Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 8 (4): 328-344, 2014
ISSN 1819-4311 / DOI: 10.3923/jse.2014.328.344

Identifying Coincidental Correctness in Fault Localization via Cluster Analysis

Yihan Li and Chao Liu
School of Computer Science and Engineering, Beihang University, Beijing, 100191, China

Corresponding Author: Yihan Li, School of Computer Science and Engineering, Beihang University, Beijing, 100191, China

ABSTRACT

Coverage-based fault localization is a statistical technique that assists developers in finding faulty entities efficiently by contrasting program traces. Although coverage-based fault localization has been shown to be promising, its effectiveness still suffers from occurrences of coincidental correctness which means test cases exercise faulty statements but do not result in failure information. Recent researches indicate that coincidental correctness is a common problem in software testing and harmful for fault localization. To enhance effectiveness of fault localization, in this study, we present a clustering approach to identify coincidental correctness in test suites for fault localization. An effective clustering technique is used to group test cases. Then we present an adaptive sampling strategy to identify and choose potential coincidentally correct tests from clusters such that the number of the identified coincidentally correct tests is guaranteed to be no more than the actual number of coincidentally correct tests in the test suite. Three representative fault localization techniques are evaluated to see whether they can benefit from identified coincidentally correct tests. The experimental results show that our approach can alleviate the coincidental correctness problem and improve the effectiveness of fault localization.

Key words: Fault localization, debugging, coincidental correctness, cluster analysis, K-means++

INTRODUCTION

Computer software may behave unexpectedly due to faulty codes that survive the development process. When failures are revealed during testing process, software developers often take great efforts to find faulty program entities with the help of execution information of both passed and failed tests and then try to fix faults. Due to resource constraint, debugging is always prohibitively expensive and time-consuming. According to a previous study, locating faults is one of the most expensive tasks in debugging (Vessey, 1985). Therefore, in recent years a variety of fault localization techniques (Abreu *et al*., 2006; Jones *et al*., 2002; Naish *et al*., 2011) have been proposed to narrow down the possible locations of software faults, with the aim to improve efficiency of debugging.

In particularly, Coverage-Based Fault Localization (CBFL) techniques which are based on the analysis of differences between passed tests and failed tests, are promising in identifying program entities that induce program failures. They are lightweight techniques and can be easily integrated into existing test plans. Specifically, these techniques aim at exploiting correlations between program entities and program failures via statistically analyzing coverage information and then

assign various suspiciousness scores to program entities according to statistical formulas. Finally program entities are ranked in descending order of their suspiciousness scores for developers' inspection. The intuition behind these techniques is that program entities that are primarily executed by failed tests are assigned higher suspiciousness scores than those that are primarily executed by passed tests, thus these entities are given higher priority for examination during debugging. For effectively locating faults, many kinds of statistical formulas have been proposed, such as Tarantula (Jones *et al*., 2002), Jaccard and Ochiai (Abreu *et al*., 2006).

Although CBFL techniques have shown encouraging effectiveness in previous studies (Abreu *et al*., 2006; Parnin and Orso, 2011; Qi *et al*., 2013), their accuracy to diagnose faulty statements that cause tests to fail still needs improving. A key insight of CBFL techniques is based on the assumption that the program will yield failure information when the faulty statements are exercised. However, this is not necessarily always the case due to coincidental correctness which occurs when "No failure is detected, even though a fault has been executed" (Richardson and Thompson, 1993). Generally, program faults do not always cause failure unless certain special case conditions are met. Voas (1992) presented a model called Propagation Infection Execution (PIE) to investigate the conditions for a failure to be observed which specifies the three conditions that must be satisfied: (1) Fault is executed, (2) Fault cause the program to yield erroneous state and (3) Erroneous state infects the output. A test is called to be coincidentally correct if the condition (1) Holds but the condition, (2) Does not hold, regardless the condition and (3) Holds or not. Previous studies have empirically found that coincidentally correct tests occur frequently in software programs and fault localization techniques are potentially susceptible to occurrence of coincidental correctness (Masri *et al*., 2009), since coincidentally correct tests potentially cause faulty statements to be ranked as less suspicious than when they are not present. Thus it is crucial to develop techniques to mitigate negative effects produced by coincidentally correct tests on fault localization.

In this study we propose a cluster-based approach, ICCCA, to identify test cases that are possible to be coincidentally correct. An effective clustering technique is used to group test cases. The technique clusters test cases based on execution profiles. Our intuition is that passing test cases that are grouped in the same cluster with some failed test cases may also execute faulty statements as failed ones do. Thus, these passing tests are likely to be coincidentally correct. Then, an adaptive sampling strategy is proposed to estimate the possible number of coincidentally correct tests in the test suite and choose some passing tests from clusters that contain failed test cases. After identification of potentially coincidentally correct tests from clusters, two strategies that deal with coincidental correctness are studies on three representative fault localization techniques. To demonstrate the feasibility of our proposed method for assisting in fault localization, we conduct experiments on the Siemens suite and Unix programs.

Jones *et al*. (2002) proposed a technique called Tarantula to identify buggy statements. The technique gathers execution information of passed tests and failed tests and then highlights the particular statements that are suspected to contain faults in different colors. The basic idea is that statements that are covered by more failed tests and fewer passed tests are given higher confidence to be faulty. Then, different colors are assigned to statements according to how likely each statement is to be faulty. A buggy statement is supposed to be colored as red for highlight so that developers can focus on it quickly. In the experiments conducted by Jones *et al*. (2002), they observed that a test case might not result in failure when executing faulty statements which reduce suspiciousness values of faulty statements.

To study the accuracy of diagnosis of fault localization, Abreu *et al.* (2006) investigated following several parameters: (1) Similarity coefficient which is a statistical formula used to rank potentially faulty statements, (2) Observation quality which involves classification of tests as passed or failed and (3) Observation quantity which is the size of tests available for debugging. Their experiments indicated that coverage-base fault localization suffers from inaccuracy of observation quality which means that errors may not propagate to failure and thus go undetected. Therefore, the set of test cases that produce failure information is only a subset of the set of test cases in which faulty statement is executed. In their experiments, they showed that on average about 20% of the program under debugging still need to be inspected by developers.

Masri *et al.* (2009) found that coincidental correctness is prevalent in programs and is one of factors that degrade effectiveness of existing coverage-base fault localization techniques. They demonstrated that coincidentally correct tests could cause the fault to be ranked less suspicious than when they are not present for Tarantula style coverage-base fault localization. They proposed variations of techniques to detect coincidental correctness in passed tests (Masri and Abou-Assi, 2010). Their experimental results on 18 versions of subject programs are promising on Tarantula. However, the application scope of their techniques is small and the rate of false positives averaged 41.3% which may make their techniques not applicable to some other fault localization techniques.

Miao *et al.* (2012) proposed a cluster-based technique to identify coincidental correctness in test suites. It groups similar runs based on statement profiles into the same clusters using simple K-means clustering method. Then, by checking whether each cluster contains failed tests or not, it selects a subset of test cases and regards all of them as coincidentally correct tests. However, since the actual number of coincidentally correct tests in the test suite is unknown, the number of identified coincidentally correct tests chosen by their approach may be greater than the actual number of coincidentally correct tests in the test suite and thus yield too many false positives, especially when a test suite contains no coincidentally correct tests. To deal with such problems, in this study a different clustering technique is used to effectively group similar test cases and a novel sampling strategy is presented to keep level of false positives.

AN ILLUSTRATIVE EXAMPLE

Figure 1 illustrates how coincidental correctness reduces suspiciousness rank of faulty statements in fault localization.

Program mid() that is adapted from the previous study, Yu *et al.* (2008) inputs three integers and outputs the median value of the three integers. There are 13 statements $s_1 \ldots s_{13}$ in the program in Fig. 1, where s_6 is faulty. Suppose the program has 8 test cases $t_1 \ldots t_8$. A dot for a statement under a test case means the corresponding statement is executed in the corresponding test case and pass/fail status is shown at the bottom. To the right of the test cases are suspiciousness scores for each statement and corresponding ranks. CBFL techniques make summary on the execution information and employ various statistical formula to calculate suspiciousness values for each statement and then rank them in descending order. Use Ochiai (Abreu *et al.*, 2006) as an example. The suspiciousness score of the statement s_7 is greatest among all executable statements and thus s_7 is ranked first in the suspiciousness list. Note that the faulty statement s_6 is ranked second in the list. Therefore, following the rank list, developers have to examine at least two statements to find faulty statements.

As shown in Fig. 1, there are three test cases that execute faulty statement, only one of which results in failure.

mid () {		Test cases								Techniques		
int x, y, z, m;	t1	t2	t3	t4	t5	t6	t7	t8				
	2, 3, 1	5, 3, 4	1, 2, 3	5, 5, 5	5, 5, 3	5, 3, 4	4, 2, 5	7, 5, 4	Ochiai	Rank	Ochiai-CC	Rank
1. read (x, y, z);	●	●	●	●	●	●	●	●	0.35	4	0.61	4
2. m = z;	●	●	●	●	●	●	●	●	0.35	4	0.61	4
3. if (y<z) {	●	●	●	●	●	●	●	●	0.35	4	0.61	4
4. if(x<y)		●	●			●	●		0.50	3	0.87	2
5. m = y;			●						0.00	5	0.00	5
6. else if(x<z-2)/*FAULT*/		●				●	●		0.58	2	1.00	1
7. m = x; }						●	●		0.71	1	0.82	3
8. else {	●			●	●			●	0.00	5	0.00	5
9. if (x>y)	●			●	●			●	0.00	5	0.00	5
10. m = y;								●	0.00	5	0.00	5
11. else if (x>z)	●			●	●				0.00	5	0.00	5
12. m = x; }	●				●				0.00	5	0.00	5
13. print (m); }	●	●	●	●	●	●	●	●	0.35	4	0.61	4
Pass/Fail	P	P	P	P	P	F	P	P				

Fig. 1: Example program, information about its test suite and its rank results

Thus, in accordance with definition of coincidental correctness, t_2 and t_7 are called coincidentally correct tests. How the suspiciousness scores change for each statement if these two coincidentally correct tests are recognized? The column Ochiai-CC shows the suspiciousness scores after these two tests are detected as coincidentally correct tests and both are regarded as failed tests. In such case the faulty statement s_6 is ranked first. Following the updated rank list, developers would be directed first to focus attention on the faulty statement which could save developers' efforts in debugging. Thus, it is necessary to identify coincidentally correct tests in test suites to enhance CBFL techniques.

OVERVIEW OF PROPOSED APPROACH

For the convenience of following discussions, let us suppose a faulty program P is tested against a test suite T which comprises of the set of passed test cases denoted as T_p and the set of failed test cases denoted as T_f. Also suppose that the set of coincidentally correct test cases in the test suite T is denoted as T_{cc} and the set of identified coincidentally correct tests is denoted as T_{icc}, each of which is a potential candidate of the members of T_{cc}.

Empirical observations have shown that test cases may behave similarly due to the same fault (Ammann and Knight, 1988). Since coincidentally correct tests must invoke execution of the faulty codes, such tests may share some similar paths in their executions with failed tests to reach faulty statements. Therefore, cluster analysis can be a potential method to identify coincidentally correct tests. In our cases to identify coincidental correctness based on this technique, our intuition is that if we measure test cases with respect to some proper execution profiles, passing test cases that are grouped in the same cluster with failed test cases may also execute faulty statements as failed ones do. Thus, these passing tests in the clusters with higher ratio of failed tests are more likely to be coincidentally correct. The process of our approach to CBFL using cluster analysis is illustrated in Fig. 2. Given the original source code and the test suite as inputs, source code is executed against test suite and execution profiles and execution results for each test are collected first. Then, execution profiles and test results are fed to a clustering technique. After grouping tests, a sampling approach is employed to choose tests from clusters. Finally, the chosen tests are regarded as identified coincidentally correct tests and two dealing strategies are used to evaluate impact on coverage-base fault localization techniques.

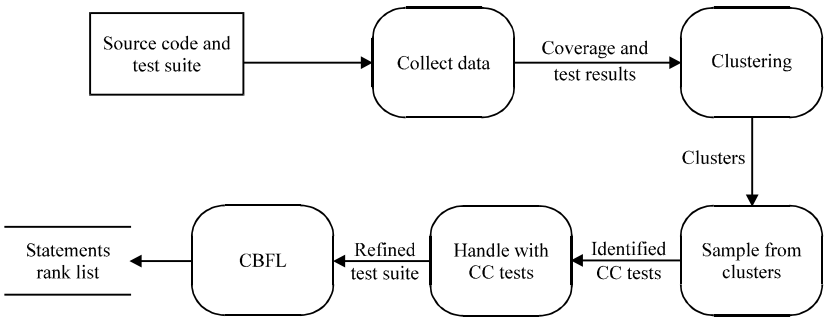

Fig. 2: Process of our approach to CBFL techniques using cluster analysis

Typically, a clustering procedure often involves the following four steps: (1) Particular clustering technique employed, (2) Feature used to cluster, (3) Number of clusters to be formed and (4) A strategy for sampling from clusters.

Clustering technique: Clustering is an effective technique in discovering similarity between objects. Of the many clustering methods, the K-means method is a widely used technique that seeks to group similar objects into the same cluster. Specifically, given an integer K and a set of N objects, the K-means aims to choose K centers so as to minimize $, the sum of the squared distances between each object and its closer center.

Since its simplicity and speed are very appealing in practice, the K-means has been widely used in software testing (Dickinson *et al*., 2001; Zhang *et al*., 2010). However, the K-means algorithm is generally susceptible to initialization of starting points. Initial starting objects are randomly selected as cluster centers in the K-means algorithm. When initial cluster centers are close to the desired centers, K-means has high possibility to generate better results. Otherwise, it will lead to local optima (Arthur and Vassilvitskii, 2007). Although the K-means can be improved by computing more than once, it increases computation expense and is still difficult to determine the computation limit.

In this study, we adopted a new clustering technique that is developed from K-means. In order to guarantee the accuracy of clustering, Arthur and Vassilvitskii (2007) proposed a way of initializing K-means by choosing random starting centers with very specific probabilities, leading to a combined algorithm they called K-means++. The algorithm is both fast and simple and can already achieve approximation guarantees that K-means cannot. Their experimental results showed that K-means++ substantially outperformed standard K-means in terms of both speed and accuracy.

Feature collection: Previous studies have shown that predicates in the program are strong indicator for faulty statements (Liblit *et al*., 2003; Liu *et al*., 2006). Furthermore, since coincidentally correct tests must invoke execution of the faulty codes, such tests may share some similar paths in their executions to reach faulty statements. To go through specific paths, evaluation of predicates in the paths may show similar patterns. Thus, in this study, predicates are used as features of test cases for clustering. Suppose a program P contains m predicates $<p_1, p_2,..., p_m>$. Predicates in the program are instrumented. Then the instrumented program is executed against the test suite T. When a test case t_i is executed on the program, for each predicate p_j, following data are collected: C_{ij}^{t} and C_{ij}^{f} which are execution counts of true evaluation and false

evaluation for p_j, respectively. If either evaluation of p_j is not exercised during execution of t_i, its corresponding value is set to 0. In this way, a value set containing counts for each evaluation of predicates can be obtained for t_i as features of that test case: $E(t_i)=<C_{i1}^t, C_{i1}^f, C_{i2}^t, C_{i2}^f,..., C_{im}^t, C_{im}^f>$. In this study profiles of all the test cases in T are used as inputs to the clustering technique. To measure the similarity between test cases, Euclidean distance is employed. Given two test profiles $E(t_i)$ and $E(t_j)$, the distance of these two tests is:

$$D(t_i, t_j) = \sqrt{\sum_{z=1}^{m}(C_{iz}^t - C_{jz}^t) + \sum_{z=1}^{m}(C_{iz}^f - C_{jz}^f)^2} \quad (1)$$

The smaller the distance is, the more similar two test cases are with respect to dynamic behaviors.

Number of clusters: The K-means++ takes the number of clusters (i.e., K) as a parameter. The clustering results depend on the number of clusters to be formed. If the number is set too small, dissimilar objects may be put together in one cluster. If the number is set too great, similar objects may be separated into different clusters. In this study, the number of clusters is set according to some percentage of the size of the test suite T, that is $K = L*|T|$, where $0<L<1$. In our experiments, we first set L = 6% to present more detailed results and evaluate whether our approach work well. This fixed percentage value is chosen according to previous studies (Dickinson *et al*., 2001; Yan *et al*., 2010; Miao *et al*., 2012). Different L will be discussed later.

Sampling strategy: After test cases are grouped together by the clustering algorithm, a subset of the population of test cases is chosen as identified coincidentally correct tests. In this section, we propose an adaptive sampling strategy to identify and choose coincidentally correct tests from clusters such that the number of the chosen test cases is guaranteed to be no more than the actual number of coincidentally correct tests in the test suite.

After clustering, passing tests grouped together with failed tests are likely to be coincidentally correct. Note that the actual number of coincidentally correct tests is unknown beforehand. To keep level of false positives, it is important to choose the appropriate number of passed test cases as results. Thus, we first estimate the possible number of coincidentally correct test in the test suite. The following procedure is conducted. First, the suspiciousness value of each statement *e* that contain fault is calculated using some statistical metric which is defined as S(e). In this section, Ochiai is employed since its performance on fault localization is effective and consistent (Abreu *et al*., 2006). Thus, S(e) is defined as follows:

$$S(e) = \frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf}) \times (a_{ef} + a_{ep})}} = \frac{a_{ef}}{\sqrt{|T_f| \times (a_{ef} + a_{ep})}} \quad (2)$$

The notation $<a_{np}, a_{nf}, a_{ep}, a_{ef}>$ denotes the number of test cases that satisfy certain conditions respectively where the first part of the subscript indicates whether the statement is executed (e) or not (n) and the second one indicates whether the test case is passed (p) or failed (f). In the context of disambiguation that these notations are provided for the specific statement e, we sometimes append "(e)" to each notation. For example, $a_{ep}(e)$ denotes the number of test cases each of which executes the statement e and is passed.

Suppose the faulty statement is e_f in the single-fault program. According to the definition of coincidental correctness, we can easily know that $a_{ep}(e_f)$ denotes the actual number of coincidentally correct tests in the test suite. Since the location of the fault is unknown, it is difficult to get the value. However, it is easy to get such a statement e' in the program P so that it satisfies the following condition:

$$\arg\max_{e' \in p} \{S(e') | a_{ef}(e') = |T_f|\} \tag{3}$$

Suppose the statement e* satisfies the Eq. 3. According to the Eq. 3, since $a_{ef}(e_f) = |T_f|$, it could be easily shown that $S(e^*) \geq S(e_f)$. Aslo the condition $a_{ef}(e^*) = a_{ef}(e_f)$ is satisfied. Consequently, we can infer that $a_{ep}(e^*) \leq a_{ep}(e_f)$ which reveals that $a_{ep}(e^*)$ is strictly less than or equal to the actual number of coincidentally correct tests. To verify:

$$S(e^*) \geq S(e_f)$$

$$\Rightarrow \frac{a_{ef}(e^*)}{\sqrt{|T_f| * (a_{ef}(e^*) + a_{ep}(e^*))}} \geq \frac{a_{ef}(e_f)}{\sqrt{|T_f| * (a_{ef}(e_f) + a_{ep}(e_f))}}$$

$$\Rightarrow \frac{a_{ef}(e^*)}{\sqrt{(a_{ef}(e^*) + a_{ep}(e^*))}} \geq \frac{a_{ef}(e_f)}{\sqrt{(a_{ef}(e_f) + a_{ep}(e_f))}}$$

$$\Rightarrow a_{ef}(e_f) + a_{ep}(e_f) \geq a_{ef}(e^*) + a_{ep}(e^*)$$

$$\Rightarrow a_{ep}(e_f) \geq a_{ep}(e^*)$$

Thus, we get lower bound of the actual number of coincidentally correct tests in the test suite and this value can be used as a threshold N to guide how many test cases should be chosen from clusters. In case of no coincidentally correct tests existing in the test suites, this strategy also suggests no tests should be chosen, thus does not produce any false positives in the results.

Figure 3 lists the sample algorithm. The algorithm takes in a set of K clusters generated by the clustering technique and the estimated possible number of coincidentally correct tests N and outputs a set of test cases that are identified as coincidentally correct. At lines 3-4 we first calculate

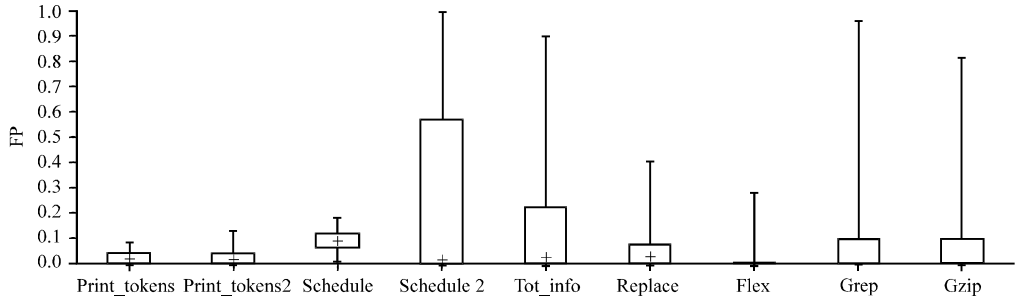

Fig. 3: False positives

the possibility of each cluster to contain coincidentally correct test and sort clusters in descending order according to its possibility. In the pseudocode, procedure R(c) calculate the ratio of failed tests in the cluster c which is defined as follows:

$$R(c) = \frac{\left|\{t \mid t \in c \wedge t \in T_f\}\right|}{|c|}$$

Passing tests in a cluster with higher ratio of failed tests are more likely to be coincidentally correct. At lines 5-11 we iteratively include the passing tests in clusters with higher ratio of failed tests in the returned test cases. The procedure P(c) returns the set of passing test cases in the cluster c. If the size of returned test cases plus the size of passing test cases in the current cluster c is greater than estimated possible coincidentally correct test (i.e., N), we skip this cluster so as to keep level of false positives (Table 1).

Fault localization: After N tests have been selected, these selected tests are all regarded as identified coincidentally correct tests. In this study, we present two different strategies that deal with identified coincidentally correct tests and investigate how these strategies impact various fault localization techniques.

- **Discarding strategy:** The identified coincidentally correct tests are discarded from the original test suite. Thus, the remaining test cases are used as inputs to fault localization techniques
- **Relabeling strategy:** The identified coincidentally correct tests are regarded as failing tests. This would increase failure information from tests which is desirable for fault localization techniques. However, false positives in the identified coincidentally correct tests may introduce risk

Table 1: Sample algorithm pseudocode

Algorithm sample algorithm	
Input:	
C:	K clusters of grouped test cases generated by clustering technique
N:	Estimated possible number of coincidentally correct tests in the test suite
Output: A set of test cases that are identified as coincidentally correct tests	
1:	Procedure sample (C, N)
2:	$T_{icc} \neg \{\}$
3:	$\forall c \in C$, calculate R(c) by Eq. 4
4:	Sort clusters in c in descending order according to R(c)
5:	For each c in C
6:	If c contains at least one failed test case
7:	If $\|T_{icc}\| + \|P(c)\| <= N$ then
8:	$T_{icc} \leftarrow T_{icc} \cup P(c)$
9:	End if
10:	End if
11:	End for
12:	Return T_{icc}
13:	End procedure

In this study, we study three representative fault localization techniques to investigate impact of coincidental correctness on various techniques. Tarantula is proposed by Jones *et al.* (2002). The rational for this technique is that the more failed tests and the less successful tests cover a statement, the more likely for the statement to be faulty. Different from Tarantula, Jaccard (Abreu *et al.*, 2006) also takes the absence in failed executions into account. Ochiai is first used as similarity coefficient in the previous studies (Abreu *et al.*, 2006). Compared To Jaccard, Ochiai amplifies the differences between executions and test results. Although these techniques share the same basic principle, the difference in metrics makes each technique perform differently in locating the faulty statements. Thus we plan to investigate whether these fault localizers can all benefit from identified coincidentally correct tests.

EXPERIMENTAL DESIGN

Subject programs: We used Siemens and Unix programs as our subject programs for the empirical studies. These programs have been extensively used to evaluate effectiveness of fault localization techniques in previous studies (Jones *et al.*, 2002; Abreu *et al.*, 2006; Naish *et al.*, 2011). For each program, there are a variety of test cases and faulty versions available. Table 2 presents the detailed information on the subject programs. The Versions column lists the number of faulty versions for each subject program. The column LOC shows the lines of code for each program. The column size of T represents the total number of available test cases in the test pool for each program. The last column lists range of the number of coincidentally correct tests for each faulty program. For better evaluation on effectiveness of the technique, each faulty version contains only one fault.

From the study of Do *et al.* (2006), we excluded those faulty versions whose faults cannot be detected by any test case in the test suites. Besides, we also removed the versions whose faults are introduced from modifications in the header files, mutants in variable declaration statements, or modifications in a macro statement started with "#define". In summary, we used all the remaining 143 faulty versions in our data analysis, of which 39 faulty versions contain no coincidentally correct tests.

Evaluation metrics: Following previous work (Masri and Abou-Assi, 2010), to empirically evaluate to what extent coincidentally correct tests are identified by our technique, we compute metrics to quantify the generated false negatives and false positives. Furthermore, expense reduction score is used to assess the impact of our approach on the effectiveness of fault localization techniques.

Table 2: Detailed information of the subject programs

Subject	Versions	LOC	Size of T	Range of CC
print_tokens	4	565	4130	357-1546
print_tokens2	10	510	4115	0-3815
replace	27	563	5542	0-4948
schedule	4	412	2650	1199-1481
schedule2	8	307	2650	1802-2636
tot_info	23	406	1054	0-1047
flex	27	5217	567	0-517
grep	23	12653	809	0-703
gzip	17	6573	213	0-47

- **Measure of generated false negatives:** This measure assesses whether or not we are successfully identifying all of the coincidentally correct tests in the test suite T. A lower measure value indicates that the technique can identify more coincidentally correct tests:

$$FN = \frac{|T_{cc} - T_{icc}|}{|T_{cc}|}$$

- **Measure of generated false positives:** This measure evaluates whether passing tests that are non-coincidentally correct are erroneously classified as coincidentally correct tests. Similarly, a lower measure value indicates that the technique makes fewer mistakes in recognizing coincidentally correct tests:

$$FP = \frac{|T_p - T_{cc} \cap T_{icc}|}{|T_p - T_{cc}|}$$

- **Measure of effectiveness improvement:** To measure the effectiveness of fault localization techniques, Yu *et al.* (2008) proposed Expense metric. The metric measures the percentage of the programs that must be examined to find the fault following rank list from top down. The lower the measure is, the better the effectiveness is. It is defined as follow:

$$Expense = \frac{|V_{examined}|}{|V|} \times 100\%$$

where, $|V|$ measures the size of executable statements in the program and $|V_{examined}|$ measures the number of statements that has to be inspected so as to find the fault. To study the effects of cluster analysis on fault localization, Expense reduction score $\Delta Expense = Expense - Expense'$ is used to measure relative effectiveness improvement, where, Expense and Expense' refers to Expense values before and after applying our approach to CBFL techniques respectively. A positive value of $\Delta Expense$ indicates that the effectiveness of fault localization is improved after applying our approach. The greater the measure value is, the more improvement the effectiveness of fault localization gains

RESULTS AND ANALYSIS

We reported the raw results of our experiments across the subject programs and then analyze them in detail. The first three experiments use L = 6% as a parameter.

Recognition accuracy: This subsection investigates how well our sampling strategy can identify coincidentally correct tests from clusters. Figure 3 and 4 list results of false positives and false negatives for each individual program separately. As shown in two figures, most versions of subject programs yield a relatively lower rate of false negatives but a higher rate of false negatives. The average rate of false positives over all the programs is 4.85%. Specifically, 92% versions generate a small number of false positives in the ranges (0, 10%). On the other hand, the average rate of

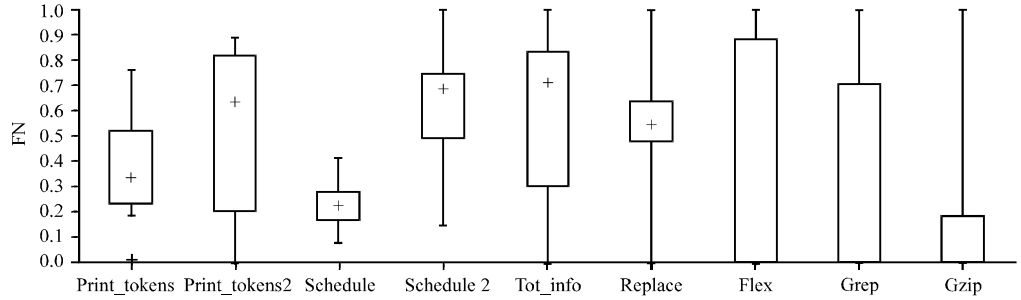

Fig. 4: False negatives

Table 3: Mean improvement for fault localization techniques

	Relabeling		Discarding	
Metric	+	-	+	-
Tarantula (%)	6.15	-1.8	5.32	-0.71
Jaccard (%)	5.11	-1.4	4.31	-0.24
Ochiai (%)	4.21	-1.5	3.62	-0.32

false negatives over all programs is 47.4%. More specifically, about 6% versions have detected all the coincidentally correct tests. We also observed that although some versions have no coincidentally correct tests, our sampling techniques can still successfully yield 0% false positive and 0% false negative, thus don't produce any misleading results. In summary, our sampling strategy is a relatively effective and safe method, 51% versions generate 0-50% false negatives and 0-10% false positives.

Fault localization improvement: Figure 5a-c list the results for the three techniques, respectively. For all these figures, due to space limitation, the first six names on the x-axis are short names for corresponding program names presented in Table 2. The y-axis denotes the number of versions that satisfy certain conditions. The "+", "=" and "-" symbols in the figures indicate that ΔExpense>0, = 0 and <0, respectively. Table 3 presents mean Expense reduction scores for three techniques.

From Fig. 5a-c and Table 3 we can observe that for most of the programs the three techniques can benefit from our approach. Take Tarantula for example. In total, about 84.6% versions of programs gain improvement or stay the same on fault localization using relabeling strategy with an average Expense reduction score of 6.15% and about 90.2% versions when using discarding strategy with an average Expense reduction score of 5.32%. The Expense reduction score for versions that are deteriorated is -1.8 and -0.71% on average for relabeling and discarding strategies respectively which is much lower than corresponding average improvements. The similar observations can be also made for Jaccard and Ochiai.

From Fig. 5a-c and Table 3, it can be seen that compared with Tarantual and Jaccard, Ochiai seems to get smaller improvement applying our approach. It is reasonable because Ochiai outperforms the other two techniques in locating faults. For some faulty versions, the faults are already ranked first by Ochiai. Thus, there is little space for further improvement.

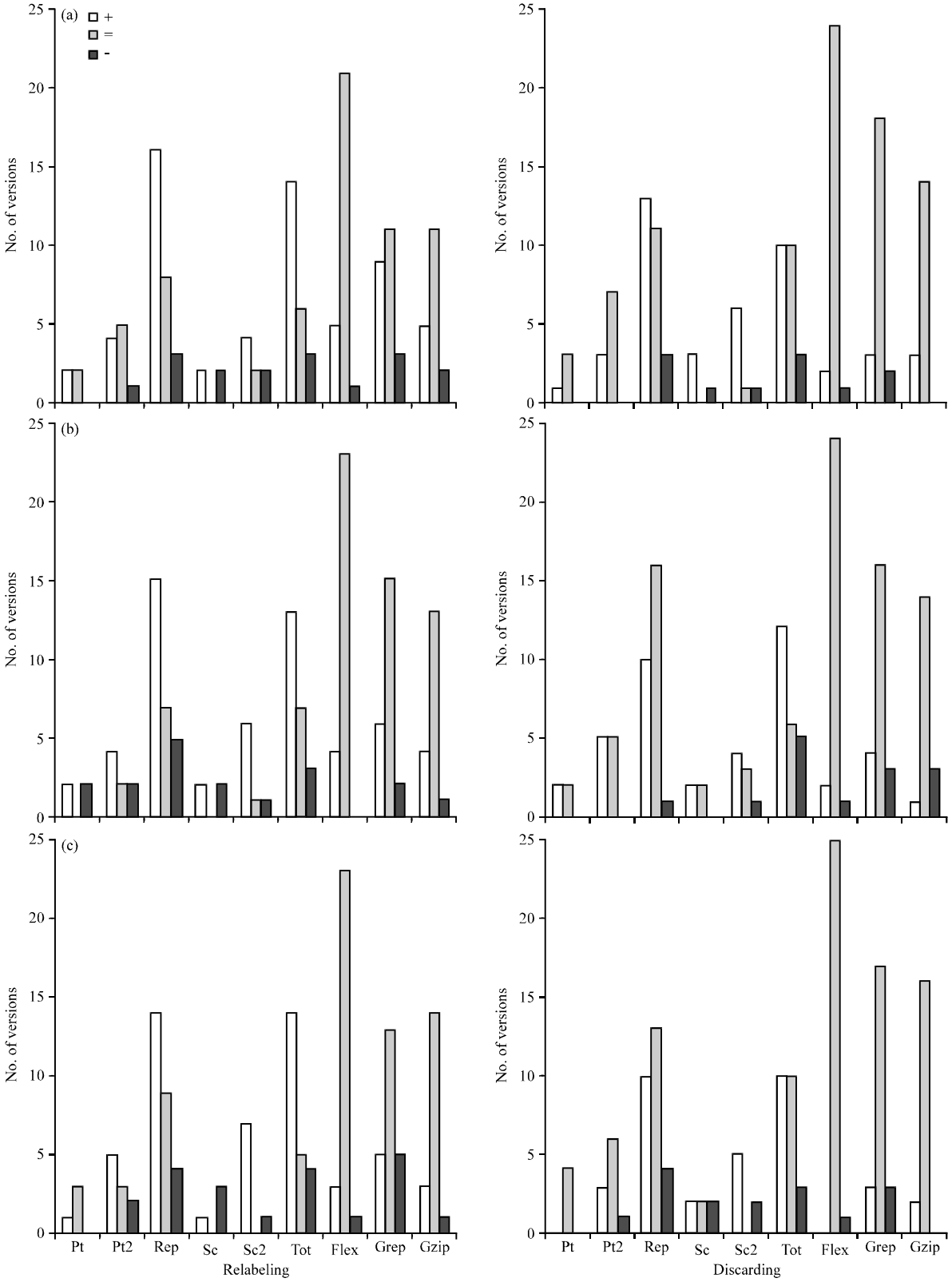

Fig. 5(a-c): Effectiveness of fault localization techniques after applying cluster analysis on (a) Taranula, (b) Jaccard and (c) Ochiai

Table 4: The p-values of U-tests for three techniques

Parameters	Tarantula	Jaccard	Ochiai
Relabeling	0.02	0.02	0.04
Discarding	0.01	0.02	0.03

Furthermore, we can find that discarding strategy seems to perform consistently more stable than relabeling strategy for all the three techniques. As shown in Table 3, when using discarding strategy, the mean ΔExpense for the versions that get deteriorated is smaller for all the three techniques compared with relabeling strategy. The reason why discarding strategy is more stable than relabeling strategy is that discarding a test case merely reduces the size of test cases provided for fault localization. However, relabeling an identified non-coincidentally correct test to a failing test case may make non-faulty statements more suspicious.

It can be seen that on average there are effective improvements after applying our approach to a base fault localization technique. Furthermore, we conduct two-tailed Mann-Whitney U-tests on the differences between the Expense scores before and after applying our approach on the three techniques, respectively. We make the following hypothesis: "H0: Does a technique enabled with our approach have no significant difference from the base technique?" Table 4 lists the p-values for hypothesis testing on the three techniques respectively. It can be observed that, for each technique, either by using discarding strategy and relabeling strategy, both of the p-values are less than 0.05. It indicates that the null hypothesis can be successfully rejected at the 0.05 significance level. Considering our previous observation that our approach, on average, improves its base version, we regard the improvements are significant at the 0.05 level for all the three techniques.

In summary, the results reveal that it is encouraging to improve effectiveness of fault localization by dealing with identified coincidental correctness with proper strategy. Specifically, discarding strategy is relatively safe method while relabeling strategy may introduce risk.

Comparative studies with peer works: Comparison of present study, technique with Miao *et al*. (2012) study is presented here, we compare our technique with the technique proposed by in their study (Miao *et al*., 2012). We denote this technique as "SEKE12" while our technique is denoted as "ICCCA". For comparison, these two techniques use the same K value. Due to space limitation, we will analyze the results of Ochiai in details. Figure 6 lists the experimental results on subject programs for two strategies that deal with coincidentally correct tests, respectively. In these figures, the x-axis represents the percentage of executable statements to be examined. The y-axis denotes the percentage of faulty versions whose faults have been located by examining no more than corresponding percentage of executable statements in x-axis.

From Fig. 6, we can observe that at most checkpoints, our technique can assist Ochiai in locating more faults than SEKE12 using either relabeling strategy or discarding strategy. When relabeling strategy is considered, by examining upto 5% of the codes in faulty versions, our technique can help Ochiai locate 26% more faults than SEKE12. And when discarding strategy is considered, this value is 29.6%. Our experimental results also show that for Tarantula and Jaccard techniques, our approach can be more helpful than SEKE12 in locating faults.

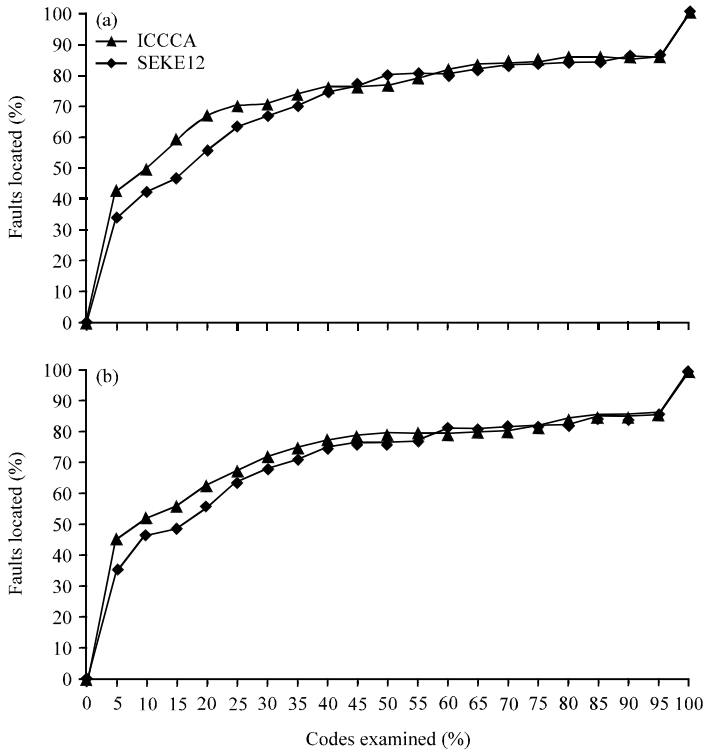

Fig. 6(a-b): Comparission of peer work and Ochiai technique, (a) Relabeling and (b) Discarding

DISCUSSION

The present experiments only used a fixed fraction of the number of test cases as the number of clusters to be form. In this subsection, we move on to further discussions on the impact of the number of clusters on performance of our approach.

The number of clusters to be formed affects the degree to which test cases are separated. As the number of clusters varies, the generated false positives and false negatives also vary, thus may introduce different impact on fault localization techniques. In this subsection, we set L = 1%, 2.5, 10 and 15%, respectively to further evaluate the performance of our approach on CBFL techniques.

Table 5 lists the averaged rates of false negatives and averaged rates of false positives over all the subject programs using different L. The results indicate that our approach always yields higher rate of false negatives but a lower rate of false positives using different values of L. Moreover, it can be seen that the false negative rate is negatively correlated to the false positive rate. As the number of clusters (i.e., L) increases, the ratio of the generated false negatives decreases while that of false positives increases. It is reasonable because our sampling strategy depends on the number of clusters. When the number of clusters increases, the size of each cluster decreases. Thus, some of passed tests which may be coincidentally correct or not, once put into a cluster with failed tests, may be spread to another clusters without failed tests. Therefore, these passed test cases will not be included in the results. Thus, L should not be set too large for our approach to be effective at grouping coincidentally correct tests.

Further, we perform two-tailed Mann-Whitney U-tests to compare the CBFL techniques enhanced by our approach with the corresponding base techniques on all the subject programs using different L. The p-values for hypothesis testing on the programs are listed in Table 6.

Table 5: Averaged FN and FP using different L percent

Parameters	Percentage				
L	1.00	2.50	6.00	10.00	15.00
FN	39.00	42.80	47.40	52.50	55.40
FP	6.11	5.65	4.85	4.10	3.45

Table 6: The p-values of U-tests for three techniques using different L

CBFL technique	L (%)	Relabeling	Discarding
Tarantula	1.0	0.03	0.02
	2.5	0.03	0.01
	10.0	0.01	0.02
	15.0	0.04	0.02
Jaccard	1.0	0.07	0.02
	2.5	0.03	0.01
	10.0	0.01	0.03
	15.0	0.03	0.01
Ochiai	1.0	0.11	0.03
	2.5	0.04	0.02
	10.0	0.03	0.02
	15.0	0.05	0.03

From the results, it can be seen that all but ones with bold font of the p-values are smaller than 0.05 which indicates that the null hypothesis can be successfully rejected at the 5% significance level. We can also observe that when discarding strategy is considered, the improvement is significant at the 5% level under all values of L. Thus, when our approach is applied on CBFL, we recommend that discarding strategy can be first considered to improve the effectiveness of fault localization.

CONCLUSION

In this study we proposed a clustering approach to identify coincidentally correct tests in the test suite. An effective clustering technique is employed to group similar test cases based on execution profiles. Then we presented an adaptive sampling strategy to identify and choose coincidentally correct tests from clusters such that the number of the chosen tests is guaranteed to be strictly on more than the actual number of coincidentally correct tests in the test suite. To mitigate the adverse effect of coincidental correctness on fault localization, two strategies are studied to deal with identified coincidentally correct tests on three representative fault localization techniques respectively. We conducted experiments to evaluate our techniques on Siemens Test Suites and Unix programs. The experimental results demonstrate that, our approach can always generate higher rate of false negatives but a lower rate of false positives. Furthermore, from the experiments, we can make the following conclusions:

- Our approach can effectively identify coincidentally correct tests in the test suite without introducing too many false positives
- Various fault localization techniques can benefit from identified coincidentally correct tests dealt with proper strategy. As a conclusion, the discarding strategy is relatively safe method while the relabeling strategy may introduce risk
- Compared with other peer works, our technique can assist fault localization techniques in locating more faults using either dealing strategy

For future work, we would like to develop more effective clustering techniques for grouping coincidentally correct tests. We also wish to consider further optimizations to the sampling strategy so as to reduce rate of false negatives and explore the benefits of applying the clustering techniques against programs with varying number of faults in them. In addition, more studies examining the impact of identified coincidentally correct tests on other fault localization techniques are to be conducted.

REFERENCES

Abreu, R., P. Zoeteweij and A.J.C. van Gemund, 2006. An evaluation of similarity coefficients for software fault localization. Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing, December 18-20, 2006, Riverside, CA., USA., pp: 39-46.

Ammann, P.E. and J.C. Knight, 1988. Data diversity: An approach to software fault tolerance. IEEE Trans. Comput., 37: 418-425.

Arthur, D. and S. Vassilvitskii, 2007. K-means++: The advantages of careful seeding. Proceedings of the 18th Annual ACM-SIAM Symposium of Discrete Analysis, January 7-9, 2007, New Orleans, LA., USA., pp: 1027-1035.

Dickinson, W., D. Leon and A. Fodgurski, 2001. Finding failures by cluster analysis of execution profiles. Proceedings of the 23rd International Conference on Software Engineering, May 12-19, 2001, Toronto, ON., Canada, pp: 339-348.

Do, H., G. Rothermel and A. Kinneer, 2006. Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis. Empirical Software Eng., 11: 33-70.

Jones, J.A., M.J. Harrold and J. Stasko, 2002. Visualization of test information to assist fault localization. Proceedings of the 24th International Conference on Software Engineering, May 19-25, 2002, Orlando, FL., USA., pp: 467-477.

Liblit, B., A. Aiken, A.X. Zheng and M.I. Jordan, 2003. Bug isolation via remote program sampling. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, June 8-11, 2003, San Diego, CA., USA., pp: 141-154.

Liu, C., L. Fei, X. Yan, J. Han and S.P. Midkiff, 2006. Statistical debugging: A hypothesis testing-based approach. IEEE Trans. Software Eng., 32: 831-848.

Masri, W., R. Abou-Assi, M. El-Ghali and N. Al-Fatairi, 2009. An empirical study of the factors that reduce the effectiveness of coverage-based fault localization. Proceedings of the 2nd International Workshop on Defects in Large Software Systems, July 19-23, 2009, Chicago, IL., USA., pp: 1-5.

Masri, W. and R. Abou-Assi, 2010. Cleansing test suites from coincidental correctness to enhance fault-localization. Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation, April 6-10, 2010, Paris, France, pp: 165-174.

Miao, Y., Z. Chen, S. Li, Z. Zhao and Y. Zhou, 2012. Identifying coincidental correctness for fault localization by clustering test cases. Proceedings of the 34th International Conference on Software Engineering, June 2-9, 2012, Zurich, Switzerland, pp: 267-272.

Naish, L., H.J. Lee and K. Ramamohanarao, 2011. A model for spectra-based software diagnosis. ACM Trans. Software Eng. Methodol., 20: 1-32.

Parnin, C. and A. Orso, 2011. Are automated debugging techniques actually helping programmers? Proceedings of the International Symposium on Software Testing and Analysis, July 17-21, 2011, Toronto, Canada, pp: 199-209.

Qi, Y., X. Mao, Y. Lei, Z. Dai, Y. Qi and C. Wang, 2013. Empirical effectiveness evaluation of spectra-based fault localization on automated program repair. Proceedings of the 37th Annual Computer Software and Applications Conference, July 22-26, 2013, Kyoto, Japan, pp: 828-829.

Richardson, D.J. and M.C. Thompson, 1993. An analysis of test data selection criteria using the relay model of fault detection. IEEE Trans. Software Eng., 19: 533-553.

Vessey, I., 1985. Expertise in debugging computer programs: A process analysis. Int. J. Man-Mach. Stud., 23: 459-494.

Voas, J.M., 1992. PIE: A dynamic failure-based technique. IEEE Trans. Software Eng., 18: 717-727.

Yan, S., Z. Chen, Z. Zhao, C. Zhang and Y. Zhou, 2010. A dynamic test cluster sampling strategy by leveraging execution spectra information. Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation, April 6-10, 2010, Paris, France, pp: 147-154.

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

Zhang, C., Z. Chen, Z. Zhao, S. Yan, J. Zhang and B. Xu, 2010. An improved regression test selection technique by clustering execution profiles. Proceedings of the 10th International Conference on Quality Software, July 14-15, 2010, Zhangjiajie, China, pp: 171-179.