Theoretical Analysis and Empirical Study on the Impact of Coincidental Correct Test Cases in Multiple Fault Localization

Yonghao Wu, Yong Liu, *Member, IEEE*, Weibo Wang, Zheng Li, Xiang Chen, *Member, IEEE*, and Paul Doyle

***Abstract*—To improve the efficiency of the fault localization process, different automatic fault localization approaches have been proposed. Among these approaches, the spectrum-based fault localization (SBFL) approach has been widely used and studied due to its lightweight and high effectiveness. However, while the existence of coincidental correct (CC) test cases can influence the usefulness of SBFL in single-fault programs, their influence on multiple fault programs has not been thoroughly investigated. Therefore, in this article, we conduct a theoretical analysis and an empirical study to investigate the effect of CC test cases on multiple fault localization. The theoretical analysis is based on a suspiciousness calculation formula of SBFL, which divides CC test cases into three categories (specific, irrelevant, and unspecific) according to their association with a specific faulty statement. Following this analysis, we conduct an empirical study on two well-known open-source repositories (SIR and Defects4J), and the experimental results verify the correctness of our theoretical analysis. Specifically, reducing the number of specific CC test cases for a faulty statement can improve or maintain fault localization accuracy, while eliminating irrelevant CC test cases can have a negative effect. Finally, we design a CC test case identification solution based on the isolation-based multiple fault localization approach and demonstrate its effectiveness via a simulation experiment.**

***Index Terms*—Coincidental correct test case, empirical study, fault localization, multiple fault, theoretical analysis.**

I. Introduction

As SOFTWARE complexity increases, software testing activities involving fault localization and program repair have significantly grown, becoming more critical and demanding [1]. When bugs occur in software, developers must identify the exact location of the faults, and then repair them. The process of identifying the localization of faulty statements is commonly referred to as fault localization, and it has been recognized as one of the most expensive and laborious steps in the entire debugging activity [2]–[4], as it is frequently challenging to comprehend the complex internal logic of the program under test and reason about possible fault locations. Therefore, researchers developed automatic fault localization approaches to improve the efficiency of fault localization, and then saved the time for software delivery [2], [5]. By analyzing the characteristics of programs, such techniques aim to identify the location of potentially faulty statements, which can assist developers in determining the exact location of the fault.

While extensive automatic fault localization approaches have been developed, the spectrum-based fault localization (SBFL) approach has received considerable attention due to the fact that it is a lightweight but highly accurate technique [1], [3], [5]–[12]. To calculate the suspiciousness of each program statement, SBFL uses the execution trace of test cases (i.e., test case coverage information) and the execution results of test cases. A statement with a higher suspiciousness score is more likely to be a faulty statement, allowing developers to check program statements from high to low, according to the suspiciousness rank list. Ideally, developers want to localize the faulty statement by only checking a few statements.

However, previous research [10], [13] usually assumed that a faulty program only contains a single faulty and that each test case, which executes faulty statements, must produce incorrect output. The previous assumptions did not conform to the characteristics of the practical application scenario. Recent research indicates that a faulty program may contain multiple faults (i.e., multiple fault localization) and coincidental correct (CC) test cases. Where a CC test case refers to a test case that executes faulty statements, but the output is the same as expected [14]. Therefore, the existence of multiple fault programs and CC test cases breaks the assumptions of previous fault localization studies, thereby casting doubt on their validity.

As a result, recent studies have explored more practical scenarios. Several studies investigated how multiple faults impact

Manuscript received November 28, 2021; revised March 25, 2022; accepted April 2, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61902015, Grant 61872026, Grant and 61672085, and in part by Nantong Application Research Plan under Grant JC2021124. Associate Editor: Y. Dai. *(Corresponding authors: Zheng Li; Yong Liu.)*

Yonghao Wu, Yong Liu, Weibo Wang, and Zheng Li are with the College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100013, China (e-mail: appmlk@outlook.com; lyong@mail.buct.edu.cn; 1095719690@qq.com; lizheng@mail.buct.edu.cn).

Xiang Chen is with the School of Information Science and Technology, Nantong University, Nantong 226007, China (e-mail: xchencs@ntu.edu.cn).

Paul Doyle is with the School of Computer Science, Technological University Dublin, D07 EWV4 Dublin, Ireland (e-mail: paul.doyle@tudublin.ie).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2022.3165126.

Digital Object Identifier 10.1109/TR.2022.3165126

traditional SBFL approaches [15], [16], which were typically evaluated in faulty programs with only one faulty statement. Then, they propose new SBFL approaches to increase the accuracy of multiple fault localization [9], [13], [17]. Similarly, researchers have designed many strategies to address the negative impact of CC test cases on fault localization [18]–[25]. However, existing research usually studies CC test cases and multiple faults separately. That is, studies on multiple fault localization did not consider CC test cases, whereas studies on CC test cases also did not consider multiple fault programs.

Therefore, to analyze how multiple faults and CC test cases interfere with the traditional SBFL approach, we first perform a theoretical analysis on the effect of CC test cases on the accuracy of multiple fault localization. In particular, faulty statements in a faulty program may be executed by different failed test cases, and different test cases may be related to different faulty statements [7], [23], [24], [26], [27], so we simulate different CC test case identification situations in our empirical study. Specifically, for a specific faulty statement, all CC test cases can be classified into three categories: 1) specific; 2) unspecific; and 3) irrelevant. While reducing the number of specific CC test cases for a specific faulty statement can improve or maintain the faulty statement's fault localization accuracy, while eliminating irrelevant CC test cases can have a negative effect on fault localization accuracy.

We also conduct an empirical study on two well-known open-source repositories: 1) software-artifact infrastructure repository (SIR) [28]; and 2) Defects4J [29], to verify the correctness of our theoretical analysis. Specifically, inspired by previous multiple fault localization approaches, we design a CC test case identification solution based on the isolation-based multiple fault localization approach. The fault localization results demonstrate that using the Clean strategy (a detailed description can be found in Section IV-C) to remove all CC test cases that executed the faulty statement associated with clusters generated by isolation methods can improve fault localization accuracy.

In summary, to our best knowledge, the main contributions of our study can be summarized as follows:

1) We investigate the association between CC test cases and faults in multiple fault localization, including performing an analysis of test case impact by eliminating different association types on fault localization accuracy. According to the relationship between test cases and faulty statements, we divide all CC test cases into three different categories: a) specific; b) irrelevant; and c) unspecific. We find that the accuracy of multiple fault localization can be improved by reducing the number of specific CC test cases for a faulty statement, but cannot be improved by removing irrelevant CC test cases.
2) We conduct an empirical study to analyze how the number of different types of CC test cases influence the accuracy of multiple fault localization. Our empirical results verify our theoretical analysis. We find that reducing the number of specific CC test cases for a single faulty statement can improve fault localization accuracy.
3) Motivated by previous multiple fault localization approaches, we design a CC test case identification solution based on the isolation-based multiple fault localization approach in multiple fault programs, and we demonstrate its effectiveness through a simulation-based experiment.

Reproduction Package: To facilitate replication of our study, we have shared our source code and data on a GitHub repository.[1]

The rest of this article is organized as follows. Section II provides an overview of fault localization techniques (including coincidental correctness), a summary of relevant related studies, and describes the motivation for our study. Section III introduces the theoretical analysis of CC test cases' negative impact on multiple fault localization. Section IV introduces the research questions, subject programs, and performance metrics used in our experimental study. Section V first discusses the experimental setup and the detailed result analysis. Then, we summarize the practical guidelines based on the results of our theoretical analysis and empirical studies. Section VI discusses observations through the experiment. Section VII discusses the main threats to validity of our empirical studies. Finally, Section VIII concludes this article.

II. Background and Related Work

A. Fault Localization

The traditional approach for fault localization is to debug the program by inserting breakpoints, but this manual procedure requires considerable developer effort and time. To increase the code quality while decreasing software bug debugging time, researchers have proposed different automated fault localization approaches. These approaches enable developers to rapidly localize faults and minimize the time required to review large volumes of code. To date, there have been numerous achievements in the field of automated fault localization research domain, which we categorize as follows:

1) The first category is mutation-based fault localization (MBFL), which analyzes the behavioral similarity between the faulty program and the mutated program seeded with artificial faults. MBFL uses a suspiciousness formula to calculate the fault probability of each statement to assist the developer in localizing the faulty statements [4]. Although MBFL can achieve a high fault localization accuracy, it requires the execution of all test cases on a large number of mutated programs, which has a huge mutation execution cost [30]. The usage of MBFL in practical software fault localization is limited due to this disadvantage.
2) The second category is information retrieval-based fault localization (IRFL), which extracts information from texts (such as bug reports) to find the location of faults in a program. For example, Zhou *et al.* [31] proposed an information retrieval fault localization tool called BugLocator, which is based on the revised vector space model. This tool uses information from similar fixed bugs to adjust the ranking to help localize the bug-related files. Saha *et al.* [32] argued that for information retrieval-based bug

[1][Online]. Available: https://github.com/appmlk/CC-in-multiple-fault-program-empirical-study.

TABLE I
SUSPICIOUSNESS FORMULAS

Name	Formula
$Jaccard$ [35]	$\mathcal{S}us(s) = \frac{fail(s)}{totalfail+pass(s)}$
$Ochiai$ [36]	$\mathcal{S}us(s) = \frac{fail(s)}{\sqrt{totalfail \times (fail(s)+pass(s))}}$
$Dstar^*$ [37]	$\mathcal{S}us(s) = \frac{fail(s)^*}{pass(s)+(totalfail-fail(s))}$

localization, it is better to extract information when the information is structured according to the code structure. Based on this finding, they proposed the method bug localization using information retrieval, and found that their proposed method performs better than BugLocator. However, Wang *et al.* [33] evaluated the effectiveness of IRFL, and found a dilemma with IRFL techniques. Specifically, the fault localization performance of IRFL is highly dependent on the often inconsistent quality of bug reports, which may limit the accuracy achieved using IRFL approaches.

3) The third category is SBFL, which is based on the assumption that faulty statements are executed by a larger number of failed test cases than that of passed test cases. SBFL suspiciousness formulas were designed based on this assumption to calculate the probability of each statement being a faulty statement using test case coverage information and execution results [5], [34].
Commonly used suspiciousness formulas in SBFL include Jaccard [35], Ochiai [36], and Dstar [37], which are given in Table I. In this table, fail(s) and pass(s) represent the number of times the statement(s) were executed by failed and passed test cases, respectively, while totalfail and totalpass represent the number of failed and passed test cases, respectively. The possible values for sus(s) in Jaccard [35], Ochiai [36], and Dstar* [37] is between 0 and 1. It should be noted that when Wong *et al.* [37] proposed the Dstar formula, a sample was used, as an example, to illustrate the influence of exponent * on the performance of fault localization. In their example, when exponent * was set to 3, the suspiciousness rank of the faulty statement can exceed most of the correct statements and be tied for first place. Furthermore, Wong *et al.* [37] found that the performance of Dstar increases with the increase of exponent *. Therefore, in the rest of this article, we use Dstar^3 to denote Dstar*. According to these three formulas in Table I, the suspiciousness value corresponding to each statement can be calculated, and the higher the suspiciousness for a statement, the more likely it contains a fault.
Table II provides an example of coverage information and execution results. This example illustrates a program segment with five statements (s_1, s_2, s_3, s_4, and s_5) and five test cases (t_1, t_2, t_3, t_4, and t_5). Table II also includes the statement coverage and execution results, with bugs occurring in statements s_2 and s_4. Black bullets (•) are used to indicate that the statement is covered by the corresponding test case, and blank space indicates that the statement is not covered by the corresponding test case. Test case execution results are represented by **P** or **F**, which indicates that the test case has passed or failed, respectively. Columns 7–9 of Table II give the suspiciousness of each statement calculated by three formulas: 1) Jaccard; 2) Ochiai; and 3) Dstar^3. The fault localization result shows that the suspiciousness value of the faulty statement s_2 calculated by Jaccard, Ochiai, and Dstar^3 are 0.67, 0.82, and 8, respectively, which are ranked in order of suspiciousness in most cases.

4) The fourth category is slicing-based fault localization [15]. Slicing a program can generate a subprogram based on user-specified slicing criteria. By performing program analysis, techniques, such as the data flow equation calculation and dependency analysis, can be used to extract subprograms that may affect the slicing criteria. Slicing-based fault localization works on the principle that if a failed test case discovers that a variable's value is different from the expected value, the fault is likely to occur in the slice containing this variable, and thus, the search space for fault localization can be limited to this slice without checking the whole program. However, this method relies on data flow analysis for fault localization and has relatively low accuracy.

In addition, to the abovementioned four commonly used approaches, there are also other types of approaches, such as model-based strategies [38], genetic algorithm-based strategies [13], and neural network-based strategies [39], which can be applied to automated fault localization.

Among these approaches, the SBFL approach has been widely used because of its lightweight algorithm and promising performance [5], [6], [13], [16], [25], [34], [40], [41]. Current studies on SBFL have achieved promising fault localization accuracy in single-fault programs. However, the aforementioned fundamental assumptions of the traditional SBFL cannot directly apply to multiple fault programs. Because a correct statement may be covered by several failed test cases caused by distinct faulty statements in a multiple fault program, the correct statement will be covered by more failed test cases than other faulty statements. In this situation, the correct statement is more suspicious than the faulty statement, which can result inefficient fault localization when utilizing a traditional SBFL approach on a multiple fault program.

B. Multiple Fault Localization

A multiple fault program refers to a program under test that contains multiple faulty statements. However, the key assumptions of traditional SBFL are not applicable to multiple fault programs.

In a multiple fault program, the coyesrrect statement may be covered by more failed test cases than other faulty statements. In

TABLE II
SIMPLE EXAMPLE FOR THE PROCESS OF SBFL

s_j	t_1	t_2	t_3	t_4	t_5	Origin			Clean t_3			Relabel t_3		
						$Jaccard$	$Ochiai$	$Dstar^3$	$Jaccard$	$Ochiai$	$Dstar^3$	$Jaccard$	$Ochiai$	$Dstar^3$
s_1	●	●	●	●	●	0.6	0.77	13.5	0.75	0.87	27.00	0.80	0.89	64.00
s_2(faulty)	●	●				0.67	0.82	8	0.67	0.82	8.00	0.50	0.71	4.00
s_3		●	●	●	●	0.4	0.58	2.67	0.50	0.67	4.00	0.60	0.75	13.50
s_4(faulty)		●	●	●		0.5	0.67	4	0.67	0.82	8.00	0.75	0.87	27.00
s_5	●				●	0.25	0.41	0.33	0.25	0.41	0.33	0.20	0.35	0.25
Result	F	F	P	F	P	-	-	-	-	-	-	-	-	-

this case, the correct statement has a higher degree of suspiciousness than the faulty statement, which leads to the problem of low fault localization efficiency when using a traditional SBFL approach on multiple fault programs.

To improve the efficiency of multiple fault localization, researchers have proposed different approaches [42]. Previous multiple fault localization approaches can be mainly divided into three categories: 1) debugging approaches that localize one fault at a time; 2) debugging approaches that localize multiple faults at a time; and 3) parallel debugging fault localization approaches.

One-fault-at-a-time approaches are proposed based on the assumption that developers will localize all faulty statements in a multiple fault program through multiple iterations [6], [43], [44]. One iteration of the developer's debugging process will only localize and correct one program fault when employing a one-fault-at-a-time debugging approach. For example, suppose the faulty program contains two faulty statements. In this case, the developer must execute two fault localization iterations to localize all faulty statements, where multiple fault localization can be achieved via a variety of debugging techniques, such as SBFL, MBFL, or slicing-based fault localization, in each iteration.

Multiple fault-at-a-time approaches are proposed based on the assumption that developers localize faulty statements in multiple fault programs in a single iteration [9], [13], [45]. With the proper strategy, debugging many faults at once can efficiently localize all or most of the faulty statements in a program, which can improve debugging efficiency and decrease software delivery time. Researchers currently proposed methods to improve the accuracy of the ranking of faulty statements using the suspiciousness metric, thus decreasing the workload of fault localization in a single iteration; or using neural network methods to localize multiple faulty statements at once. For example, by formalizing the multiple fault localization problem into a search problem, Zheng *et al.* [13] proposed a genetic algorithm-based multiple fault localization framework.

Parallel debugging approaches assume that multiple developers can work on the same program concurrently, which can reduce the time for fault localization [6], [46]. The isolation-based fault localization approach is a form of parallel debugging that achieves a high fault localization accuracy [16], [47]. Therefore, many recent studies used isolation-based fault localization techniques to localize multiple faults [17], [18], [34], [48].

When using isolation-based fault localization, the developer divides a complex debugging task into smaller debugging assignments, which can allow multiple developers to work on multiple tasks simultaneously. Specifically, the failed test cases caused by the same faults are grouped into the same cluster, thereby generating a cluster centered on the failure, that is, the failed test cases within the same cluster are related to the same faulty statement. Then, using a single cluster of failed test cases and some or all of the passed test cases, a failure-centric suspiciousness ranking list can be created. By inspecting the code according to this ranking, developers can quickly identify the faulty statement corresponding to one cluster. The key challenge of isolation-based fault localization is determining the proper clusters for the failed test cases. Since researchers do not know the number of faulty statements in advance, it is impossible to identify the correct number of clusters or to assign initial centers to these clusters, which has resulted in the development of numerous unsupervised clustering-based approaches for isolating failed test cases. For example, current research has leveraged several clustering algorithms, such as hierarchical clustering [6], k-means clustering [49], and k-medoids clustering [16], to analyze the coverage information and isolate the failed test cases, thus completing the process of parallel debugging [6], [16], [17], [49], [50].

As illustrated by the abovementioned three strategies, most multiple fault localization approaches aim to increase accuracy and efficiency in locating multiple faults. For example, the one-fault-at-a-time strategy attempts to optimize the efficiency of the first faulty statement localized during each iteration. The multiple-fault-at-a-time strategy improves the efficiency of each faulty statement localized during a single iteration in a multiple fault program. While parallel debugging based techniques focus on increasing the number of faulty statements that can be located in each iteration, they decrease the number of iterations required.

Related studies on multiple fault localization have made progress and achieved many promising results. However, the existence of CC test cases has not been thoroughly investigated. According to recent studies [19], [20], CC test cases are prevalent in the software testing process and have a negative effect on SBFL. As a result, detecting and processing CC tests effectively can help SBFL run more efficiently in multiple fault programs.

C. Coincidental Correct Test Cases

Coincidental correctness (CC) refers to the situation during the execution of a test case where the program enters an abnormal state due to a fault in execution, but the output of the program

is consistent with the expected output. The test case that causes this situation is called a CC test case [51]. Specifically, when the fault in a program needs to be detected by a test case, the following three conditions must be met [10], [19], [23], [25], [52].

1) The faulty code causes the program to be infected and enter an abnormal state.
2) The abnormal state continues to propagate and affect the subsequent execution of the program.
3) These exceptions and subsequent fault propagation affect the output of the program, causing the actual output to be inconsistent with the expected output.

However, there are some exceptions in the execution of test cases as follows.

1) The test case executes the faulty code in the program and enters an abnormal state, but these exceptions have no effect on the subsequent program execution, that is, the test case only meets the first condition.
2) The test case executes the program and enters an abnormal state. These abnormal states continue to propagate and cause the program to affect subsequent execution, but these exceptions are not reflected in the output. The actual output is exactly the same as the expected output, that is, the test case only meets the first and second conditions, but does not meet the third condition.

In both exception cases, the test cases will lead to coincidental correct behavior, and those test cases will be referred to as CC test cases. Taking Table II as an example, s_4 is a statement that contains a fault, t_3 executes the faulty statement s_4, but no failure can be detected, so t_3 is determined to be a CC test case.

Experimental results of recent studies have shown that the existence of CC test cases can significantly affect the effect of fault localization. For example, Ball *et al.* [22] claimed that according to their fault localization approach, CC test cases caused three of their fifteen experimental programs to fail to localize the fault. Wang *et al.* [25] used the Tarantula approach to conduct empirical research on CC test cases. Their study showed that the accuracy of fault localization is related to the proportion of CC test cases in the test suite. As the proportion of CC test cases in the test suite increases, the accuracy of the fault localization approach based on SBFL can be significantly reduced. These previous studies showed that the existence of CC test cases in the test suite has a negative impact on the accuracy of fault localization.

Since CC test cases are widespread in programs and seriously affect the accuracy of fault localization, many previous studies aimed to improve the efficiency of SBFL methods by identifying CC test cases [18]–[22]. Many of the approaches for CC test case identification are based on the assumption that CC test cases are similar to the failed test cases. This assumption holds in the single-fault case because each CC test case executes the same faulty statement, and thus, these CC test cases will share a characteristic similarity with the failed test cases that also execute the faulty statement. For example, Masri and Assi [23] proposed an approach to determine a fixed percentage of test cases, as CC test cases based on the possibility of CC.

Although previously proposed methods are effective in improving fault localization accuracy in the single-fault cases, these proposed CC identification and processing strategies for the single-fault hypothesis do not apply to multiple fault cases. Taking the program given in Table II as an example, t_3 is considered to be a CC test case because of its identical coverage path to t_4. Columns 10–15 in Table II give the suspiciousness values for each statement after applying the two common CC processing strategies of Clean t_3, i.e., removing t_3 from the test case set, and Relabel t_3, i.e., correcting the execution result of t_3 from pass to fail. As given in Table II, the fault localization accuracy shows a decreasing trend after processing the CC test cases due to the complex impact of CC on multiple fault cases. Specifically, the faulty statement s_2, which is originally ranked highest in suspiciousness in most cases, is no longer easily found, while the correct statement s_1 gets the highest suspiciousness rank. The abovementioned example demonstrates that the proposed CC identification and processing approach for single-fault cases may have a negative effect on multiple fault cases.

D. Related Work

1) Research on Multiple Fault Localization: Fault localization is an active research field in the domain of software engineering research, with different fault localization techniques proposed based on different scenarios. Similar to our study, some previous studies also consider multiple fault programs and coincidental correct test cases during fault localization.

Localizing one-fault-at-a-time is one of the most commonly used debugging strategies in the field of multiple fault localization research. It has been widely used by developers due to its simple and easy-to-operate algorithm. For example, Jones *et al.* [6] proposed a multiple fault localization approach, which first uses all failed test cases to localize and fix one faulty statement at a time. Next, the test cases are re-executed to use all failed test cases to localize and fix the next faulty statement, with this process iterating until all faults are fixed. Subsequently, Kim *et al.* [53] proposed a variable-based fault localization (VFL) method to solve the problem of the SBFL method having poor fault localization ability when the test cases cover similar information. VFL guides the ranking of program statements by identifying the suspiciousness variables in the program. Experimental results based on the Defects4J dataset have shown that VFL has better localization performance and has the advantage of being lightweight and more scalable, and can be combined with other methods to further improve the fault localization performance. Later, Kim *et al.* [53] demonstrated that their proposed method can achieve better fault localization performance when dealing with multiple fault programs by using the one-fault-at-a-time strategy. The one-fault-at-a-time localization strategy has been widely studied for its promising applications, but by only localizing one fault at a time, it can negatively impact the developers' bug fix rate.

Compared to one-fault-at-a-time and parallel debugging approaches, multiple faults-at-a-time strategy-based fault localization approaches are less well-studied. These approaches are usually designed with the help of machine learning algorithms

to localize multiple faulty statements at once. For example, Wong *et al.* [54] proposed a multiple fault localization method based on radial basis functions by combining neural network algorithms. This method treats the coverage paths of test cases as feature vectors and uses the execution results of the test cases as the output data of the neural network. Next, virtual coverage paths are constructed for each program statement for the training data. Where each virtual coverage path only covers one statement, the output of the neural network is the suspiciousness value of that statement. For example, if a virtual coverage path is constructed for the third statement in a program, which contains five statements, the path is "00100". The virtual path is fed into the neural network as a feature vector, and the output value is the suspiciousness value of the third statement. The experimental results show that this method outperforms multiple fault localization baselines by using formulas, such as Crosstab or Tarantula. Zheng *et al.* [13] proposed a multiple fault localization framework based on a genetic algorithm, which encodes program statements into chromosomes and converts the multiple fault localization problems into the search problem, so that their proposed framework can localize multiple faulty statements simultaneously. Researchers have proposed some approaches from the multiple faults-at-a-time perspective, and these approaches can achieve better performance than the baselines. However, the effectiveness of neural network-based approaches is highly dependent on how the model is constructed, and the cost in terms of time grows significantly as the program size increases, which can limit the general applicability of this kind of approach.

For fault localization approaches based on parallel debugging, the effectiveness of these approaches depends on the quality of the clustering results. For these approaches, the failed test cases are divided into different clusters, and it is expected that the failed test cases related to each fault can be divided into the same cluster, so as to reduce the impact of multiple faults on the effectiveness of the SBFL approach. Jones *et al.* [6] proposed an isolation-based fault localization approach, whose purpose is to reduce the cost of manual debugging in the case of multiple faults. In isolation-based fault localization, failed test cases are divided into multiple clusters that target different faults. Therefore, each fault-focusing cluster will be combined with passed test cases to get a specialized test suite that targets a single fault. Finally, different clusters will be allocated to multiple developers to debug programs in parallel. Their experimental results have shown that the isolation-based fault localization approach could effectively help developers reduce the cost of debugging. Gao and Wong [16] proposed the MSeer approach to address the challenging problem of determining the number of faulty statements in a faulty program. This approach can estimate the number of faulty statements in advance, followed by the use of fuzzy clustering to cluster the failed test cases, and classify the cluster to localize the fault. Their experimental results have shown that MSeer can perform better than the one-fault-at-a-time approaches and the method proposed by Jones *et al.* [6] in terms of both fault localization efficiency and effectiveness. This effective isolation-based strategy for converting multiple fault localization problems into single-fault localization problems also inspired our study to design a practical approach for handling complex CC problems under multiple fault situations. Further details can be found in Section V-D.

2) Eliminate the Impact of CC Test Cases: Many studies attempt to alleviate the negative impact of CC test cases in multiple fault programs. For example, Bandyopadhyay [7] proposed two approaches that predict CC test cases and used the prediction results to improve the effectiveness of SBFL. In the first approach, he assigned lower weights to the CC test cases to have a lower impact on the final suspiciousness calculation. In the second approach, he removeed all CC test cases and used the reduced test cases set to calculate suspiciousness. Miao *et al.* [21] further proposed a clustering-based approach. They used the k-means clustering algorithm to cluster test cases with similar coverage information into the same cluster. However, these approaches are mainly proposed for single-fault programs.

Moreover, Hofer [55] applied the SBFL in circuit and spreadsheet debugging. This study aimed to evaluate the influence of the removal of the actual coincidental correctness, and the results showed that the ranking of actual faults never worsens when only considering single-fault programs; however, the fault localization performance can decrease in multiple fault programs. Liu *et al.* [56] proposed a weighted fuzzy classification approach, i.e., fuzzy weighted K-nearest neighbor (FW-KNN), to identify and manipulate CC test cases. This approach is based on the fuzzy KNN classification algorithm, which uses failed test cases as the training data and passed test cases as the test data for classification. The passed test cases with higher similarity to the failed test cases will be classified as CC test cases. Experimental results in single-fault and multifault programs show that the application of the FW-KNN approach can effectively improve fault localization performance when compared with traditional SBFL. Assi *et al.* [57] improved the effectiveness of test suite reduction (TSR), test case prioritization (TCP), and SBFL by using substate profiling. They conducted experiments on the Defects4J dataset showing that the performance of SBFL could be further improved after removing CC test cases. Consequently, Sabbaghi *et al.* [58] proposed a fuzzy expert system to model the CC identification process, and proposed a fuzzy CC identification approach (FCCI). A set of fuzzy rules, which successfully correlate the CC identification elements, is used to assess the CC likelihood of the passed test cases. They evaluated FCCI on 17 open-source programs. Their experimental results suggested that FCCI could enhance the CC identification and representative SBFL technology. The abovementioned approaches are effective in improving the accuracy of fault localization after dealing with CC test cases, but their study ignores the relationship between CC and specific faults. That is, it is possible that a passed test case may be CC for one of the faulty statements in the program, but not for another faulty statement.

Subsequently, Assi *et al.* [59] conducted an empirical study on the impact of CC test cases on three tasks: 1) TSR; 2) TCP; and 3) SBFL. Regarding the SBFL task, by conducting experiments on the Defeats4J dataset, they found that the negative impact of CC test cases on SBFL was highly correlated with the used evaluation metrics. However, when all CC test cases are removed, the evaluation results of EXAM and TOP-N evaluation metrics

decrease in most cases. Later, Assi *et al.* [59] mentioned that the presence of multiple faults might lead to the fault localization accuracy decreasing after clearing CC in their threat analysis. Similar to our study, Assi *et al.* [59] also considered that clearing all CC test cases in a multiple fault program may have a negative impact on fault localization. Therefore, based on their study, we further investigated the impact of clearing CC test cases with different relationships to faulty statements on fault localization through theoretical analysis and empirical studies.

3) Research on Fault Interactions: Furthermore, existing studies attempted to investigate the essential differences between multiple fault and single-fault programs. Multiple faults in a program may interact in various ways, resulting in unexpected behaviors that do not exist in single-fault programs. New fault behaviors exist in various types and have unpredictable negative effects on traditional single-fault localization techniques. Debroy and Wang [60] provided a detailed analysis of fault interactions and proposed the fault masking phenomenon in an empirical study. Fault masking is an extreme case of mixed types of faults interacting with each other. When fault-masking occurs, the failed test case associated with a program fault loses association with that fault due to fault obfuscation (i.e., the fault can no longer be executed), while this part of the fault is simultaneously associated with another program, i.e., the former is completely masked by the latter, when the former fault cannot be detected by the test case. Furthermore, Digiuseppe and Jones [61] conducted an in-depth study of fault interactions and classified multiple fault behaviors into following four types.

1) *Fault Synergy:* Increased failures detected by test cases compared to single-fault programs.
2) *Fault Obfuscation:* Decreased failures detected by test cases compared to single-fault programs.
3) *Pass/Fail Independence:* The faults' interaction did not change the pass/fail status of the program.
4) *Multitype:* Both fault synergy and fault obfuscation occur.

Moreover, Li *et al.* [62] focused on three types of fault interactions: 1) pass/fail independence; 2) fault masking; and 3) fault obfuscation, and explored what factors caused their interactions. By focusing on and analyzing the two fault programs, they found that fault disturbances usually involve the same variables, which implies that the occurrence of fault disturbances is somewhat conditional. Based on the abovementioned study, we found that the complexity of handling CC test cases in multiple fault programs is due to the existence of complex fault interactions in multiple fault programs. However, until now, the identification and processing of faults interactions in multiple fault programs has remained a challenging problem [63], [64]. Therefore, we aim to analyze and study the CC problem in multiple fault programs, and then design a feasible solution in this study.

III. Theoretical Analysis on the Impact of CC Test Cases on SBFL

In this section, we conduct a theoretical analysis to explore the influence of CC test cases on multiple fault localization in SBFL. Here, we mainly show the analytical approach using the Jaccard formula. However, an analysis of other classical SBFL formulas, such as Ochiai and Dstar, can also achieve the same conclusion. Specifically, we followed the theoretical analysis model of Zhang *et al.* [65], who demonstrated that cloning of failed test cases could effectively improve fault localization accuracy using the SBFL formula.

A. Problem Formulation

Given a faulty program P and a set of test cases T, the faulty program P is composed of program statements S, which include one or more faulty statements S_F ($S_F \in S$). The test cases T are classified into passed test cases T_p, failed test cases T_f, and CC test cases T_c, i.e., $T = T_p \cup T_f \cup T_c$.

As discussed in Section II, most multiple fault localization approaches aim to increase the accuracy of each faulty statement's localization. Therefore, we conduct our theoretical study by analyzing the influence of the number of CC test cases on a single faulty statement. However, CC test cases on multiple fault programs will present more complex behaviors, as each CC test case may be caused by different or even numerous faulty statements. Thus, for each faulty statement, we first need to categorize all CC test cases according to their relevance to the faulty statement.

According to whether specific faulty statements are covered, each failed or CC test case can be related to one or more faulty statements. For instance, if the test case t ($t \in (T_f \cup T_c)$) covers the faulty statement S_a ($S_a \in S_F$), the failed test case t is related to the statement S_a, which can be expressed as $\text{cov}(S_a, t) = 1$. If the test case t does not cover the faulty statement S_a, $\text{cov}(S_a, t) = 0$, then for a specific faulty statement S_a, all test cases can be classified into three categories: 1) specific; 2) unspecific; and 3) irrelevant. Specifically, these three different categories can be formally defined as follows.

1) *Specific:* CC test cases are only related to faulty statement S_a, and not related to other faulty statements. These type of test cases can be denoted as $T_{cs}(S_a)$, where $T_{cs}(S_a) = \{t \in T_c | \text{cov}(S_a, t) = 1\} - \{t \in T_c | \text{cov}(f, t) = 1, f \in F, f \neq S_a\}$.
2) *Irrelevant:* CC test cases are not related to faulty statement S_a. These type of test cases can be denoted as $T_{ci}(S_a)$, where $T_{ci}(S_a) = \{t \in T_c | \text{cov}(S_a, t) = 0\}$.
3) *Unspecific:* CC test cases are not only related to faulty statement S_a, but also related to other faulty statements. These type of test cases can be denoted as $T_{cu}(S_a)$, where $T_{cu}(S_a) = \{t \in T_c | \text{cov}(S_a, t) = 1\} \cap \{t \in T_c | \text{cov}(f, t) = 1, f \in F, f \neq S_a\}$.

In our study, we aim to investigate whether the SBFL accuracy can be improved by reducing the number of test cases in each of the abovementioned three categories.

B. Problem Simplification

In this section, we simplify our comparison of fault localization accuracy to a suspiciousness rank comparison. Because the suspiciousness rank list directly reflects the efforts associated with fault localization, a higher rank of faulty statements indicates a more effective approach for fault localization. As a result, when the number of CC test cases decreases, we can

observe the changing trends of SBFL accuracy by evaluating the suspiciousness rank changing trends of faulty statements.

To clearly illustrate the changing trend of the suspiciousness rank when eliminating CC test cases, we classify all statements into three subsets (i.e., S_H, S_E, and S_L) as follows.

1) S_H consists of all statements with suspiciousness higher than the faulty statement S_a, that is, $S_H = \{S_i \in \mathrm{S} \,|\, \mathrm{Sus}(S_i) > \mathrm{Sus}(S_a)\}$.
2) S_E consists of all statements with suspiciousness equal to the faulty statement S_a, that is, $S_E = \{S_i \in \mathrm{S} \,|\, \mathrm{Sus}(S_i) = \mathrm{Sus}(S_a)\}$.
3) S_L consists of all statements with suspiciousness lower than the faulty statement S_a, that is, $S_L = \{S_i \in \mathrm{S} \,|\, \mathrm{Sus}(S_i) < \mathrm{Sus}(S_a)\}$.

Where $\mathrm{Sus}(S_a)$ means the suspiciousness value of the statement S_a, calculated by the suspiciousness formula. We can find that increasing the size of S_L leads to promising results because the suspiciousness rank of faulty statements increases. In contrast, increasing the size of S_H leads to unsatisfactory results.

Then, we design a comparison function T based on the Jaccard formula to estimate the changing trend of S_H, S_E, and S_L. We let $\mathrm{totalfail} = F$, $\mathrm{totalpass} = P$, $\mathrm{fail}(S_i) = a$ $(a \in [0, F])$, $\mathrm{fail}(S_a) = b$ $(b \in [0, F])$, $\mathrm{pass}(S_i) = c$ $(c \in [0, P])$, and $\mathrm{pass}(S_a) = d$ $(d \in [0, P])$, where c represents the size of C and d represents the size of D. The function T can then be constructed as follows:

$$\begin{aligned} T &= \mathrm{Jaccard}(S_i) - \mathrm{Jaccard}(S_a) \\ &= \frac{\mathrm{fail}(S_i)}{\mathrm{totalfail} + \mathrm{pass}(S_i)} - \frac{\mathrm{fail}(S_a)}{\mathrm{totalfail} + \mathrm{pass}(S_a)} \\ &= \frac{a}{P + c} - \frac{b}{P + d} \\ &= \frac{a(P + d) - b(P + c)}{(P + c)(P + d)}. \end{aligned} \tag{1}$$

When $T < 0$, it means that the suspiciousness of statement S_i is lower than that of the faulty statement S_a, and $S_i \in S_L$. Similarly, $T = 0$ means that $S_i \in S_E$, and $T > 0$ means that $S_i \in S_H$. Given that the value of the denominator in formula (1) must be larger than 0, we can use the numerator $(a(P + d) - b(P + c))$ to simplify the calculation.

Moreover, the statements S_i and S_a may be executed by the same passed test cases. We let this part of the pass test cases be E, i.e., $E = C \cap D$, with e being the length of E, and we can let $d = e + d'$ and $c = e + c'$, so the numerator of formula (1) can be structured as $(a - b)e + (a - b)F + ad' - bc'$.

If there exists S_i, which can make T become equal to or less than 0 from a value larger than 0, after the reduction of CC test cases, we can say that the size of S_L increases and that of S_H decreases, which means the fault localization accuracy is improved. Similarly, if there exists S_i, which makes T become equal to or larger than 0 from less than 0, the changing process leads to lower fault localization accuracy.

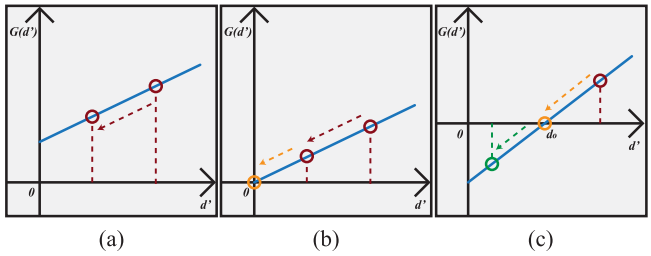

Fig. 1. Function graph of $G(d')$. (a) $(a - b)e + (a - b)F - bc' > 0$. (b) $(a - b)e + (a - b)F - bc' = 0$. (c) $(a - b)e + (a - b)F - bc' < 0$.

C. Theoretical Analysis

In the rest of this section, we analyze the relationship between T and 0 before and after the reduction of the following three types of CC test cases.

1) *Specific CC Test Cases:* When we reduce the number of specific test cases, only the value of d' decreases. Thus, the independent variable is d'. We can then construct the function by using the numerator of T as follows:

$$G(d') = ad' + (a - b)e + (a - b)F - bc' \tag{2}$$

where a must be larger than 0, so $G(d')$ is a monotone increasing function. The possible function graphs of $G(d')$ are shown in Fig. 1.

Fig. 1 shows the following set of possible findings.

a) In Fig. 1(a), the value of $G(d')$ is always larger than 0, which indicates that the statement S_i is always a member of the S_H set.
b) In Fig. 1(b), the value of $G(d')$ will decrease to 0 when $d' = 0$, so the statement S_i may change from the S_H set to the S_E set.
c) In Fig. 1(c), the value of $G(d')$ will decrease to 0 when $d' = d_0$, and $G(d') < 0$ when $d' < d_0$. So, the statement S_i may change from the set S_H to the set S_E or the set S_L, or change from the set S_E to the set S_L.

Therefore, regardless of the type of the function graph, the accuracy of fault localization will maintain or improve after removing specific CC test cases.

2) *Irrelevant CC Test Cases:* While reducing the irrelevant test cases, only the value of c' will decrease, so the independent variable is c'. We can then construct the function based on the numerator of T as follows:

$$G(c') = -bc' + (a - b)e + (a - b)F + ad' \tag{3}$$

where b must be larger than 0, so $G(c')$ is a monotone decreasing function. The possible function graphs of $G(c')$ are shown in Fig. 2.

Fig. 2 shows the following set of possible findings.

a) In the situation shown in Fig. 2(a), the value of $G(c')$ is always less than 0, so the statement S_i always belongs to the S_L set.
b) In the situation shown in Fig. 2(b), the value of $G(c')$ will increase to 0 when $d' = 0$, so the statement S_i may change from the set S_L to the set S_E.

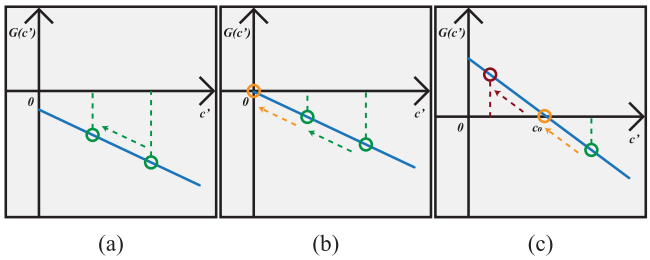

Fig. 2. Function graph of $G(c')$. (a) $ad' + (a-b)e + (a-b)F < 0$. (b) $ad' + (a-b)e + (a-b)F = 0$. (c) $ad' + (a-b)e + (a-b)F > 0$.

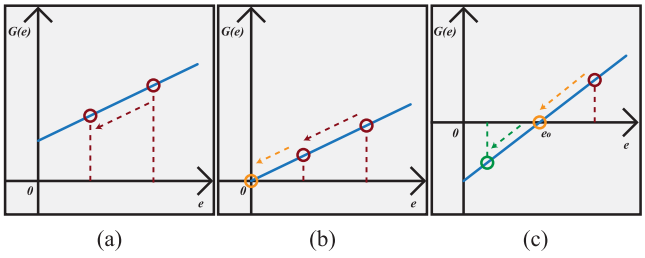

Fig. 3. Function graph of $G(e)$ while $a > b$. (a) $(a-b)F + ad' - bc' > 0$. (b) $(a-b)F + ad' - bc' = 0$. (c) $(a-b)F + ad' - bc' < 0$.

c) In the situation shown in Fig. 2(c), the value of $G(c')$ will increase to 0 when $c' = c_0$, and $G(c') > 0$ when $c' > c_0$. So, the statement S_i may change from the set S_L to the set S_E or the set S_H, or change from the set S_E to the set S_H.

Therefore, after removing the irrelevant CC test cases, the accuracy of fault localization will either maintain or worse.

3) *Unspecific CC Test Cases:* While removing the unspecific test cases, the value of e will decrease. So, the independent variable is e. The function can then be constructed as follows:

$$G(e) = (a-b)e + (a-b)F + ad' - bc' \qquad (4)$$

where the function slope is $(a-b)$, its polarity is not constant. So we will continue our analysis in three situations: a) $a - b > 0$; b) $a - b = 0$; or 3) $a - b < 0$.

When $a - b > 0$, $G(e)$ is a monotone increasing function, the possible function graphs of $G(e)$ are shown in Fig. 3.

As these function graphs have the same shape as Fig. 1, we can draw the same conclusion as that of specific CC test cases. Specifically, if $a - b > 0$, after removing the unspecific test cases, the fault localization accuracy will maintain or improve.

When $a - b > 0$, $G(e)$ is a constant, the possible function graphs of $G(e)$ are shown in Fig. 4.

In Fig. 4, we can find that the reduction of CC test cases cannot impact the value of $G(e)$. Therefore, if $a - b = 0$, the fault localization accuracy will maintain.

When $a - b < 0$, $G(e)$ is a monotone decreasing function, the possible function graphs of $G(e)$ are shown in Fig. 5.

As the function graphs have the same shape as Fig. 2, we can draw the same conclusions as that of irrelevant CC test cases. As a result, if $a - b < 0$, while reducing the unspecific test cases, the fault localization accuracy will maintain or get worse.

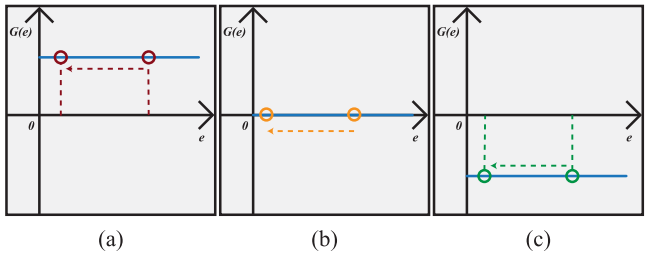

Fig. 4. Function graph of $G(e)$ while $a = b$. (a) $G(e) > 0$. b) $G(e) = 0$. (c) $G(e) < 0$.

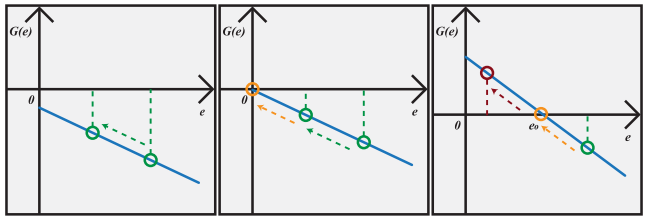

Fig. 5. Function graph of $G(e)$ while $a < b$. (a) $(a-b)F + ad' - bc' < 0$. (b) $(a-b)F + ad' - bc' = 0$. (c) $(a-b)F + ad' - bc' > 0$.

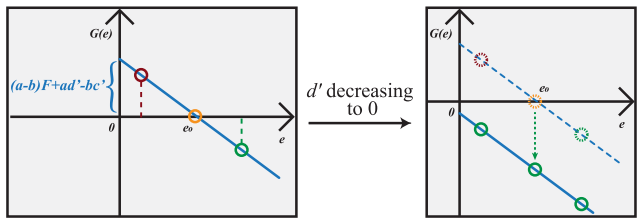

Fig. 6. Change process of $G(e)$ function when d' decreases. (a) $(a-b)F + ad' - bc' > 0$. (b) $(a-b)F + ad' - bc' < 0$.

However, this does not mean that reducing the number of unspecific CC test cases for S_i will irreversibly lead to lower fault localization accuracy. Further research reveals that the lower accuracy of fault localization can be attributed to the incomplete reduction of S_i related CC test cases. If we continue to reduce the S_i related CC test cases until it no longer exists, the accuracy of fault localization would be improved to a better level than when no CC test cases are processed. Fig. 6 intuitively illustrates this process.

Where the left-hand side side of Fig. 6 is a more detailed representation of Fig. 5(c). $(a-b)F + ad' - bc' > 0$ leads to the function graph of $G(e)$ intersecting the y-axis above the x-axis. If we continue to reduce the S_i related CC test cases, the value of d' would decrease. Because $(a-b) < 0$ and $bc' > 0$, if the value of d' decreases to 0, $(a-b)F + ad' - bc' < 0$, and the entire function graph of $G(e)$ would drop below the x-axis, which is shown in the right-hand side side of Fig. 6, then the fault localization accuracy will maintain no matter how the number of unspecific CC test cases changes.

D. Summarization of Theoretical Analysis

Based on the abovementioned rank-oriented theoretical analysis, for each faulty statement of a multiple fault program, we can draw the following conclusions:

1) Reducing the specific CC test cases of this faulty statement can improve or maintain the fault localization accuracy of this faulty statement.
2) Reducing the irrelevant CC test cases of this faulty statement can decrease or maintain the fault localization accuracy of this faulty statement.
3) Reducing the unspecific CC test cases of this faulty statement may lead to unpredictable results, but we can ensure that the fault localization accuracy cannot be decreased by removing more CC test cases related to this faulty statement.

IV. Empirical Study Design

In this section, we conduct an empirical study to analyze the impact of CC test cases on SBFL in faulty programs with multiple faulty statements, and verify the correctness of our theoretical analysis, which is analyzed in Section III.

A. Research Questions

We conduct our empirical study to address the following four research questions.

1) *RQ1: How Prevalent are CC Test Cases in Faulty Programs?* CC test cases can weaken the effectiveness of SBFL [10], [26], [51]. Investigating their prevalence can find the relationship between the distribution of CC test cases and the program faults. In this RQ, we analyze the prevalence of CC test cases in programs that contain multiple faulty statements, which can comprehensively improve our understanding of CC test cases.
2) *RQ2: Can the Proposed CC Identification Approaches for Single-Fault Programs Increase the Accuracy of Fault Localization in Multiple Fault Programs?* Previous studies on CC test cases have proved both theoretically and experimentally that decreasing the number of CC test cases in single-fault programs can significantly improve fault localization accuracy [18], [19], [21], which is why traditional CC test case handling approaches usually tend to eliminate all CC test cases. However, we demonstrate theoretically in this study that reducing CC test cases in a multiple faults situation may have a negative effect on fault localization accuracy. In this RQ, we aim to eliminate all CC test cases from the original test suite, which can help us to investigate the effect of traditional CC test case handling strategies on the accuracy of multiple fault localization.
3) *RQ3: How Does SBFL Accuracy Change When Three Categories of CC Test Cases are Removed in Multiple Fault Programs?* CC test cases can generate more complex results due to multiple faulty statements [25], [27], [66], and we define and analyze these CC test cases in detail in Section III. In this RQ, we aim to verify the correctness of our theoretical analysis, specifically looking at the impact of the reduction of specific, unspecific, and irrelevant CC test cases on fault localization accuracy. The result of this process will facilitate further insights and suggestions for better CC test case identification and processing.
4) *RQ4: How do CC Test Cases Affect SBFL Accuracy When Applying Isolation-Based Fault Localization Approaches in Multiple Fault Programs?* Isolation-based fault localization approaches were proposed to solve multiple fault programs [17], [34], [48]. However, few approaches attempt to address CC test cases in multiple fault localization. In this RQ, we aim to investigate the influence of CC test cases on isolation-based fault localization approaches, which can examine how the current fault localization approaches perform in the presence of CC test cases.

B. Subject Programs

We collected 24 open-source programs as our experimental subjects. These 24 programs include eight programs developed using the C programming language (i.e., Grep, Print tokens, Print tokens2, Schedule, Schedule2, Replace, Tcas, Tot info, and Sed) and 15 large-scale Java programs with real faults (i.e., Chart, Cli, Codec, Compress, Csv, Gson, JacksonCore, JacksonDatabind, JacksonXml, Jsoup, JxPath, Lang, Math, Mockito, and Time). All the C programs adopted in our empirical study can be obtained from SIR,[2] while Java programs can be downloaded from the Defects4J GitHub repository.[3] These programs have been widely used in previous multiple fault localization studies [4], [6], [16], [49], [56], which can alleviate the external threat of our empirical results.

Table III gives the characteristics of the subject programs used in our study. The first and second columns of Table III present the program's categories and name, respectively, and the third column presents the number of multiple fault programs for each subject program, where the multiple fault program versions contain multiple faulty statements.

SIR provides the correct version of each subject program as well as several faulty versions with seeded faults [28]. However, because SIR supplies only a limited number of faulty versions (generally no more than ten versions per program), we need to generate more faulty versions for our empirical study. Specifically, we manually inject faulty statements into correct programs to increase the amount of single-fault program versions, and then randomly combine these faulty statements from single-fault program versions to obtain sufficient multiple fault program versions. This strategy of generating faulty programs has been widely used in previous studies [16], [45], [67], which can also alleviate the external threat of our empirical study. Moreover, to show the generalization of our empirical results, we also consider the Defects4j repository in our experiments. Defects4J is an open-source Java program repository that contains real-world defects. Martinez *et al.* [29] developed Defects4J, which has been widely used to evaluate automated fault localization and repair approaches. Defects4J is regarded as

[2][Online]. Available: https://sir.csc.ncsu.edu/php/.
[3][Online]. Available: https://github.com/rjust/defects4j.

TABLE III
CHARACTERISTICS OF SUBJECT PROGRAMS

Repository	Object	# Multiple Fault Versions
SIR	Grep	832
	Print tokens	854
	Print tokens2	850
	Schedule	838
	Schedule2	868
	Replace	873
	Tcas	896
	Tot info	889
	Sed	447
Defects4J	Chart	5
	Cli	15
	Codec	4
	Compress	19
	Csv	2
	Gson	3
	JacksonCore	11
	JacksonDatabind	32
	JacksonXml	2
	Jsoup	4
	JxPath	11
	Lang	18
	Math	43
	Mockito	5
	Time	7

the largest peer-reviewed well-organized structured database of real Java defects. As a result, Defects4J programs have become the primary assessment programs in the field of fault localization in recent years. In total, we obtain 7528 multiple fault program versions in our experimental study.

C. Coincidental Correct Test Case Elimination Strategies

To accurately reflect the influence of CC test cases and draw conclusions for our research questions, we use labeled test cases to simulate the CC test case identification process. Therefore, the accuracy of the CC test case identification results presented in our study can be guaranteed.

Following this, we eliminate all of the identified CC test cases. Recent research has proposed a variety of strategies for dealing with the identified CC test cases. For example, Miao *et al.* [21] proposed two distinct strategies for manipulating CC test cases: 1) Clean; and 2) Relabel. Specifically, the Clean strategy removes identified CC test cases from the original test suite, whereas the Relabel strategy changes the execution results of the identified CC test cases from pass to fail.

When calculating the suspiciousness of S_i via the Jaccard formula presented in Table I, the Jaccard formula with two CC dealing strategies: 1) Clean; and 2) Relabel, can be, respectively, defined by the following:

$$\text{Sus}_{\text{Clean}}(S_i) = \frac{\text{fail}(S_i)}{\text{total fail} + \text{pass}(S_i) - \text{cc}(S_i)} \quad (5)$$

$$\text{Sus}_{\text{Relabel}}(S_i) = \frac{\text{fail}(S_i) + \text{cc}(S_i)}{\text{total fail} + \text{cc}(S_i) + \text{pass}(S_i) - \text{cc}(S_i)} \quad (6)$$

where $\text{cc}(S_i)$ refers to the number of CC test cases that execute the statement S_i and cc refers to the total number of CC test cases.

Due to the reduction process of CC test cases, the Clean strategy is the most consistent technique with our theoretical analysis. Therefore, we compare the fault localization results based on the original test suite with the test suite after processing with the Clean strategy in our empirical study.

D. Evaluation Metrics

We use the metrics EXAM and TOP-N to evaluate the accuracy of fault localization. The details of these two evaluation metrics are introduced as follows.

1) EXAM: EXAM has been commonly used to evaluate the accuracy of fault localization [6], [68], which ranks the efforts of locating the first faulty statement from the suspiciousness rank list. A lower EXAM value means that fewer statements need to be checked before localizing the real faulty statement, and the corresponding fault localization approach is more accurate. The EXAM value can be calculated as follows:

$$\text{EXAM} = \frac{\text{rank of the faulty statement}}{\text{number of the executable statements}} \quad (7)$$

where the numerator is the rank of faulty statement in the suspiciousness ranking list and the denominator is the total number of executable statements that need to be checked. However, when other existing statements share the same suspiciousness value as the faulty statement, the numerator of EXAM cannot be determined directly. To address this issue, recent research [16], [68] has presented three approaches for calculating EXAM in this situation. Let the faulty statement be S_a, the number of statements with a higher suspiciousness value than S_a be A, and the number of statements with the same suspiciousness value as S_a be B. We can then calculate the value of EXAM as follows:

a) *Best-Case Scenario:* In this scenario, the developers can find the faulty statements as fast as possible. This means that the faulty statement can be located first within several statements that share the same suspiciousness value. Then, the value of EXAM is $A + 1$.

b) *Worst-Case Scenario:* In this scenario, the developers will find faulty statements as slowly as possible. This means that the faulty statement will be finally located among several statements that share the same suspiciousness value. Then, the value of EXAM is $A + B$.

c) *Average-Case Scenario:* It is more likely in practice that programmers will find faulty statements within a period of time, which is somewhere between the best and worst case scenario. Thus, it is better to report the average localization effectiveness. Then, the value of EXAM is $\frac{(A+1)+(A+B)}{2}$.

To simulate the real-world software debugging scenario, we use the average-case scenario to calculate the EXAM value.

2) TOP-N: The metric TOP-N is also a widely used metric in the field of fault localization [16], [22], [25], which indicates the number of faulty statements that can be identified when less

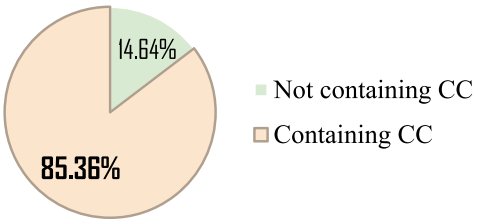

Fig. 7. Prevalence of coincidental correct test cases.

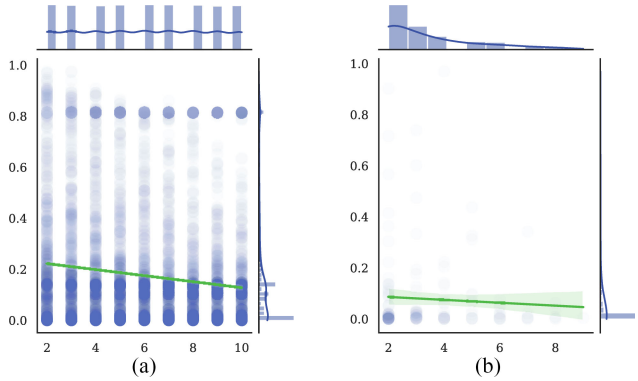

Fig. 8. Percentage of CC test cases generated by programs with differing number of faulty statements. (a) SIR. (b) Defects4J.

than N statements are examined. The higher the TOP-N value, the fewer statements developers must check when localizing faults, which indicates that the corresponding fault localization approach is more effective. Note that this metric is very important in practice, since developers usually only inspect top-ranked statements (e.g., over 70% of developers only check TOP-5 ranked statements [69]).

V. Result Analysis

In this section, we analyze the results for each of our four research questions and provide guidelines for potential future studies.

A. RQ1: How Prevalent are CC Test Cases in Faulty Programs?

Fig. 7 shows the percentage of our 7528 programs contains CC test cases. In this figure, we can find that 6426 (i.e., 85.36%) faulty programs contain CC test cases. In particular, 6262 (i.e., 85.23%) of the 7347 faulty programs from SIR contain CC test cases, while 164 (i.e., 90.61%) of the 181 faulty programs from Defects4J contain CC test cases.

Based on the abovementioned statistical results, most multiple fault programs contain CC test cases. Therefore, the existence of CC test cases is clearly common in multiple fault programs.

Moreover, we analyze the correlation between the number of CC test cases and the number of faulty statements. Our analysis is performed by investigating the percentage of CC test cases generated by programs with differing amounts of faulty statements. The results are shown in Fig. 8.

The joint and marginal histograms plots are shown in Fig. 8, where Fig. 8(a) shows the data from the SIR dataset, while Fig. 8(b) shows the data from the Defects4J dataset. The horizontal axis indicates the number of faulty statements contained in the program, while the vertical axis indicates the proportion of CC test cases generated by the program. The bars at the top and right-hand side of each subfigures indicate the distribution of the corresponding samples. For example, the height of the bars at the top of Fig. 8(a) are basically the same, which indicates that the number of programs containing different faulty statements in the SIR dataset of this study are basically the same. Moreover, the maximum value of the right-hand side bar in Fig. 8(a) is located at a y-axis value, which is close to zero. This indicates that the percentage of CC test cases generated by most faulty programs in the SIR dataset is very close to zero. Finally, the color block in the middle of each subfigure indicates the degree of sample enrichment at the location corresponding to the horizontal and vertical coordinates, and the darker the color, the higher the number of samples at that location. Furthermore, the straight line in Fig. 8 is the fitted regression line, which can show the trend for the proportion of CC test cases, as the number of faulty statements in the program increases.

As illustrated in Fig. 8(a) and (b), a large number of samples are tightly grouped, which are close to the horizontal axis. This indicates that regardless of the number of faulty statements in programs, the CC test cases contained in the faulty programs constitute a small fraction of the total test cases. According to our data analysis, we can find that 3490 (i.e., 46.36%) of the 7528 multiple faulty programs result in less than 10% CC test cases of all test cases. Meanwhile, according to the trend of the regression line, we can find that the percentage of CC test cases in the program tends to decrease slightly, as the number of faulty statements in the program increases.

Answer to RQ1: The experimental results indicate that more than 85% of multiple fault programs contain CC test cases during the execution of their corresponding test suite, which demonstrates that CC problems are widespread in multiple fault programs. Moreover, regardless of the number of faulty statements in programs, the percentage of contained CC test cases is often between 0% and 20% of the whole test suite. Finally, the percentage of CC test cases tends to decrease slightly, as the number of faulty statements in the program increases.

B. RQ2: Can the Proposed CC Identification Approaches for Single-Fault Programs Increase the Accuracy of Fault Localization in Multiple Fault Programs?

To answer this RQ, we conduct the CC test case processing strategy, proposed for single-fault programs, on all 7528 fault programs. Specifically, we adopt the strategy of removing all CC test cases, and then compare the fault localization accuracy before and after removing these CC test cases.

Table IV gives the detailed experimental results. The first column lists the various considered values of EXAM, and the other columns are the percentages of faulty statements whose EXAM is smaller than that of the corresponding EXAM value when associated SBFL approaches are applied with different

TABLE IV
FAULT LOCALIZATION ACCURACY COMPARISON OF THE TRADITIONAL CLEAN STRATEGY IN TERMS OF EXAM METRIC

EXAM (%)	Percentage of Faulty Statements (%)											
	SIR						Defects4J					
	Ochiai		*Dstar*		*Jaccard*		*Ochiai*		*Dstar*		*Jaccard*	
	Origin	Clean	Origin	Clean	Origin	Clean	Origin	Clean	Origin	Clean	Origin	Clean
1	**6.4**	*5.8*	**6.0**	*5.5*	**6.5**	*5.8*	**7.7**	*7.5*	**7.5**	*7.3*	**7.8**	*7.4*
5	**19.9**	*18.8*	**19.3**	*18.4*	**19.9**	*18.7*	21.7	25.1	21.7	24.2	22.2	24.9
10	**37.0**	*36.4*	**36.6**	*36.0*	**36.9**	*36.2*	33.0	32.3	31.5	30.1	33.3	32.0
15	48.5	50.3	48.2	50.0	48.5	50.2	37.7	38.7	36.9	37.0	37.8	38.6
20	59.2	59.6	59.2	59.3	58.8	59.5	42.2	42.7	41.5	41.1	42.6	42.7
30	88.4	88.3	88.2	88.1	88.4	88.2	47.5	52.0	46.9	51.2	47.7	52.0
40	98.6	98.6	98.6	98.6	98.6	98.6	52.9	56.1	52.8	56.1	53.0	56.1
50	99.6	99.6	99.6	99.6	99.6	99.6	57.6	58.4	57.6	58.4	57.7	58.4
60	99.6	99.6	99.6	99.6	99.6	99.6	74.2	73.4	74.2	73.4	74.3	73.4
70	99.9	99.9	99.9	99.9	99.9	99.9	86.7	86.9	86.7	86.9	86.8	86.9
80	100	100	100	100	100	100	94.4	94.3	94.4	94.3	94.4	94.3
90	100	100	100	100	100	100	99.2	99.2	99.2	99.2	99.2	99.2
100	100	100	100	100	100	100	100	100	100	100	100	100

The bold font refers to positive results and italic font refers to negative results.

test cases and suspiciousness formulas. For example, for the first row, its EXAM value is less than or equal to 1%. This means that when considering the top 1% statements, 6.4% of all faulty statements can be localized under SBFL using the Ochiai formula and the original test suite. When cleaning all CC test cases, the corresponding percentage is 5.8%. Obviously, for each column, the higher the percentage is, the more accurate the corresponding fault localization approach is.

In Table IV, we can find when only the top 1% of statements are checked, employing the Clean strategy to remove all CC test cases can result in a decrease of fault localization accuracy. This indicates that after removing all CC test cases, the developers can identify fewer faulty statements.

We highlight the differences only at the top of the Table IV, with green (bold) denoting positive results and orange (italic) denoting negative results. Because the results near the top (especially the 1%), according to a developer-oriented survey [69] outlined in Section IV-D2, are more decisive. Especially, for large-scale projects, such as Defeats4J, with over 10 000 lines of code, where each 1% decrease can result in over 100 lines of code, which do not needed to be checked.

To show the results in Table IV more visually, we use a plot illustrated in Fig. 9 to examine the growth trend of Table IV.

In Fig. 9, x-axis denotes the value of the metric EXAM and y-axis indicates the percentages of faulty statements whose EXAM value is smaller than that of the corresponding EXAM value under different test cases and repositories. A higher value of the y-axis at the same x-axis position means better fault localization accuracy. It should be noted that the formula used in Fig. 9 is Jaccard, which can perform best before cleaning, as given in Table IV.

In Fig. 9, we find, when all CC test cases are removed, the amount of statements (especially the top 2%) checked by the developers before localizing the faulty statement decreases. This result is consistent with that in Table IV.

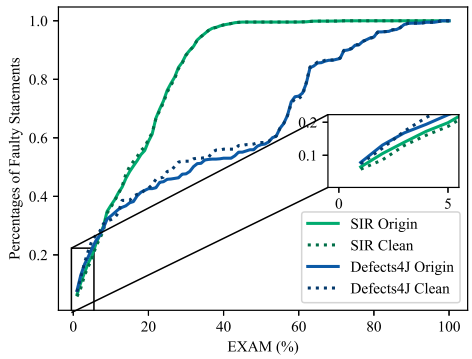

Fig. 9. Fault localization accuracy of using the traditional Clean strategy in terms of EXAM metric.

Moreover, we employ the metric TOP-N to illustrate the experimental results described in this section. The results are presented in Table V, where each row corresponds to a unique N value, which requires checking the corresponding number of statements. Here, the better result for each row in Table V is highlighted in green. Taking the results shown in the third column of the third row as an example, we can find that when checking the first statement in the suspiciousness ranking list generated by the Ochiai formula, 339 of faults can be localized in SIR programs.

In Table V, we can find that when all CC test cases are removed via the Clean strategy, the value of TOP-1–TOP-10 becomes significantly smaller, which means that the number of faulty statements that can be found by detecting the same statement becomes smaller (i.e., decreasing fault localization accuracy).

Answer to RQ2: The proposed strategy for handling CC test cases tends to remove all CC test cases for single-fault programs. However, our experimental results demonstrate that this strategy is not applicable to multiple fault programs. When all CC test cases are removed from the original test suite, the accuracy of fault localization decreases.

TABLE V
FAULT LOCALIZATION ACCURACY COMPARISON OF THE TRADITIONAL CLEAN STRATEGY IN TERMS OF THE METRIC TOP-N

Subject	Formula	TOP-1		TOP-3		TOP-5		TOP-10	
		Origin	Clean	Origin	Clean	Origin	Clean	Origin	Clean
SIR	$Ochiai$	**339**	165	**1295**	1097	**2328**	1811	**4104**	3010
	$Dstar$	**323**	158	**1221**	1065	**2180**	1753	**3944**	2929
	$Jaccard$	**345**	164	**1309**	1094	**2347**	1806	**4081**	3002
Defects4J	$Ochiai$	**23**	4	**107**	81	**157**	126	**251**	251
	$Dstar$	**24**	4	**107**	79	**156**	125	**250**	240
	$Jaccard$	**23**	4	**108**	81	**156**	126	**252**	251
ALL	$Ochiai$	**362**	169	**1402**	1178	**2485**	1937	**4355**	3261
	$Dstar$	**347**	162	**1328**	1144	**2336**	1878	**4194**	3169
	$Jaccard$	**368**	168	**1417**	1175	**2503**	1932	**4333**	3253

The bold font refers to positive results and italic font refers to negative results.

TABLE VI
COMPARISON OF THE FAULT LOCALIZATION ACCURACY OF THE CLEAN STRATEGY ON THREE DIFFERENT TYPES OF CC TEST CASES IN TERMS OF THE METRIC EXAM

$EXAM$ (%)	Percentage of Faulty Statements (%)							
	Origin		Clean					
			Specific		Unspecific		Irrelevant	
	SIR	Defects4J	SIR	Defects4J	SIR	Defects4J	SIR	Defects4J
1	6.5	7.8	**6.7**	**8.2**	**7.0**	**9.2**	*5.2*	*6.6*
5	19.9	22.2	19.9	**22.9**	**20.9**	**24.6**	*18.0*	*21.5*
10	36.9	33.3	36.9	**33.4**	**37.6**	**36.6**	*35.4*	*30.5*
15	48.5	37.8	48.4	**38.3**	**49.3**	**39.8**	48.6	*34.9*
20	58.8	42.6	58.7	**43.0**	**59.3**	**44.5**	59.1	*39.5*
30	88.4	47.7	88.3	**47.8**	**88.5**	**49.0**	88.1	48.8
40	98.6	53.0	98.6	53.0	98.6	**54.2**	98.6	54.0
50	99.6	57.7	99.6	57.7	99.6	**58.4**	99.6	57.3
60	99.6	74.3	99.6	74.3	99.6	74.3	99.6	73.4
70	99.9	86.8	99.9	86.8	99.9	86.8	99.9	86.9
80	100	94.4	100	94.4	100	94.4	100	94.3
90	100	99.2	100	99.2	100	99.2	100	99.2
100	100	100	100	100	100	100	100	100

The bold font refers to positive results and italic font refers to negative results.

C. RQ3: How Does SBFL Accuracy Change When Three Categories of CC Test Cases are Removed in Multiple Fault Programs?

To verify the correctness of the theoretical analysis in Section III, we perform the Clean strategy of three types of CC test cases on all programs and observe the accuracy of fault localization.

Note that in each program, the faulty statement, which has the highest suspiciousness in the original circumstance, is used as the target faulty statement S_a for the purpose of identifying and removing the three types of CC test cases in this experiment. Table VI gives the experimental findings. The first column contains the various values considered for EXAM. The second and third columns show the corresponding results when the original test suite is used for fault localization; and the fourth–fifth/sixth–seventh/eighth–ninth columns contain the average value after removing the corresponding specific/irrelevant/unspecific test cases of the target faulty statement S_a in each program.

As given in Table VI, we find the experimental results are consistent with the conclusions of our theoretical analysis. Specifically, from the fourth and fifth columns, we find that the number of localized faulty statements increases when checking statements after removing the specific CC test cases, while the value of number of localized faulty statements decreases after removing the irrelevant CC test cases. When removing the unspecific CC test case, the fault localization accuracy also changes for the better in general, which is also consistent with one of the conclusions in our theoretical analysis.

In Table VII, we give the results of employing the TOP-N metric to assess the effect of removing three categories of CC test cases. Each row in Table VII is highlighted in green for positive results when compared to the original test suite and in red for negative results. We can observe that the value of TOP-5 increases after removing the specific or unspecific CC test cases, while the value of TOP-5 decreases after removing the irrelevant CC test cases.

Answer to RQ3: Our empirical results verify the correctness of our theoretical analysis. Specifically, for any faulty statement in a multiple fault program, removing specific CC test cases can help to improve the fault localization accuracy, while removing irrelevant CC test cases has the negative influence.

TABLE VII
FAULT LOCALIZATION ACCURACY COMPARISON OF THE CLEAN STRATEGY ON THREE DIFFERENT TYPES OF CC TEST CASES IN TERMS OF THE METRIC TOP-N

Subject	Formula	TOP-5			
		Origin	Clean		
			Specific	Unspecific	Irrelevant
SIR	$Ochiai$	2328	**2400**	**2595**	*1637*
	$Dstar$	2180	**2268**	**2431**	*1563*
	$Jaccard$	2347	**2419**	**2610**	*1630*
Defects4J	$Ochiai$	157	**170**	**175**	*125*
	$Dstar$	156	**168**	**173**	*128*
	$Jaccard$	156	**167**	**177**	*127*
ALL	$Ochiai$	2485	**2570**	**2770**	*1762*
	$Dstar$	2336	**2436**	**2604**	*1691*
	$Jaccard$	2503	**2586**	**2787**	*1757*

The bold font refers to positive results and italic font refers to negative results.

D. RQ4: How Do CC Test Cases Affect SBFL Accuracy When Employing Isolation-Based Fault Localization Approaches in Multiple Fault Programs?

Although removing specific CC test cases can improve fault localization accuracy for any faulty statement in a multiple fault program, previous CC test case identification approaches do not directly identify specific CC test cases for a specific faulty statement in a multiple fault program. As described in Section II-C, most of the existing CC test case identification approaches are based on the assumption that CC test cases are more similar to failed test cases, which holds in for single-fault programs because each CC test case in the single-fault program must execute the same faulty statement, and thus, these CC test cases will achieve a similar characteristic to the failed test cases that also execute the faulty statement. However, this assumption does not hold in the multiple fault scenarios because different CC test cases may execute different faulty statements in the multiple fault program. For example, if a CC test case does not execute the faulty statement S_a, its characteristic may not be similar to the failed test case that executes the statement S_a. Therefore, in the multiple fault program, a CC test case identified via the CC identification approach proposed for single-fault programs cannot determine that which of the faulty statements in the program is executed, thus, it is not possible to know that which type of CC test case (i.e., specific, irrelevant, or unspecific) has been identified.

However, the isolation-based multiple fault localization approaches provide a solution to this problem. As described in Section II-B, the isolation-based multiple fault localization approaches divide all the failed test cases into multiple clusters. Ideally, the failed test cases within each cluster are caused by a single faulty statement [70]. That is, in a multiple fault program containing the faulty statement S_a, if all test cases in a cluster execute the statement S_a, it is more likely that the CC test cases determined by the failed test cases in that cluster also execute the statement S_a. According to the experimental results of RQ3, removing this kind of test case can improve the fault localization accuracy of the statement S_a. Repeating the abovementioned steps for all clusters obtained by the isolation-based multiple fault localization approach; we can ideally improve the localization accuracy of all faulty statements.

TABLE VIII
FAULT LOCALIZATION ACCURACY COMPARISON OF THE CC TEST CASE CLEAN STRATEGY UNDER ISOLATION-BASED FAULT LOCALIZATION IN TERMS OF THE METRIC EXAM

$EXAM$ (%)	Percentage of Faulty Statements (%)			
	SIR		Defects4J	
	Isolation	Isolation &Clean	Isolation	Isolation &Clean
1	**7.7**	**8.6**	**8.0**	**9.9**
5	**20.6**	**21.6**	21.6	**24.4**
10	36.2	36.7	30.6	**34.2**
15	46.5	47.2	35.2	**37.6**
20	56.9	57.1	39.8	**41.6**
30	88.1	88.1	44.6	**45.4**
40	98.6	98.6	49.2	**50.5**
50	99.6	99.6	53.0	**53.8**
60	99.6	99.6	74.8	74.8
70	99.9	99.9	87.3	87.3
80	100	100	94.7	94.7
90	100	100	99.4	99.4
100	100	100	100	100

The bold font refers to positive results and italic font refers to negative results.

To verify the effectiveness of this conjecture, we simulate the process of dividing the failed test cases of the programs under test, where the failed test cases within each class cluster are caused by a single faulty statement. Then, for each cluster, we consider all the passed test cases along with the failed test cases within this cluster. This is the process of original isolation-based multiple fault localization, which served as the control group for this experiment. Next, we remove the CC test cases that execute the corresponding faulty statements of each cluster, and use the passed test cases after removing CC test cases for fault localization together with the failed test cases in the corresponding cluster, and these results of fault localization were used as the experimental group. The final results are given in Table VIII. In this table, the second/fourth columns give the results only after isolation, and the third/fifth columns give the results after isolation and cleaning CC test cases that execute the corresponding faulty statements of each cluster.

As given in the second and fourth columns of Table VIII, developers may discover more faulty statements after isolation when only checking the first few statements, in comparison to the original condition described in Table VI. Moreover, as found by the third and fifth columns of Table VIII, the accuracy of fault localization can be further improved significantly after the relevant CC test cases were removed. Similarly, to evaluate the effect on high-accuracy fault localization results, Table IX gives the results of this experiment in terms of the metric TOP-N.

As given in Table IX, the TOP-5 values in the fourth column are all larger than those in the third column, which indicates that the accuracy of fault localization may be significantly improved by applying the isolation-based multiple fault localization approach. Moreover, the values in the fifth column are all larger than the values in the fourth column, which indicates that the

TABLE IX
FAULT LOCALIZATION ACCURACY COMPARISON OF THE CC TEST CASE CLEAN STRATEGY UNDER ISOLATION-BASED FAULT LOCALIZATION IN TERMS OF TOP-N METRIC

Subject	Formula	TOP-5		
		Origin	Isolation	Isolation & Clean
SIR	*Ochiai*	2328	**2837**	**3209**
	Dstar	2180	**2714**	**3067**
	Jaccard	2347	**2857**	**3252**
Defects4J	*Ochiai*	157	**165**	**193**
	Dstar	156	**166**	**193**
	Jaccard	156	**166**	**194**
ALL	*Ochiai*	2485	**3002**	**3402**
	Dstar	2336	**2880**	**3260**
	Jaccard	2503	**3023**	**3446**

The bold font refers to positive results and italic font refers to negative results.

accuracy of fault localization can be further improved by applying both the Clean strategy and the isolation-based multiple fault localization approach.

Answer to RQ4: Our experimental results show that using both the isolation-based multiple fault localization approach and the Clean strategy, to remove all CC test cases that executed the faulty statement associated with the cluster, can ideally improve fault localization accuracy.

E. Practical Guidelines

In this section, we provide practical guidelines for future studies. Specifically, if we want to improve the fault localization accuracy of a specific faulty statement of multiple fault programs the following hold.

1) *We should avoid removing irrelevant CC test cases associated with this faulty statement*, since our theoretical analysis shows that reducing irrelevant CC test cases may have a negative effect on the accuracy of fault localization for the corresponding faulty statement. Furthermore, the results in RQ3 verify our conclusions by theoretical analysis. When irrelevant CC test cases associated with each faulty statement are removed, the accuracy of fault localization for both artificial and real faults in SIR and Defects4J decreases.
2) *We should consider combining the isolation-based multiple fault localization approach and the traditional CC test case identification strategy to solve the CC problem in multiple fault programs*, as the traditional CC test case identification assumptions proposed for single-fault programs do not hold for multiple fault programs.

Therefore, in RQ4, we propose a new CC test case identification and processing strategy based on the existing isolation-based multiple fault localization approach. The simulation results demonstrate that our strategy can effectively increase fault localization accuracy.

VI. DISCUSSIONS

A. Experiments in Function-Level Fault Localization

Existing studies suggest that fault localization at the statement level might not be an optimal technique [71], particularly for programs in the Defects4J repository. Due to the small number of test cases and sparse coverage paths in the Defects4J repository, previous investigations have usually used function-level fault localization [72], [73]. As a result, we replicate our experiments at the function level for the Defects4J repository.

Tables X and XI summarize the results of Jaccard's (i.e., best formula in our experiment) fault localization at the function level. Since a single faulty function may include several faulty statements, the number of multiple fault programs at the function level is reduced somewhat in comparison to the statement level. After filtering out single-fault versions, the total number of faulty versions with multiple functions is 93, which contains 323 faulty functions.

The experimental results in this section are given in Table X in terms of the metric EXAM, which is consistent with our conclusion in RQ. Specifically, when checking a few functions toward the top of the suspiciousness ranking list, the number of localized faulty functions increases when all specific CC test cases are removed, but decreases when all irrelevant CC test cases are removed. Besides, as compared to the original situation, the isolation-based multiple fault localization approach can improve the accuracy of fault localization while checking a small number of functions, and the CC test case cleaning strategy on this basis can further enhance this advantage.

Following this, as given in Table XI, the result in terms of the metric TOP-N is also consistent with the preceding statement-level experiments in the majority of cases. That is, removing the specific and unspecific CC test cases can significantly improve the results of high-accuracy fault localization compared to the original situation. The isolation-based multiple fault localization approach can also effectively improve fault localization accuracy. In addition, in most cases, removing relevant CC test cases on the basis of the isolation-based multiple fault localization approach improves fault localization accuracy even further.

However, the results in Table XI are inconsistent with those in Tables VII or IX. For instance, removing the irrelevant CC test case can lead to an increase in terms of TOP-3, or a decrease in terms of TOP-3 and TOP-5 after the application of the isolation-based fault localization approach. Based on our analysis, the problem arises because test case coverage information is insufficient, and the coverage paths of the failed test cases are frequently identical at the function level, which makes it difficult to distinguish the correct functions from the faulty functions, eventually causing more correct functions to have the same suspiciousness value as the faulty function. In other words, this does not mean that the faulty function's suspiciousness value lowers, but that the suspiciousness value of more correct functions elevates to the same level as the faulty function, which makes the faulty function more difficult to detect.

TABLE X
COMPARISON OF THE FAULT LOCALIZATION ACCURACY OF THE CLEAN STRATEGY IN TERMS OF THE METRIC EXAM IN DEFECTS4J REPOSITORY

$EXAM$ (%)	Percentage of Faulty Statements (%)					
	Origin	Clean			Isolation	Isolation + Clean
		Specific	Unspecific	Irrelevant		
1	1.9	**2.8**	1.9	*0.9*	**2.8**	**3.4**
5	17.3	**19.8**	19.2	17.6	16.1	**19.5**
10	28.5	**30.3**	30.7	29.1	26.3	**31.0**
15	35.6	**36.5**	37.8	35.6	32.2	**36.2**
20	41.5	**41.8**	43.0	41.5	37.2	39.0
30	48.0	47.7	48.0	48.0	41.5	42.1
40	53.3	52.9	54.2	52.6	46.1	47.1
50	56.3	56.3	57.6	56.3	49.2	50.5
60	77.1	77.4	78.0	76.8	77.1	77.1
70	93.2	93.2	93.5	92.9	92.6	92.9
80	95.0	95.0	95.0	95.7	94.7	94.7
90	98.5	98.5	98.5	98.8	98.5	98.5
100	100	100	100	100	100	100

The bold font refers to positive results and italic font refers to negative results.

TABLE XI
COMPARISON OF THE FAULT LOCALIZATION ACCURACY OF THE CLEAN STRATEGY IN TERMS OF THE METRIC TOP-N FOR DEFECTS4J REPOSITORY

Formula	TOP-N	Origin	Clean			Isolation	Isolation + Clean
			Specific	Unspecific	Irrelevant		
$Ochiai$	TOP-1	21	**24**	**27**	*11*	**26**	**38**
$Dstar$		20	**24**	**26**	*11*	**28**	**39**
$Jaccard$		22	**25**	**28**	*11*	**26**	**39**
$Ochiai$	TOP-3	83	**90**	**91**	**92**	83	**97**
$Dstar$		85	**91**	**91**	**86**	85	**97**
$Jaccard$		85	**91**	**94**	**91**	*82*	**95**
$Ochiai$	TOP-5	124	**125**	**131**	*116*	*115*	**126**
$Dstar$		119	**122**	**125**	*110*	*114*	**126**
$Jaccard$		127	**128**	**132**	*115*	*114*	*125*

The bold font refers to positive results and italic font refers to negative results.

However, we recommend that researchers can follow the guidelines provided in Section V-E. After all, there is still a potential risk that removing irrelevant CC test cases may result in a decrease in fault localization accuracy, which is not observed when removing specified and unspecific CC test cases. Moreover, adopting an isolation-based approach to fault localization and removing CC test cases can achieve positive results for high-accuracy fault localization (i.e., in terms of TOP-1).

B. Experiments in the Relabel Strategy

According to existing research, another popular strategy for dealing with CC test cases is the Relabel strategy [21]. As mentioned in Section IV-C, the Relabel strategy changes the result of detected CC test cases from pass to fail after execution. In recent studies regarding CC test cases, several approaches deal with the CC test cases that involve the use of relabeling [24], [56]. Therefore, we employ the Relabel strategy once more to verify the validity of our empirical observations.

Table XII summarizes experimental results in terms of the metric TOP-N, where N is set to 5, which is consistent with the setting in Section IV-D2. As given in Table XII, the experimental results by using the Relabel strategy are consistent with the experimental results by using the Clean strategy. Specifically, removing the specific and unspecific CC test cases increases the accuracy of fault localization in comparison to the original case, whereas removing the irrelevant CC test cases decreases the accuracy of fault localization. The last column of Table XII demonstrates that employing the Relabel strategy for CC test cases on the basis of the isolation-based multiple fault localization approach can further increase fault localization accuracy.

Therefore, the conclusions and practical guidelines proposed in our study still hold when the CC test case processing strategy is changed from the Clean strategy to the Relabel strategy.

VII. THREATS TO VALIDITY

In this section, we discuss the potential threats to our study.

A. Internal Validity

The main internal threat in our study is the derivation part of our theoretical research via the Jaccard formula. To alleviate this threat, we also used the Dstar formula and the Ochiai formula for

TABLE XII
FAULT LOCALIZATION ACCURACY COMPARISON OF THE RELABEL STRATEGY IN TERMS OF THE METRIC TOP-N

Subject	Formula	TOP-5					
		Origin	Relabel			Isolation	Isolation + Relabel
			Specific	Unspecific	Irrelevant		
SIR	$Ochiai$	2328	**2423**	**2638**	*1724*	**2837**	**3208**
	$Dstar$	2180	**2309**	**2490**	*1664*	**2714**	**3068**
	$Jaccard$	2347	**2440**	**2659**	*1729*	**2857**	**3250**
Defects4J	$Ochiai$	157	**172**	**195**	*140*	**165**	**211**
	$Dstar$	156	**172**	**195**	*136*	**166**	**209**
	$Jaccard$	156	**170**	**198**	*139*	**166**	**213**
ALL	$Ochiai$	2485	**2595**	**2833**	*1864*	**3002**	**3419**
	$Dstar$	2336	**2481**	**2685**	*1800*	**2880**	**3277**
	$Jaccard$	2503	**2610**	**2857**	*1868*	**3023**	**3463**

The bold font refers to positive results and italic font refers to negative results.

theoretical derivation to verify its effectiveness. In the current field of fault localization, Jaccard, Dstar, and Ochiai are among the most commonly used formulas [35]–[37]. In the future, we propose to use other fault localization formulas, such as Op2 and Tarantula, [25], [41] to further verify the effectiveness of our theoretical research.

B. External Validity

The external threat is related to the scale of our experiment and the representativeness of our experimental subjects. To alleviate this threat, we used 7374 faulty versions, 22 975 test cases, and 43 992 faults in multiple faults from the SIR; 181 faulty versions, 17 983 test cases, and 971 faults in multiple faults from the Defects4J repository. It is worth mentioning that the SIR and the Defects4J repository have been widely used in previous fault localization studies [4], [6], [16], [49], [56]. Therefore, the quality of the programs from these repositories can be guaranteed.

C. Construct Validity

Threats to construct validity include how well we measure our experimental results. To alleviate this threat, we used the metrics EXAM and TOP-N to evaluate the performance of our approach. These metrics have been widely used in evaluating the performance of multiple fault localization and CC test cases study [6], [16], [22], [25], [68]. In future work, we intend to use additional metrics to analyze the experimental results.

VIII. CONCLUSION

In this article, we conducted a theoretical analysis on the impact of CC test cases in multiple fault localization, and performed an empirical study to verify the correctness of the theoretical analysis. In particular, after theoretical analysis, we found that for each faulty statement in a multiple fault program, removing specific CC test cases helps to improve the fault localization accuracy, while removing irrelevant CC test cases has a negative effect. To verify the theoretical analysis, we conducted an empirical study on two large-scale open-source corpora SIR and Defects4J. Final experimental results verified the validity of our theoretical analysis.

In our empirical study, we found that more than 85% of multiple fault programs contained CC test cases during the execution of their associated test suite, and the percentage of CC test cases in the test suites was often between 0% and 20%. Moreover, our experimental results demonstrated that the traditional CC handling strategy proposed for single-fault programs is not applicable to multiple fault programs, since it results in lower accuracy for multiple fault localization. Furthermore, we proposed CC test cases identification solution based on the isolation-based multiple fault localization approach, which was inspired by existing multiple fault localization approaches. Specifically, using the isolation-based multiple fault localization approach in combination with the Clean strategy to remove all CC test cases that executed the faulty statement associated with the cluster can improve fault localization accuracy.

In the future, we first plan to conduct more formula derivations to verify the correctness of our theoretical analysis. Then, we aim to larger scale programs to verify the generalization of our empirical results. Finally, based on our findings, we also aim to design more practical CC test case identification approaches for multiple fault programs to increase the accuracy of multiple fault localization.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions, which can substantially improve the quality of this article.

REFERENCES

[1] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, Aug. 2006.
[2] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proc. 20th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2005, pp. 273–282.
[3] G. K. Baah, A. Podgurski, and M. J. Harrold, "Causal inference for statistical fault localization," in *Proc. 19th Int. Symp. Softw. Testing Anal.*, 2010, pp. 73–84.
[4] Y. Liu, Z. Li, R. Zhao, and P. Gong, "An optimal mutation execution strategy for cost reduction of mutation-based fault localization," *Inf. Sci.*, vol. 422, pp. 572–596, 2018.
[5] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. van Gemund, "A practical evaluation of spectrum-based fault localization," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1780–1792, 2009.

[6] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in parallel," in *Proc. Int. Symp. Softw. Testing Anal.*, 2007, pp. 16–26.

[7] A. Bandyopadhyay, "Mitigating the effect of coincidental correctness in spectrum based fault localization," in *Proc. IEEE 5th Int. Conf. Softw. Testing, Verification Validation*, 2012, pp. 479–482.

[8] K. Lin, L. Xu, and J. Wu, "A fast fuzzy C-means clustering for color image segmentation," *J. Image Graph.*, vol. 9, no. 2, pp. 159–163, 2004.

[9] R. Abreu, P. Zoeteweij, and A. J. van Gemund, "Spectrum-based multiple fault localization," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2009, pp. 88–99.

[10] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.

[11] W. E. Wong and A. P. Mathur, "Reducing the cost of mutation testing: An empirical study," *J. Syst. Softw.*, vol. 31, no. 3, pp. 185–196, 1995.

[12] W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *J. Syst. Softw.*, vol. 83, no. 2, pp. 188–208, 2010.

[13] Y. Zheng, Z. Wang, X. Fan, X. Chen, and Z. Yang, "Localizing multiple software faults based on evolution algorithm," *J. Syst. Softw.*, vol. 139, pp. 107–123, 2018.

[14] W. Masri and R. A. Assi, "Prevalence of coincidental correctness and mitigation of its impact on fault localization," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, pp. 1–28, 2014.

[15] M. Weiser, "Programmers use slices when debugging," *Commun. ACM*, vol. 25, no. 7, pp. 446–452, 1982.

[16] R. Gao and W. E. Wong, "MSeer– An advanced technique for locating multiple bugs in parallel," *IEEE Trans. Softw. Eng.*, vol. 45, no. 3, pp. 301–318, Mar. 2019.

[17] A. Zakari, S. P. Lee, R. Abreu, B. H. Ahmed, and R. A. Rasheed, "Multiple fault localization of software programs: A systematic literature review," *Inf. Softw. Technol.*, vol. 124, 2020, Art. no. 106312.

[18] Y. Li and C. Liu, "Using cluster analysis to identify coincidental correctness in fault localization," in *Proc. 4th Int. Conf. Comput. Inf. Sci.*, 2012, pp. 357–360.

[19] L. Weishi and X. Mao, "Alleviating the impact of coincidental correctness on the effectiveness of SFL by clustering test cases," in *Proc. Theor. Aspects Softw. Eng. Conf.*, 2014, pp. 66–69.

[20] X. Xue, Y. Pang, and A. S. Namin, "Trimming test suites with coincidentally correct test cases for enhancing fault localizations," in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, 2014, pp. 239–244.

[21] Y. Miao, Z. Chen, S. Li, Z. Zhao, and Y. Zhou, "Identifying coincidental correctness for fault localization by clustering test cases," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, 2012, pp. 267–272.

[22] T. Ball, M. Naik, and S. K. Rajamani, "From symptom to cause: Localizing errors in counterexample traces," in *Proc. 30th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang.*, 2003, pp. 97–105.

[23] W. Masri and R. A. Assi, "Cleansing test suites from coincidental correctness to enhance fault-localization," in *Proc. 3rd Int. Conf. Softw. Testing, Verification Validation*, 2010, pp. 165–174.

[24] A. Bandyopadhyay and S. Ghosh, "Proximity based weighting of test cases to improve spectrum based fault localization," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2011, pp. 420–423.

[25] X. Wang, S. C. Cheung, W. K. Chan, and Z. Zhang, "Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, 2009, pp. 45–55.

[26] W. Masri, R. Abou-Assi, M. El-Ghali, and N. Al-Fatairi, "An empirical study of the factors that reduce the effectiveness of coverage-based fault localization," in *Proc. 2nd Int. Workshop Defects Large Softw. Syst., ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2009, pp. 1–5.

[27] R. Gopinath, C. Jensen, and A. Groce, "The theory of composite faults," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation*, 2017, pp. 47–57.

[28] D. O. Hyunsook, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng.*, vol. 10, no. 4, pp. 405–435, 2005.

[29] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, "Automatic repair of real bugs in Java: A large-scale experiment on the Defects4j dataset," *Empirical Softw. Eng.*, vol. 22, no. 4, pp. 1936–1964, 2017.

[30] Z. Li, H. Wang, and Y. Liu, "HMER: A hybrid mutation execution reduction approach for mutation-based fault localization," *J. Syst. Softw.*, vol. 168, 2020, Art. no. 110661.

[31] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *Proc. 34th Int. Conf. Softw. Eng.*, 2012, pp. 14–24.

[32] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2013, pp. 345–355.

[33] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of IR-based fault localization techniques," in *Proc. Int. Symp. Softw. Testing Anal.*, 2015, pp. 1–11.

[34] A. Zakari *et al.*, "Spectrum-based fault localization techniques application on multiple-fault programs: A review," *Glob. J. Comput. Sci. Technol.*, vol. 20, pp. 41–48, 2020.

[35] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull. Soc. Vaudoise Sci. Nat.*, vol. 37, pp. 547–579, 1901.

[36] R. Abreu, P. Zoeteweij, and A. J. van Gemund, "An evaluation of similarity coefficients for software fault localization," in *Proc. 12th Pacific Rim Int. Symp. Dependable Comput.*, 2006, pp. 39–46.

[37] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Trans. Rel.*, vol. 63, no. 1, pp. 290–308, Mar. 2014.

[38] F. Wotawa and M. Nica, "Program debugging using constraints-is it feasible?," in *Proc. 11th Int. Conf. Qual. Softw.*, 2011, pp. 236–243.

[39] W. E. Wong, Y. Shi, Y. Qi, and R. Golden, "Using an RBF neural network to locate program bugs," in *Proc. 19th Int. Symp. Softw. Rel. Eng.*, 2008, pp. 27–36.

[40] N. DiGiuseppe and J. A. Jones, "On the influence of multiple faults on coverage-based fault localization," in *Proc. Proc. Int. Symp. Softw. Testing Anal.*, 2011, pp. 210–220.

[41] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–32, 2011.

[42] Z. Li, Y. Wu, H. Wang, X. Chen, and Y. Liu, "Review of software multiple fault localization approaches," *Chin. J. Comput.*, pp. 1–33, 2021. [Online]. Available: http://cjc.ict.ac.cn/online/bfpub/lz-2021525102358.pdf

[43] C. Artho, "Iterative delta debugging," *Int. J. Softw. Tools Technol. Transfer*, vol. 13, no. 3, pp. 223–246, 2011.

[44] H.-L. Cao and S.-J. Jiang, "Multiple-fault localization based on chameleon clustering," *Tien Tzu Hsueh Pao/Acta Electronica Sinica*, vol. 45, no. 2, pp. 394–400, 2017.

[45] S.-M. Lamraoui and S. Nakajima, "A formula-based approach for automatic fault localization of multi-fault programs," *J. Inf. Process.*, vol. 24, no. 1, pp. 88–98, 2016.

[46] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proc. 24th Int. Conf. Softw. Eng.*, 2002, pp. 467–477.

[47] Y. Wu, Z. Li, Y. Liu, and X. Chen, "FATOC: Bug isolation based multi-fault localization by using optics clustering," *J. Comput. Sci. Technol.*, vol. 35, no. 5, pp. 979–998, 2020.

[48] B. Liu, Lucia, S. Nejati, L. Briand, and T. Bruckmann, "Localizing multiple faults in simulink models," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reengineering*, 2016, pp. 146–156.

[49] Y. Huang, J. Wu, Y. Feng, Z. Chen, and Z. Zhao, "An empirical study on clustering for isolating bugs in fault localization," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2013, pp. 138–143.

[50] Z. Wei and B. Han, "Multiple-bug oriented fault localization: A parameter-based combination approach," in *Proc. IEEE 7th Int. Conf. Softw. Secur. Rel. Companion*, 2013, pp. 125–130.

[51] T. A. Budd and D. Angluin, "Two notions of correctness and their relation to testing," *Acta Informatica*, vol. 18, no. 1, pp. 31–45, 1982.

[52] W. Masri and A. Podgurski, "An empirical study of the strength of information flows in programs," in *Proc. Int. Workshop Dyn. Syst. Anal.*, 2006, pp. 73–80.

[53] J. Kim, J. Kim, and E. Lee, "VFL: Variable-based fault localization," *Inf. Softw. Technol.*, vol. 107, pp. 179–191, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584918302453

[54] W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, "Effective software fault localization using an RBF neural network," *IEEE Trans. Rel.*, vol. 61, no. 1, pp. 149–169, Mar. 2012.

[55] B. Hofer, "Removing coincidental correctness in spectrum-based fault localization for circuit and spreadsheet debugging," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2017, pp. 199–206.

[56] Y. Liu, M. Li, Y. Wu, and Z. Li, "A weighted fuzzy classification approach to identify and manipulate coincidental correct test cases for fault localization," *J. Syst. Softw.*, vol. 151, pp. 20–37, 2019.

[57] R. A. Assi, W. Masri, and C. Trad, "Substate profiling for enhanced fault detection and localization: An empirical study," in *Proc. IEEE 13th Int. Conf. Softw. Testing, Validation Verification*, 2020, pp. 16–27.
[58] A. Sabbaghi, M. R. Keyvanpour, and S. Parsa, "FCCI: A fuzzy expert system for identifying coincidental correct test cases," *J. Syst. Softw.*, vol. 168, 2020, Art. no. 110635. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220301102
[59] R. A. Assi, W. Masri, and C. Trad, "How detrimental is coincidental correctness to coverage-based fault detection and localization? An empirical study," *Softw. Testing, Verification Rel.*, vol. 31, no. 5, 2021, Art. no. e1762.
[60] V. Debroy and W. E. Wong, "Insights on fault interference for programs with multiple bugs," in *Proc. 20th Int. Symp. Softw. Rel. Eng.*, 2009, pp. 165–174.
[61] N. DiGiuseppe and J. A. Jones, "Fault interaction and its repercussions," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance*, 2011, pp. 3–12.
[62] J. Li, X. Yan, B. Liu, and S. Wang, "An insight of double-faults interactions in program: An empirical study," in *Proc. 2nd Int. Conf. Rel. Syst. Eng.*, 2017, pp. 1–6.
[63] N. DiGiuseppe and J. A. Jones, "Software behavior and failure clustering: An empirical study of fault causality," in *Proc. IEEE 5th Int. Conf. Softw. Testing, Verification Validation*, 2012, pp. 191–200.
[64] X. Xue and A. S. Namin, "How significant is the effect of fault interactions on coverage-based fault localizations?," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2013, pp. 113–122.
[65] L. Zhang, L. Yan, Z. Zhang, J. Zhang, W. Chan, and Z. Zheng, "A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization," *J. Syst. Softw.*, vol. 129, pp. 35–57, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121217300808
[66] K. H. T. Wah, "A theoretical study of fault coupling," *Softw. Testing, Verification Rel.*, vol. 10, no. 1, pp. 3–45, 2000.
[67] Z. Yu, C. Bai, and K.-Y. Cai, "Does the failing test execute a single or multiple faults? An approach to classifying failing tests," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, 2015, pp. 924–935.
[68] C. Liu, X. Zhang, and J. Han, "A systematic study of failure proximity," *IEEE Trans. Softw. Eng.*, vol. 34, no. 6, pp. 826–843, Nov./Dec. 2008.
[69] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 165–176.
[70] Z. Li, Y. Wu, and Y. Liu, "An empirical study of bug isolation on the effectiveness of multiple fault localization," in *Proc. IEEE 19th Int. Conf. Softw. Qual., Rel. Secur.*, 2019, pp. 18–25.
[71] Y. Kim, S. Mun, S. Yoo, and M. Kim, "Precise learn-to-rank fault localization using dynamic and static features of target programs," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, pp. 1–34, 2019.
[72] M. Zhang, X. Li, L. Zhang, and S. Khurshid, "Boosting spectrum-based fault localization using pagerank," in *Proc. 26th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2017, pp. 261–272.
[73] X. Li, W. Li, Y. Zhang, and L. Zhang, "DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2019, pp. 169–180.

Yonghao Wu received the B.S. degree in computer science and technology from Nanchang Hangkong University, Nanchang, China, in 2017, and the M.S. degree in the computer science and technology in 2020 from the Beijing University of Chemical Technology, Beijing, China, where he is currently working toward the Ph.D. degree in control science and engineering.

His research interests include fault localization and software testing.

Yong Liu (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science and technology and the Ph.D. degree in control science and engineering from the Beijing University of Chemical Technology, Beijing, China, in 2008, 2011, and 2018, respectively.

He is currently an Assistant Professor with the College of Information Science and Technology, Beijing University of Chemical Technology. He has authored or coauthored more than ten papers in referred journals or conferences, such as the *Journal of Systems and Software*, the *Information Sciences*, the IEEE INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY, RELIABILITY AND SECURITY, the International Conference on Software Analysis, Testing and Evolution (SATE), and the International Computer Software and Applications Conference. His research focuses on software engineering, including software debugging and software testing, such as source code analysis, mutation testing, and fault localization.

Dr. Liu is a Member of the China Computer Federation and the Association for Computing Machinery.

Weibo Wang received the B.S. degree in computer science and technology from Tiangong University, Tianjin, China, in 2019. He is currently working toward the master's degree in software engineering with the Beijing University of Chemical Technology, Beijing, China.

His research interests include fault localization and software testing.

Zheng Li received the B.Sc. degree in the computer science and technology from the Beijing University of Chemical Technology, Beijing, China, in 1996, and the Ph.D. degree in computer science from CREST Centre, King's College London, London, U.K., in 2009.

He is currently a Professor with the College of Information Science and Technology, Beijing University of Chemical Technology. He was a Research Associate with King's College London and University College London, London. He has authored or coauthored more than 60 papers in referred journals or conferences, such as the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, the International Conference on Software Engineering, the *Journal of Software: Evolution and Process*, the *Information and Software Technology*, the *Journal of Systems and Software*, the *International Conference on Software Maintenance*, the IEEE International Conference on Software Maintenance and Evolution, the International Computer Software and Applications Conference, the IEEE International Working Conference on Source Code Analysis and Manipulation, and the IEEE International Conference on Software Quality, Reliability and Security. His research focuses on software engineering, including program testing, source code analysis, and manipulation.

Xiang Chen (Member, IEEE) received the B.Sc. degree in information management and system from the School of Management, Xi'an Jiaotong University, Xi'an, China, in 2002, and the M.Sc. and Ph.D. degrees in computer software and theory from Nanjing University, Nanjing, China, in 2008 and 2011, respectively.

He is currently an Associate Professor with the Department of Information Science and Technology, Nantong University, Nantong, China. He has authored or coauthored more than 60 papers in referred journals or conferences, such as the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, the *Information and Software Technology*, the *Journal of Systems and Software*, the IEEE TRANSACTIONS ON RELIABILITY, the *Journal of Software: Evolution and Process*, the *Software Quality Journal*, the *Journal of Computer Science and Technology*, the International Conference on Software Engineering, the IEEE/ACM International Conference Automated Software Engineering, the IEEE International Conference on Software Maintenance and Evolution, the IEEE International Conference on Software Analysis, Evolution and Reengineering, and the International Computer Software and Applications Conference. His research focuses on software engineering, including software maintenance and software testing, such as software defect prediction, combinatorial testing, regression testing, and fault localization.

Dr. Chen is a Senior Member of the China Computer Federation and a Member of the Association for Computing Machinery.

Paul Doyle received the Ph.D. degree in astronomical distributed data processing from the Dublin Institute of Technology, Dublin, Ireland, in 2015.

He is currently the Head of the School of Computer Science, Technological University Dublin, Dublin, Ireland. For more than 20 years, he was with industry in Silicon Valley, CA, USA, and in Dublin. He was a Product and Quality Director of CR2, a banking software company, Dublin. He was a Senior Manager with Sun Microsystems, Menlo Park, CA. He was a Senior Developer with BlueStar Financial Investment. His research interests include Big Data processing of astronomical images, distributed systems, systems infrastructure, and educational pedagogy.