tree and repeat the evaluation process for the next level.

Step 9: Check for Convergence: If all test cases at the current level provide the same balance in division, stop the process.

Step 10: Return the Selected Test Cases: Return the set of selected test cases that have been optimized for their ability to balance the statement coverage across groups.

4. Results and discussion

This section illustrates the iterative process of selecting test cases based on their ability to divide statement groups effectively. Each level introduces new test cases and refines group divisions to optimize fault localization by minimizing the differences in group sizes after division, ensuring a comprehensive approach to identifying and addressing faults in the system. Table 1 outlines the progression of the DATO (Diverse and Adaptive Test Optimization) strategy over multiple levels of test case selection and group division, highlighting the steps taken to optimize test case effectiveness in fault localization. Here's a detailed explanation of each part of the table:

Level 1: At the initial level (k = 1), the strategy starts with a single test case, t6. The group of statements covered by this test case is denoted as G1, which includes statements {s1, s2, s3, s4, s6, s7, s13}. At this point, no further test cases are selected, and hence no calculation for Mi(ti) is performed. The goal at this stage is to establish the initial group of statements that may contain faults.

Level 2: In level 2 (k = 2), two additional test cases, t3 and t2, are considered. The groups are divided based on their coverage of statements.

Group G2: This group contains statements {s1, s2, s3, s13}, which are covered by both t6 and t3. The metric Mi(ti) is computed for each test case to determine its effectiveness in dividing the statement group. For t1, the metric value is 7, indicating its ability to further partition the group effectively.

Group G3: This group includes statements {s4, s6, s7} covered by t6 but not by t3. Here, t2 has the lowest metric value of 3, suggesting that t2 is less effective in dividing this group compared to other test cases.

The values of Mi(ti) for test cases t3 and t5 indicate their relative effectiveness in dividing the groups further. The metric Mi(ti) helps in selecting the most appropriate test case for optimal group division.

Level 3: At level 3 (k = 3), the group divisions from the previous level are further refined:

Group G4: The statements {s1, s2, s3, s13} are further divided by new test cases. Here, t1 is again selected with a metric value of 7, suggesting it is still effective in dividing this group.

Group G5: This group is empty, meaning no statements from G2 are covered, so the metric Mi(ti) values are computed for test cases t2, t4, and t5. Notably, t2 has a metric value of 5, indicating its effectiveness in dividing the previous groups but less than t4 and t5 in this instance.

Group G6: Contains only statement {s4}, and the test case t4 is selected with a metric value of 7, showing its effectiveness in handling this single statement.

Group G7: Contains statements {s6, s7} with test case t5 having a metric value of 5, reflecting its moderate effectiveness in further division.

Level 4: In level 4 (k = 4), further refinement occurs:

Group G8: With statements {s1, s2, s3, s13}, test case t1 is again selected with a metric value of 7, maintaining its effectiveness in this group.

Group G9: Remains empty, with test case t4 having a metric value of 7, showing its continued effectiveness.

Group G14: Contains {s6} with t5 having a metric value of 5, indicating its effectiveness in this group.

Group G15: Also includes {s6}, similar to G14, but no metric is assigned as the group is already covered by previous test cases.

Level 5: Finally, at level 5 (k = 5):

Group G16: Contains {s1, s2, s3, s13} and test case t1 has a metric value of 6, slightly reduced from previous levels but still significant.

Group G17: Remains empty, and t4 has a metric value of 6, reflecting its effectiveness in dividing the statements.

Table 1. Regression Testing Process of DATO strategy.

k Level	ST	Groups	Groups Elements	To Select Test ti	Mi(ti)
1	{t6}	G1	{s1, s2, s3, s4, s6, s7, s13}		
2	{t6, t3}	G2	{s1, s2, s3, s13}	t1	7
		G3	{s4, s6, s7}	t2	3
				t3	1
				t4	1
				t5	5
3	{t6, t3, t2}	G4	{s1, s2, s3, s13}	t1	7
		G5	{}	t2	5
		G6	{s4}	t4	7
		G7	{s6, s7}	t5	5
4	{t6, t3, t2, t5}	G8	{s1, s2, s3, s13}	t1	7
		G9	{}	t4	7
		G14	{s6}	t5	5
		G15	{s6}		
5	{t6, t3, t2, t5}	G16	{s1, s2, s3, s13}	t1	6
		G17	{}	t4	6

Figure 2 illustrates the application and results of the DATO (Diverse and Adaptive Test Optimization) strategy in the context of fault localization for a program procedure Mid(). It includes two main components: the DATO tree and the suspiciousness metrics. The DATO tree in Figure 2 (a) visually represents the process of test case selection and group division based on the DATO strategy. This tree structure helps to illustrate how the groups of statements in the program are divided and evaluated over different levels of the testing strategy.

Tree Structure: Each node in the tree represents a group of statements (identified by group ids such as G1, G2, etc.), and the tree shows how these groups are further divided as test cases are applied. For instance, at a particular level, if a group of statements is subdivided, it results in the creation of subgroups (e.g., G5 and G6).

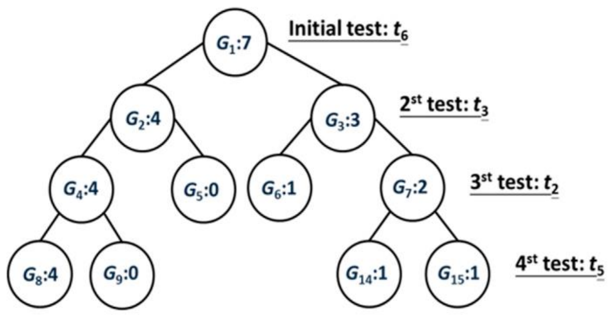
Group Division: The left of the colon in each group id denotes the group id, and the right of the colon indicates the number of the subgroup within that group. If the number of statements in a group is less than 2, as in groups G5 and G6, the group cannot be divided further. Despite this, the groups are still numbered to maintain consistency in the representation and to ensure that all potential subgroups are accounted for in the analysis.

Test Case Selection: The final selected test cases in the strategy are {t6, t3, t2, t5}. These test cases were chosen based on their ability to effectively cover the program's statements and divide the groups for optimal fault localization. The tree demonstrates how the selection of these test cases impacts the grouping and division of statements across various levels.

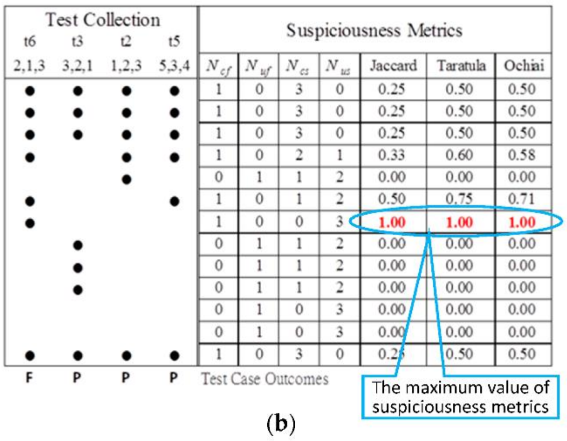
Figure 2 (b) shows the suspiciousness scores for individual statements using various metrics—Jaccard, Tarantula, and Ochiai. These metrics are employed to quantify the likelihood that a particular statement is faulty based on the test results and spectra collected from the selected test cases.

Suspiciousness Scores: The scores reflect how likely each statement is to be the cause of faults. For instance, the statement s7 is highlighted as having a high suspiciousness score, indicating that it is likely faulty according to the metrics used. The scores are calculated using the test cases {t6, t3, t2, t5}, which were selected by the DATO strategy.

Effectiveness of Selection: The comparison in Figure 2b shows that even if test case t4 had been selected instead of t3 at level 2, the overall effectiveness of fault localization would remain the same. This is because the fault localization effectiveness, in terms of identifying high suspiciousness statements, is not significantly impacted by this alternative selection. The metrics' results demonstrate that both the set {t6, t3, t2, t5} and {t6, t4, t2, t5} yield similar outcomes in identifying potential faults.



(a)



Test Collection				Suspiciousness Metrics						
t6	t3	t2	t5							
2,1,3	3,2,1	1,2,3	5,3,4	N_{cf}	N_{uf}	N_{cs}	N_{us}	Jaccard	Taratula	Ochiai
●	●	●	●	1	0	3	0	0.25	0.50	0.50
●	●	●	●	1	0	3	0	0.25	0.50	0.50
●	●	●	●	1	0	3	0	0.25	0.50	0.50
●		●	●	1	0	2	1	0.33	0.60	0.58
		●		0	1	1	2	0.00	0.00	0.00
●			●	1	0	1	2	0.50	0.75	0.71
●				1	0	0	3	**1.00**	**1.00**	**1.00**
	●			0	1	1	2	0.00	0.00	0.00
	●			0	1	1	2	0.00	0.00	0.00
	●			0	1	1	2	0.00	0.00	0.00
				0	1	0	3	0.00	0.00	0.00
				0	1	0	3	0.00	0.00	0.00
●	●	●	●	1	0	3	0	0.25	0.50	0.50
F	P	P	P	Test Case Outcomes						

The maximum value of suspiciousness metrics

(b)

Figure 2. An example program Mid() with (a) DATO tree and (b) suspiciousness metrics.

5. Conclusion

In conclusion, the DATO strategy has demonstrated significant advantages in optimizing test case selection for SBFL. By prioritizing test cases based on the diversity of their execution traces, DATO effectively reduces the number of test cases required for fault localization while maintaining near-equivalent effectiveness compared to using the entire test suite. The experiments conducted on faulty versions of the Siemens and UNIX programs reveal that DATO can significantly minimize the testing effort—using fewer test cases than previously employed methods—without compromising fault localization accuracy. This highlights DATO's potential in enhancing testing efficiency, especially in resource-constrained environments. For future research, several avenues can be explored to further improve and extend the applicability of DATO. Investigating its performance across a broader range of software systems and fault types could provide deeper insights into its generalizability. Additionally, integrating DATO with other fault localization techniques and adapting it to dynamic software environments where test cases and code frequently evolve might yield even greater benefits. Exploring automated tools that incorporate DATO into continuous integration pipelines could also enhance real-time fault detection and resolution.

References

Abreu, R.; Zoeteweij, P.; Golsteijn, R.; Van Gemund, A.J. A practical evaluation of spectrum-based fault localization. *J. Syst. Softw.* 2009, *82*, 1780–1792.

Zakari, A.; Abdullahi, S.; Shagari, N.M.; Tambawal, A.B.; Shanono, N.M.; Maitama, J.Z.; Abdulrahman, S.M. Spectrum-based fault localization techniques application on multiple-fault programs: A review. *Glob. J. Comput. Sci. Technol.* 2020, *20*, G2.

Wen, M.; Chen, J.; Tian, Y.; Wu, R.; Hao, D.; Han, S.; Cheung, S.C. Historical spectrum based fault localization. *IEEE Trans. Softw. Eng.* 2019, *47*, 2348–2368.

Wong, W.E.; Debroy, V.; Ruizhi, G.; Yihao, L. The DStar method for effective software fault localization. *IEEE Trans. Reliab.* 2013, *63*, 290–308.

Godefroid, P.; Klarlund, N.; Sen, K. Dart: Directed automated random testing. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, Chicago, IL, USA, 12–15 June 2005; pp. 213–223.

Ma, E.; Fu, X.; Wang, X. Scalable path search for automated test case generation. *Electronics* 2022, *11*, 727.

Sen, K.; Marinov, D.; Agha, G. Cute: A concolic unit testing engine for c. In Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Lisbon, Portugal, 5–9 September 2005; pp. 263–272.

Yu, Y.; Jones, J.A.; Harrold, M.J. An empirical study of the effects of test-suite reduction on fault localization. In Proceedings of the 30th International Conference on Software Engineering (ICSE), Leipzig, Germany, 10–18 May 2008; pp. 201–210.

Hao, D.; Xie, T.; Zhang, L.; Wang, X.; Sun, J.; Mei, H. Test input reduction for result inspection to facilitate fault localization. *J. Autom. Softw. Eng.* 2010, *17*, 5–31.

Xia, X.; Gong, L.; Le, T.-B.; Lo, D.; Jiang, L.; Zhang, H. Diversity maximization speedup for localizing faults in single-fault and multi-fault programs. *J. Autom. Softw. Eng.* 2016, *23*, 43–75.

Wu, Y.; Liu, Y.; Wang, W.; Li, Z.; Chen, X.; Doyle, P. Theoretical Analysis and Empirical Study on the Impact of Coincidental Correct Test Cases in Multiple Fault Localization. *IEEE Trans. Reliab.* 2022, *71*, 830–849.

Renieris, M.; Reiss, S. Fault localization with nearest neighbor queries. In Proceedings of the IEEE/ACM 18th International Conference on Automated Software Engineering (ASE), Montreal, QC, Canada, 6–10 October 2003; pp. 141–154.

Bajaj, A.; Sangwan, O.P. A systematic literature review of test case prioritization using genetic algorithms. *IEEE Access* 2019, *7*, 126355–126375.

Zakari, A.; Lee, S.P.; Abreu, R.; Ahmed, B.H.; Rasheed, R.A. Multiple fault localization of software programs: A systematic literature review. *Inf. Softw. Technol.* 2020, *124*, 106312.

GCC, the GNU Compiler Collection. Available online: https://gcc.gnu.org/ (accessed on 31 December 2020).

Nayak, S.; Kumar, C.; Tripathi, S. Analytic hierarchy process-based regression test case prioritization technique enhancing the fault detection rate. *Soft Comput.* 2021, *26*, 6953–6968.

Lei, Y.; Xie, H.; Zhang, T.; Yan, M.; Xu, Z.; Sun, C. Feature-FL: Feature-Based Fault Localization. *IEEE Trans. Reliab.* 2021, *71*, 264–283.

Xie, X.; Chen, T.Y.; Kuo, F.C.; Xu, B. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Trans. Softw. Eng. Meth.* 2013, *22,* 1–40.

Lucia, L.; Lo, D.; Jiang, L.; Thung, F.; Budi, A. Extended comprehensive study of association measures for fault localization. *J. Softw. Evol. Proc.* 2014, *26*, 172–219.

Campos, J.; Abreu, R.; Fraser, G.; d'Amorim, M. Entropy-based test generation for improved fault localization. In Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), Silicon Valley, CA, USA, 11–15 November 2013; pp. 257–267.