A Multiple Fault Localization Approach based on Multicriteria Analytical Hierarchy Process

Noureddine Aribi*, Nadjib Lazaar†, Yahia Lebbah*, Samir Loudni‡, Mehdi Maamar*
*LITIO – University of Oran 1, 1524 El-M'Naouer, 31000 Oran – Algeria
†LIRMM – University of Montpellier, CNRS, 161 rue Ada, 34090 Montpellier – France
‡CNRS, UMR 6072 GREYC – University of Caen Normandy, 14032 Caen – France

***Abstract*—Fault localization problem is one of the most difficult processes in software debugging. Several spectrum-based ranking metrics have been proposed and none is shown to be empirically optimal. In this paper, we consider the fault localization problem as a multicriteria decision making problem. The proposed approach tackles the different metrics by aggregating them into a single metric using a weighted linear formulation. A learning step is used to maintain the right expected weights of criteria. This approach is based on Analytic Hierarchy Process (AHP), where a ranking is given to a statement in terms of suspiciousness according to a comparison of ranks given by the different metrics. Experiments performed on standard benchmark programs show that our approach enables to propose a more precise localization than existing spectrum-based metrics.**

***Index Terms*—Fault Localization; Spectrum-based Fault Localization; Multiple Fault; Multicriteria decision making; AHP**

I. Introduction

Developing software programs is universally acknowledged as an error-prone task. The major bottleneck in software debugging is how to identify where the bugs are [1], this is known as fault localization problem. Nonetheless, locating a fault is still an extremely time-consuming and tedious task. Over the last decade, several automated techniques have been proposed to tackle this problem.

Spectrum-based approaches. *Spectrum-based fault localization* (SBFL) (e.g. [2], [3]) is a class of popular fault localization approaches that take as input a set of failing and passing test case executions, and then highlight the suspicious program statements that are likely responsible for the failures. A *ranking metric* is used to compute a suspiciousness score for each program statement based on observations of passing and failing test case execution. The basic assumption is that statements with high scores, i.e. those executed more often by failed test cases but never or rarely by passing test cases, are more likely to be faulty. Several ranking metrics have been proposed to capture the notion of suspiciousness, such as Tarantula [4], Ochiai [5], and Jaccard [5]. The ultimate objective of SBFL is to have a metric able to always rank first the faulty statements. In practice, we are very far from this ideal [6]. SBFL metrics do not rely on a particular model of the program under test and thus, they are easy to use and practical in the presence of CPU time and memory resources constraints. SBFL metrics give different interpretation of suspiciousness degree. In addition, the semantics of statements and the dependencies are not taken into account. Thus, the accuracy of SBFL approaches is inherently limited.

Multiple fault programs. Most of current localization techniques are based on the *single fault hypothesis*. By dismissing such assumption, faults can be tightly dependent in a program, giving rise to numerous behaviours [7]. Thus, it makes the localization process difficult for multiple fault approaches [8]–[10]. The main idea of these approaches is to make a partition on test cases into fault-clusters where each one contains the test cases covering the same fault. The drawback is that a test case can cover many faults with overlapping clusters, which leads to a rough localization. Another idea consists in localizing one fault at a time [11]. Here, we start by locating a first fault, then correct it (which is an error-prone step), generate again new test cases, and so on until no fault remains in the program.

Artificial Intelligence based approaches. In the last decade, fault localization was abstracted as a data mining (DM) problem. Podgurski et al. present a method to automatically group faulty spectra with respect to the fault that leads to the failure [9]. This method relies on cluster analysis. Cellier et al. [12], [13] propose a combination of association rules and Formal Concept Analysis (FCA) to assist in fault localization. In [14], [15], the authors formalize the problem of fault localization as a closed pattern mining problem. A constraint programming model, with ClosedPattern global constraint, is used to extract the k best patterns in terms of suspiciousness degree. Other approaches tackle fault localization as a supervised learning problem [16], [17].

Multicriteria decision making. Many real-world decision problems involve several criteria. As soon as multiple conflicting criteria are considered in the evaluation of a decision, the notion of optimality is not workable, since no criterion is systematically better than all the others. In this context, the Multicriteria decision-making (MCDM) [18] provides a systematic approach to characterize and find the most-preferred trade-off solutions. While many preference models have been proposed in the literature, the additive preference model is the most commonly used in MCDM. It consists to aggregate additively the criteria into a single criterion, so as to take advantage of all advanced techniques in solving single-criterion optimization problems. A first difficult step in this direction consists to find the right weights. Fortunately, there are some existing methods that tackle, either directly or

indirectly, weights elicitation problem, in particular the AHP method described in Section III-B.

In this paper, we consider the fault localization problem as a multicriteria decision making problem. The proposed approach tackles the different metrics by aggregating them into a single metric using a weighted linear formulation. We propose a passive/active learning step to maintain the right expected weights of criteria. This approach is based on Analytic hierarchy Process (AHP), where a ranking is given to a statement in terms of suspiciousness according to a comparison of ranks given by the different metrics III-B. The approach is implemented in AHP-LOC. Experiments performed on standard benchmark programs show that our approach enables to propose a more precise localization than existing spectrum-based metrics.

This paper is organized as follows. Section 2 presents related work. Section 3 recalls preliminaries. Section 4 describes our approach. Section 5 illustrates the approach on a small program. Section 6 reports experimental results and a complete comparison with AHP-LOC and SBFL metrics. Finally, we conclude this work in Section 7.

II. RELATED WORK

To the best of our knowledge, Xuan and Monperrus propose in 2014 the first and unique work combining multiple ranking metrics [19]. MULTRIC is based on a passive learning process acting on multiple ranking metrics. MULTRIC consists of two phases: learning and ranking. It combines different ranking metrics in a weighted model. The learning phase is a passive one using a training set. The training set corresponds to a set of pairs of statements with their spectra. From each already dealt faulty program, only faulty statements and their uppers and lowers statements in terms of suspiciousness are considered. Then, pairs of faulty/non-faulty statements are extracted and added to the training set. Considering all possible pairs of faulty programs can lead to a very large training set. To bypass this limitation, MULTRIC uses a neighborhood strategy with few uppers and lowers statements of the faulty statement. Learning the weight of each metric is based on the assumption that given a pair of statements (s_f, s_n), where s_f is a faulty one and s_n is a non-faulty one, s_f should be ranked above s_n. The learning is also based on a standard binary classifier in machine learning. The ranking phase combines in a simple manner the scores of the different metrics with a weighting function and using the learned weights.

AHP-LOC has two distinguishing elements w.r.t., MULTRIC. Firstly, AHP-LOC is an adaptive approach based on an active learning makes it able to start with an empty training set. Secondly, AHP-LOC benefits from multicriteria AHP in the aggregation of different metrics.

III. BACKGROUND

A. Fault Localization

Let us consider a faulty program P_i having n_i lines, labeled $e_{i,1}$ to e_{i,n_i}. A **test case** $tc_{i,j}$ is a tuple $\langle D_{i,j}, O_{i,j}\rangle$, where $D_{i,j}$ is the input data and $O_{i,j}$ is the expected output. Let $\langle D_{i,j}, O_{i,j}\rangle$ a test case and $A_{i,j}$ be the current output returned by a program P after the execution of its input $D_{i,j}$. If $A_{i,j} = O_{i,j}$, $tc_{i,j}$ is considered as a *passing* (i.e. positive), *failing* (i.e. negative) otherwise. A **test suite** $T_i = \{tc_{i,1}, tc_{i,2}, ..., tc_{i,m_i}\}$ is a set of m_i test cases to check whether the program P_i follows a given set of requirements.

Given a test case $tc_{i,j}$ and a program P_i, the set of executed (at least once) statements of P with $tc_{i,j}$ is called a *test case coverage* $I_{i,j} = (I_{i,j,1}, ..., I_{i,j,n_i})$, where $I_{i,j,k} = 1$ if the k^{th} statement is executed, 0 otherwise. $I_{i,j}$ indicates which parts of the program are active during a specific execution.

SBFL techniques assign suspiciousness scores for each of statements and rank them in a descending order of suspiciousness. Most of suspiciousness metrics are defined manually and analytically on the basis of multiple assumptions on programs, test cases and the introduced faults. Fig 1 lists the formula of three well-known metrics: TARANTULA [4], OCHIAI [5] and JACCARD [5]. Given a statement $e_{i,j}$, $pass(T_i)$ (resp. $fail(T_i)$) is the set of all passed (resp. all failed) test cases. $pass(e_{i,j})$ (resp. $fail(e_{i,j})$) is the set of passed (resp. failed) test cases covering $e_{i,j}$. The basic assumption is that the program fails when the faulty statement is executed. Moreover, the whole of suspiciousness metrics shares the same intuition: the more often a statement is executed by failing test cases, and the less often it is executed by passing test cases, the more suspicious the statement is considered. Fig.1 shows the suspiciousness spectrum of the different metrics according to an up to $1K$ passing and/or failing test cases:

- TARANTULA allows some tolerance for the fault to be executed by passing test cases (see Fig.1a). However, this metric is not able to differentiate between statements that are not executed by passing tests. For instance, consider two statements $e_{i,j}$ and $e_{i,k}$ with $|pass(e_{i,j})| = |pass(e_{i,k})| = 0$, $|fail(e_{i,j})| = 1$ and $|fail(e_{i,k})| = 1000$: $e_{i,j}$ and $e_{i,k}$ have the same suspiciousness degree according to TARANTULA.
- OCHIAI came originally from molecular biology. The specificity of this metric is that it attaches a particular importance of the presence to a statement in the failing test cases (see Fig.1b).
- JACCARD has been defined to find a proper balance between the impact of passing/failing test cases on the scoring measure [5] (see Fig.1c).

B. Analytical Hierarchy Process (AHP) [20]

AHP is a simple and easy to use structured process for organizing and eliciting criteria weights. It involves three main steps:

1) The criteria are subjectively compared in a pairwise manner according to their respective weight w_i. The comparison is organized into a square matrix $A = [1..m, 1..m]$, where $A[i, j]$ is the relative importance of criterion i w.r.t., criterion j. The i^{th} criterion is better than the j^{th} criterion if $(A[i, j] > 1)$. In AHP, we have 9 degrees of dominance where $A[i, j]$ indicates

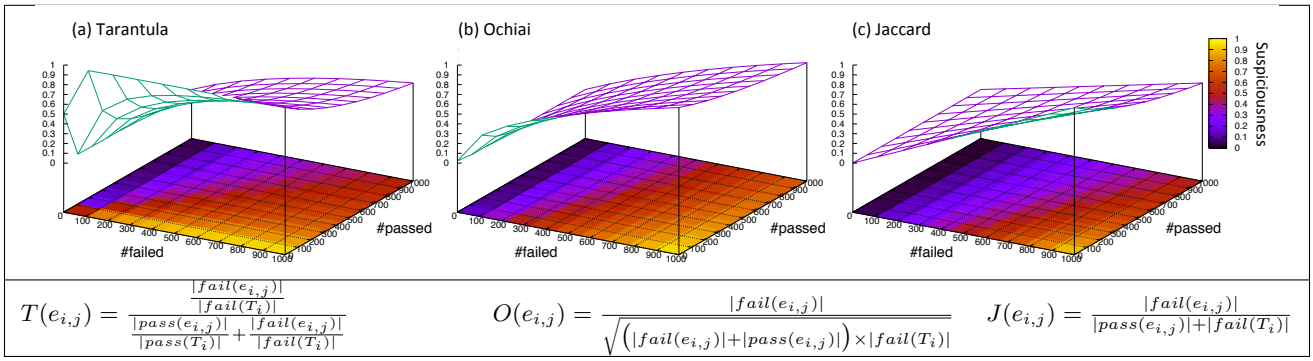

Fig. 1: Suspiciousness Degrees.

an indifference preference and $A[i,j] = 9$ a strong dominance. Note that $A[i,j] = 1/A[j,i]$ and $A[i,i] = 1$.

2) AHP assesses the criteria weighting vector w by solving the characteristic equation:

$$A \cdot w = \lambda_{max} \cdot w \tag{1}$$

where λ_{max} is the highest eigen value of A.

3) Inconsistencies may occur in pairwise comparisons, because AHP does not enforce the preferences to be transitive. For this reason, a consistency check is conducted by calculating the consistency ratio (CR).

$$CR = \frac{CI}{RI}, \qquad CI = \frac{(\lambda_{max} - n)}{n-1} \tag{2}$$

where RI is a constant taken from the Random Consistency Index table of AHP. The weighting vector w is considered as reliable if $CR < 0.1$.

The AHP can be combined with a popular direct weights elicitation method ROC (Rank-Order Centroid) [21]. ROC produces an estimation of the weights that minimizes the maximum error of each weight. The ROC method assumes that the true weights are uniformly distributed on the simplex of rank-order weights. That is, ROC is a function based on the average of the corners in the polytope defined by the simplex $S_w = w_1 > w_2 > \cdots > w_n, \sum_i w_i = 1$, and $w_i > 0$, such that:

$$w_i = \frac{1}{n} \sum_{k=i}^{n} \frac{1}{r_k} \tag{3}$$

where r_i is the rank of the i^{th} criterion.

IV. AHP-LOC APPROACH

Let $\{P_1, ..., P_n\}$ be a set of n faulty programs. A faulty program context is a triplet $(P_i, L_i, T_i), i = 1..n$, where P_i is a given faulty program, whose k faults are located at lines $L_i = \{L_{i_1}, ..., L_{i_k}\}$, and T_i is a test suite. Let $(M_1, M_2, ..., M_m)$ be some SBFL ranking metrics (e.g., Tarantula, Ochiai, Jaccard, etc.).

The aim of this paper is to propose an approach aggregating SBFL ranking metrics in a single SBFL ranking metric that takes benefit of their localization effectiveness. This is achieved thanks to AHP technique. AHP requires a squared matrix $A[1..m, 1..m]$, where $A[i,j]$ scores how much criteria i is ranked better than criteria j.

1) *Scoring a single SBFL technique.*
To evaluate how well the localization accuracy is, a suspiciousness score, denoted as EXAM score [11], is assigned to every faulty version of each subject program. The score defines the percentage of statements that need to be examined before the one locating the fault: lower is better. When running a single SBFL technique M_j on some faulty program context (P_i, L_i, T_i), it can return a set of equivalent statements in terms of suspiciousness (i.e., with the same suspiciousness degree). In this case, the effectiveness depends on which statement is to check first. For that reason, we consider two EXAM scores, the optimistic and the pessimistic one, denoted respectively $R^P_{i,j}$ and $R^O_{i,j}$. We talk about optimistic EXAM (resp. pessimistic exam) when the first (resp. last) statement to be checked in the set of equivalent statements is the faulty one. We also define a third metric, Δ EXAM $R^{\Delta}_{i,j} = (R^P_{i,j} - R^O_{i,j})$, representing the margin of the EXAM score, and middle EXAM $R^M_{i,j} = \frac{R^P_{i,j} + R^O_{i,j}}{2}$. In other words, Δ EXAM (middle exam) represents the distance between the optimistic and the pessimistic scores (resp. the average between the optimistic and the pessimistic scores).

2) *Comparing two SBFL techniques.*
Let M_j and M_l be two SBFL techniques to be compared on some faulty program context (P_i, L_i, T_i), by considering their pessimistic and optimistic results. Computing their middle gap $E^M_{i,j,l} = (R^M_{i,j} - R^M_{i,l})$ enables to know how much M_j is better than M_l: if $E^M_{i,j,l} \geq 0$, then M_l is better, else the converse. We can also consider their pessimistic or optimistic gaps, but the middle gap is preferred since that it aggregates them.
Finally, when running two SBFL techniques M_j and M_k on all of the faulty program contexts $(P_i, L_i, T_i), i = 1..n$, we can estimate the most efficient technique by averaging their middle gaps on all of the programs $AVG_{j,l} = \frac{1}{n} \sum_{i=1..n} E^M_{i,j,l}$.

3) *Computing AHP ranking matrix*

Algorithm 1: Learning

Input $D = \{\langle P_i, L_i, T_i\rangle | P_i$: faulty program, L_i: fault localions, T_i: test cases$\}$;
M set of m ranking metrics;
InOut A: AHP matrix;
foreach *metric* $M_j \in M$ **do**
  $A[j,j] \leftarrow 1$
  foreach $\langle P_i, T_i, L_i\rangle \in D$ **do**
    Compute $R^P_{i,j}$ using M_j
    Compute $R^O_{i,j}$ using M_j
$n \leftarrow 0$; $AVG_{*,*} \leftarrow 0$
foreach *pair of metrics* $M_j, M_l \in M$ **do**
  foreach $\langle P_i, T_i, L_i\rangle \in D$ **do**
    $E^P_{i,j,l} \leftarrow R^P_{i,j} - R^P_{i,l}$
    $E^O_{i,j,l} \leftarrow R^O_{i,j} - R^O_{i,l}$
    $AVG_{j,l} \leftarrow \frac{n}{n+1} AVG_{j,l} + \frac{1}{n+1}(E^P_{i,j,l} + E^O_{i,j,l})$
    $n \leftarrow n + 1$;
foreach $AVG_{j,l} : j, l \in \{1, \ldots, m\}$ **do**
  Scale $|AVG_{j,l}|$ to $[1, 9]$
  if $AVG_{j,l} < 0$ **then**
    $A[l,j] \leftarrow AVG_{j,l}$; $A[j,l] \leftarrow 1/AVG_{j,l}$;
  else $A[j,l] \leftarrow AVG_{j,l}$; $A[l,j] \leftarrow 1/AVG_{j,l}$;
return A;

Algorithm 1 generates the AHP matrix without any prior ranking of decision criteria, by exploiting pairs comparisons. It loops on all of the pairs of SBFL techniques: for all $j, l \in 1..m$, compute $AVG_{j,l}$. Than it computes a scaling of $m \times m$ comparisons $AVG_{j,l}$ to the permitted values of the AHP matrix A. As explained in section III-B, an indirect weight elicitation is performed with an indirect assessment of weights, which is obtained by computing the eigenvector w associated to the highest eigenvalue λ_{max}. This technique is far superior to any of the direct techniques due to its ability to capture the decision maker's trade-offs between criteria.
The consistency ratio CR (see formula 2) is checked to see in what extent the learned coefficients of A are plausible.

A. *Passive version of* AHP-LOC

Algorithm 1 enables to learn the AHP matrix from a training set composed of faulty program contexts $D = \{(P_1, L_1, T_1), ..., (P_n, L_n, T_n)\}$. The passive version of AHP-LOC consists of running Algorithm 1 on a an already fixed set of faulty program contexts. The performances of this passive approach depends on the quality of the given training set.

B. *Active version of* AHP-LOC

The passive approach is straightforward and simple, but it is challenging to ensure that the given training set is sufficient to start an efficient localization. In addition, it is not taking benefit of current/future localizations. We propose the active version of AHP-LOC where the learning process is dynamically done. Its principle is the following: as long as the current learning dataset implies an AHP matrix less efficient than some SBFL ranking metric, a new faulty program context is added to the learning dataset D of algo.1, updating the AHP matrix, and so on, until reaching an AHP matrix, which triggers the best localization matrix on the last added faulty program context.

C. *AHP ranking score*

Once the AHP matrix A is learned in a passive or active way, we proceed in computing the metrics weighting vector w. For that, we solve the equation 1. Afterward, we compute the score of a given statement e of a given faulty program using the following weighted aggregation function:

$$score_{AHP}(e) = \sum_{i=1}^{m} w_i \; scale(score_{M_i}(e)) \tag{4}$$

The $scale$ function is used to adjusting score values of the different scales metrics to a $[0, 1]$ scale.

V. RUNNING EXAMPLE

To illustrate our approach, we consider the *Power* program given in Fig.2. In this figure, we have six test cases where tc_1 to tc_3 are failing test cases, and tc_4 to tc_6 are passing test cases. According to the provided test cases, we report the suspiciousness ranking given by five ranking metrics ((1) AMPLE [22], (2) TARANTULA [4], (3) GP13 [23], (4) OCHIAI [5] and (5) JACCARD [5]) and our AHP-LOC approach. In this example, two faults are introduced at e_3 and e_4, where the correct statements are respectively "p = −y;" and "p = y;".

AHP-LOC learns the AHP matrix from faulty program examples in a passive/active way. In this example, we asked an expert to provide us with preferences/dominances between the different pairs of the five ranking metrics:

$$A = \begin{array}{c} \\ (1) \\ (2) \\ (3) \\ (4) \\ (5) \end{array} \begin{array}{c} \begin{array}{ccccc} (1) & (2) & (3) & (4) & (5) \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 9 & 9 \\ 1/2 & 1 & 2 & 3 & 7 \\ 1/3 & 1/2 & 1 & 2 & 4 \\ 1/9 & 1/3 & 1/2 & 1 & 2 \\ 1/9 & 1/7 & 1/4 & 1/2 & 1 \end{pmatrix} \end{array} \tag{5}$$

It is important to stress here that: First, it is not always possible to obtain the opinion of an expert. Second, an expert will not be able to provide such matrix in an AHP context when the number of criteria exceeds 10 [20]. Third, the expert is facing a challenging tast where its opinion heavily depends on the provided test cases. The learning step of AHP-LOC is proposed to bypass this limitation. We learn the AHP matrix from faulty programs contexts without the help of any expert.

Returning to our example, we compute the weighting vector w according to the equation 1. This leads us to $w = \langle 0.49, 0.25, 0.15, 0.07, 0.04\rangle$. Now, for each statement we are able to give a ranking AHP score using equation 2.

The weighting vector $w_{AHP} = \langle 0.49, 0.25, 0.15, 0.07, 0.04\rangle$ was generated using the AHP matrix (5), with a consistency ration $CR = 0.018 < 0.1$.

Fig.2 shows how the ranking metrics can be different with completely different rankings. The fault at e_4 is ranked first except for GP13 and OCHIAI. As they give a greater

prominence to the failing test cases, e_4 is ranked at the third position. The aggregation done by AHP-LOC ranks e_4 at the first position as well. For the fault introduced at e_3, TARANTULA, OCHIAI and GP13 rank it in the next-to-bottom place. The localization with AMPLE is able to rank it at the first position. But this localization is not accurate, since we have a big equivalent class. What is interesting here is that AHP aggregation ranks it at the second position. Here, AHP-LOC shows its best case by taking benefits of the difference that exists among the ranking metrics.

VI. EXPERIMENTS

This section describes the experimental settings (including benchmark programs, protocol and implementation), the experimental results and comparison with eight SBFL ranking metrics. Experiments were performed on single and multiple fault programs.

A. Benchmark programs

We evaluate our approach by analyzing the performance of fault localization over 18 faulty programs coming from different real-world applications. As our approach does not require a specific programming language in order to be applied, we investigated two types of programming paradigms, namely C and Java Object Oriented programs.

Siemens and Space datasets. We have considered both Siemens and Space datasets.[1] A complete description of Siemens suite and Space dataset can be found in [24], [25]. These C programs are the most common program suites used to evaluate software testing and fault localization approaches. The Siemens suite + Space are provided with eight C programs, each one has a correct version and a set of faulty versions (one fault per version). The suite is also provided with test suites for each faulty version.

Table I summarizes the 212 faulty programs. For each program, we report the number of faulty versions (single fault `1F`, two faults `2F` and four faults `4F`), the size of the program with its lines of code (LOC) and lines of executable code (LEC), the number of test cases. We have 139 versions with single fault, 47 with two faults and 26 versions with four faults. The single fault versions are provided, where multiple fault versions are produced by combining randomly the provided faults.

For C programs and to know the statements that are covered by a given (passing/failing) test case, we used GCOV [2] profiler tool to trace and save the coverage information of test cases as a boolean matrix (e.g., see Fig.2). Then, each test case is classified as positive/negative w.r.t. the provided correct version.

Java Object-Oriented datasets We evaluate our approach on 25,386 faults in ten Java Open-Source softwares (see Table II).[3] We used the granularity of methods (rather than the more common statements, as we did in Siemens suite). In fact, methods have the advantage of giving a natural context of each fault, and of being the natural skip/step into units of contemporary debuggers (guided by the output of the fault locator) [26].

TABLE I: Siemens suite.

Program	#Versions (`1F`, `2F`, `4F`)	LOC	LEC	Test cases
Replace	(29, 6, 3)	514	245	5,542
PrintTokens2	(9, 6, 3)	358	200	4,056
PrintTokens	(4, 6, 3)	348	195	4,071
Schedule	(5, 6, 3)	294	152	2,650
Schedule2	(8, 6, 3)	265	128	2,680
TotInfo	(19, 6, 3)	272	123	1,052
Tcas	(37, 6, 3)	135	65	1,578
Space	(28, 5, 5)	9,126	3,657	13,585
Total	**(139, 47, 26)**	**11,312**	**4,765**	**35,214**

LOC: lines of code in the correct version
LEC: lines of executable code

TABLE II: Java Open-source projects.

Program	# Versions (`1F`, `2F`, `4F`)	Methods*	MUT	Test cases
Daikon 4.6.4	(352, 10^3,10^3)	14,387	1,936	157
Eventbus 1.4	(577,10^3,10^3)	859	338	91
Jaxen 1.1.5	(600,10^3,10^3)	1,689	961	695
Jester 1.37b	(411,10^3,10^3)	378	152	64
JExel 1.0.0b13	(537,10^3,10^3)	242	150	335
JParsec 2.0	(598,10^3,10^3)	1,011	893	510
AC Codec 1.3	(543,10^3,10^3)	265	229	188
AC Lang 3.0	(599,10^3,10^3)	5,373	2,075	1,666
Eclipse Draw2d 3.4.2	(570,10^3,10^3)	3,231	878	89
HTML Parser 1.6	(599,10^3,10^3)	1,925	785	600
Total	**(5386,** 10^4**,** 10^4**)**	**29,360**	**8,397**	**4,395**

MUT: Methods under test
* The total number of methods, not counting JUnit test methods

Table II shows the details of the different projects, including the number of faulty programs (single fault `1F`, two faults `2F` and four faults `4F`), the number of methods (excluding JUnit test methods), the number of methods under test, and the number of test cases.

We have 5,386 single fault Java programs, 10^4 programs with two faults and 10^4 programs with four faults. All are provided by the distribution. The multiple faults are spread over different methods.

B. Experimental protocol

Some statements can be equivalent in terms of suspiciousness. Here, the accuracy may vary depending on which statement to check first. For such reason, we report the two exam scores, the *optimistic* and the *pessimistic* one, denoted respectively in this section O-EXAM and P-EXAM. We recall that we talk about O-EXAM (resp. P-EXAM) when the first (resp. last) statement to check in the set of equivalent statements is the faulty one. We also use Δ-EXAM = O-EXAM − P-EXAM, representing the range of the EXAM score.

Our approach AHP-LOC is trained on faulty programs to learn the coefficients of the AHP matrix (training set). To evaluate how accurately the learned AHP matrix performs, the resulting model is validated on the remaining part of the faulty

[1] sir.unl.edu/php/previewfiles.php
[2] https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[3] http://www.feu.de/ps/prjs/EzUnit/eval/ISSTA13

Program : Power	Test cases tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	Tarantula	Ochiai	Ample	Jaccard	GP13	AHP-Loc
`float pow (int x, int y) {` `int i, p;`	$x=2$ $y=3$	2 -4	3 0	-2 -2	3 3	1 -2						
e_1: `i = 0;`	1	1	1	1	1	1	2	1	4	1	1	4
e_2: `float result = 1.0 ;`	1	1	1	1	1	1	2	1	4	1	1	4
e_3: `if ( y < 0 ) p = -x ; //fault1`	0	1	0	1	0	1	6	6	1	6	6	2
e_4: `else p = x ; //fault2`	1	0	1	0	1	0	1	3	1	1	3	1
e_5: `while ( i < p) { result = result * x ;`	1	0	1	1	1	0	2	4	4	4	4	6
e_6: `i = i + 1 ; }`	1	0	1	1	1	0	2	4	4	4	4	6
e_7: `if ( y < 0 ) result = 1 / result ;`	0	1	0	1	0	1	6	6	1	6	6	2
`return result ; }`												
Passing/Failing	F	F	F	P	P	P						

Fig. 2: "Power" program containing two faults.

programs (testing set). To avoid bias of random selection, we performed a 6-folds cross-validation of the data using different partitions. In each fold, we randomly select three faulty versions to form the training set and use the remaining 15 faulty versions for evaluation (testing set). We report the performance result using averaged EXAM score over the folds.

We compared our three AHP-LOC versions **Active** AHP-LOC, **Passive** AHP-LOC and ROC based AHP-LOC, with eight widely-studied spectrum-based ranking metrics, TARANTULA, OCHIAI, JACCARD, M2 and AMPLE, and three recent proposed ranking metrics including GP13, NAISH2 and ER1B, which are proved as the optimal under theoretical assumptions (cf. [27]).

C. Implementation

We have implemented our AHP-LOC versions (active, passive and ROC) in C++. For a fair comparison between our tool and the other approaches, we have implemented the SBFL metrics, GP13, NAISH2, AMPLE, ER1B, M2, TARANTULA, OCHIAI and JACCARD in C++ as presented in [4], [5]. Our experiments were conducted on an Intel® i5-2400 CPU at 3.10GHz x 4 with 8 GB RAM.

D. Single Fault Results

Table III reports an EXAM score comparison (O-EXAM, P-EXAM and Δ-EXAM) between **Active** AHP-LOC, **Passive** AHP-LOC, ROC AHP-LOC and SBFL metrics on single fault programs. The first observation that we can draw is that, comparing our three versions, the **Active** AHP-LOC achieves the lowest O-EXAM and P-EXAM. We also observe that AHP-LOC (with its three versions) is more effective than SBFL approaches in most of the benchmarking instances.

Now, when examining Fig.3 we can see that **Active** AHP-LOC provides the narrowest Δ-EXAM (i.e. the range of statements to be inspected in vain before reaching the faulty statement is minimal). Remarkably, the P-EXAM (4.91 %) of **Active** AHP-LOC is even lower that the O-EXAM of all the metrics. It is also clear from the same figure, that metrics shown in red (i.e. (5) TARANTULA, (7) JACCARD and (10) AMPLE) perform poorly, as they generate big class of suspicious statements (see also Tables III, Δ-EXAM). Interestingly, **Passive** AHP-LOC, AHP-ROC, ER1B and NAISH2 almost achieve similar better accuracy.

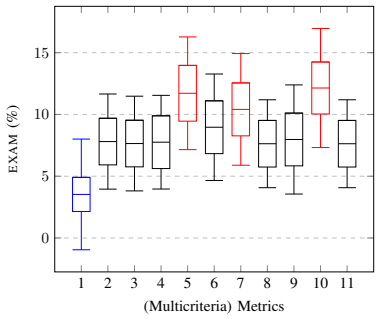

Fig. 3: Qualitative comparison for single fault programs. **(1): Active** AHP-LOC **(2): Passive** AHP-LOC **(3):** AHP-ROC **(4):** GP13 **(5):** TARANTULA **(6):** OCHIAI **(7):** JACCARD **(8):** NAISH2 **(9):** M2 **(10):** AMPLE **(11):** ER1B

E. Multiple Fault Results

In this section, we report the EXAM score comparison results on programs with multiple faults. The results for two and four faults are shown in Table III.

Let us start with the two faults programs. Our first observation is that the **Active** AHP-LOC is drastically more efficient than all SBFL metrics in terms of P-EXAM, O-EXAM and Δ-EXAM. This is indicative of high effectiveness, and thus an improved localization on most subject programs (C and Java). However, the SBFL metrics, namely, GP13 and TARANTULA show the opposite behavior, which is indicative of a low effectiveness and large wasted debugging effort.

Fig.4 shows a comparison on P-EXAM (fig.4a) and O-EXAM (fig.4b) between SBFL metrics, and AHP-LOC. When focusing on the exam score required to localize both faults (f_1 and f_2), we can see that **Active** AHP-LOC is the most accurate (quickly locates the last fault). Moreover, the **Active** AHP-LOC improve the localization accuracy by 45% for the O-EXAM and more than 47% for the P-EXAM compared to the best localization score given by ER1B metric. Here, we notice that the **Passive** AHP-LOC and AHP-ROC enable to better locate both of faults compared with SBFL metrics. Fig.4 also shows that, in some cases, the standard metric can be effective. For instance, the NAISH2 and ER1B report similar accuracy and are able to capture quickly the first fault compared to the remaining metrics.

TABLE III: Qualitative comparison for multiple faults on Siemens Suite (EXAM score %). **(1): Active** AHP-LOC **(2): Passive** AHP-LOC **(3):** AHP-ROC **(4):** GP13 **(5):** TARANTULA **(6):** OCHIAI **(7):** JACCARD **(8):** NAISH2 **(9):** M2 **(10):** AMPLE **(11):** ER1B

Program		P-EXAM (%)										
		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
A	*f1*	**4.91**	9.69	9.53	9.89	13.97	11.10	12.55	9.52	10.11	14.25	9.52
B	*f1*	**15.26**	25.19	24.42	36.09	35.46	31.91	33.84	27.94	32.94	34.03	27.94
	f2	**15.39**	28.95	28.57	38.03	40.26	39.71	40.73	29.35	37.04	37.86	29.35
C	*f1*	**24.45**	27.10	34.86	45.33	31.04	29.93	28.92	35.04	38.42	30.03	35.05
	f2	**24.78**	27.20	36.87	45.53	33.57	43.05	37.26	35.09	44.06	31.44	35.09
	f3	**24.98**	27.30	38.27	45.83	40.14	43.95	48.75	35.80	49.75	39.58	35.80
	f4	**25.11**	27.96	41.25	53.30	47.55	49.35	51.63	43.02	52.00	44.37	43.02
Global		**19.27**	24.77	30.54	39.14	34.57	35.57	36.24	30.82	37.76	33.08	30.82
Program		O-EXAM (%)										
		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
A	*f1*	**2.15**	5.92	5.76	5.62	9.46	6.83	8.27	5.74	5.84	10.04	5.74
B	*f1*	**12.52**	19.48	18.71	20.40	19.56	21.20	23.14	22.23	22.25	27.86	22.23
	f2	**12.63**	22.63	22.26	21.28	23.30	27.94	28.96	23.04	25.29	30.64	23.04
C	*f1*	20.43	23.05	32.45	29.87	15.30	14.43	**13.43**	32.45	22.95	26.93	32.35
	f2	20.64	23.07	34.27	30.06	**17.30**	27.20	21.78	32.68	28.60	29.02	32.45
	f3	**20.64**	23.30	35.88	30.76	24.38	28.45	32.90	33.41	34.68	36.95	33.41
	f4	**21.11**	23.80	38.17	37.47	32.21	34.25	36.54	39.94	36.17	41.61	39.95
Global		**15.73**	20.18	26.79	25.06	20.22	22.90	23.57	27.07	25.11	29.01	27.02
Program		Δ-EXAM (%)										
		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
A	*f1*	**2.76**	3.77	3.77	4.27	4.52	4.27	4.28	3.77	4.27	4.21	3.77
B	*f1*	**2.74**	5.71	5.71	15.69	15.90	10.71	10.71	5.71	10.69	6.17	5.71
	f2	**2.76**	6.31	6.31	16.75	16.95	11.77	11.77	6.31	11.75	7.22	6.31
C	*f1*	4.02	4.05	**2.41**	15.47	15.73	15.50	15.49	**2.41**	15.47	2.43	2.74
	f2	4.14	4.13	2.60	15.46	16.27	15.86	15.48	**2.59**	15.46	3.10	2.60
	f3	4.34	4.00	**2.39**	15.07	15.76	15.49	15.86	**2.39**	15.07	2.62	**2.39**
	f4	4.00	4.16	3.08	15.83	15.34	15.10	15.09	3.08	15.83	**2.76**	3.07
Global		**3.54**	4.59	3.75	14.08	14.35	12.67	12.67	3.75	12.65	4.07	3.80

A: 1 fault B: 2 faults C: 4 faults

Interestingly, the pessimistic EXAM score of **Active** AHP-LOC is nearly better than the optimistic score of all metrics (including the **Passive** AHP-LOC and AHP-ROC). In second position comes the AHP-based multicriteria approaches, with the best EXAM scores against the SBFL metrics.

For four-faults programs, here the results presented in Fig.5 and Table III support our previous observations on the comparison between metrics and our AHP-based approaches. In fact, our approach significantly dominates all the SBFL approaches by capturing quickly all faults with high accuracy, especially according to the P-EXAM score. Regarding the O-EXAM score, this observation remains true for both the third and fourth faults (i.e. f_3 and f_4).

Concerning the first two faults, i.e. f_1 and f_2, the result depicted in Fig.5c needs to be clarified and analyzed in depth. Indeed, standard SBFL metrics, namely, TARANTULA, OCHIAI and JACCARD seem to provide the best result. However, our approach is rather good in term of effectiveness, because those metrics give a large class of equivalent suspicious statements. To support this justification, we consider an important information given by the median of the box plots, which represent the average EXAM score of P-EXAM and O-EXAM.

Overall, by observing the global EXAM scores given in Table III, the **Active** AHP-LOC remains the best localization approach for programs with one, two and four faults.

VII. CONCLUSION

In this paper we have proposed a new fault localization method based on AHP (Analytic Hierarchy Process) multicriteria based approach, which aggregates spectrum-based ranking metrics. We propose a passive and active learning versions to maintain the right expected weights of criteria.

We have compared experimentally our approach with state of the art SBFL metrics on a set of multiple faults programs. The results we obtained show that our approach enables to aggregate the benefits of various SBFL metrics to get a single efficient SBFL technique. As future works, we plan to prospect other learning strategies to boost even more the performances of our AHP aggregation multicriteria approach.

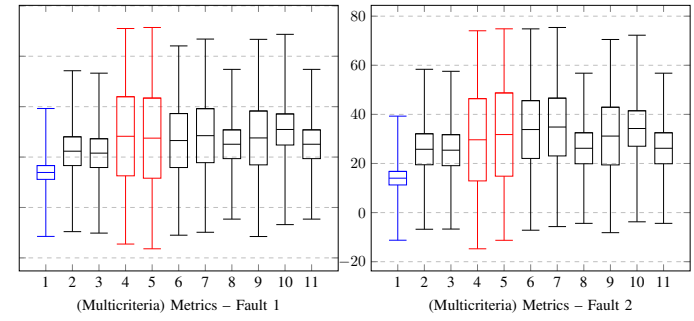

Fig. 4: Qualitative comparison for two faults programs. **(1): Active** AHP-LOC **(2): Passive** AHP-LOC **(3):** AHP-ROC **(4):** GP13 **(5):** TARANTULA **(6):** OCHIAI **(7):** JACCARD **(8):** NAISH2 **(9):** M2 **(10):** AMPLE **(11):** ER1B

REFERENCES

[1] I. Vessey, "Expertise in debugging computer programs: A process analysis," *International Journal of Man-Machine Studies*, vol. 23, no. 5, pp. 459–494, 1985.

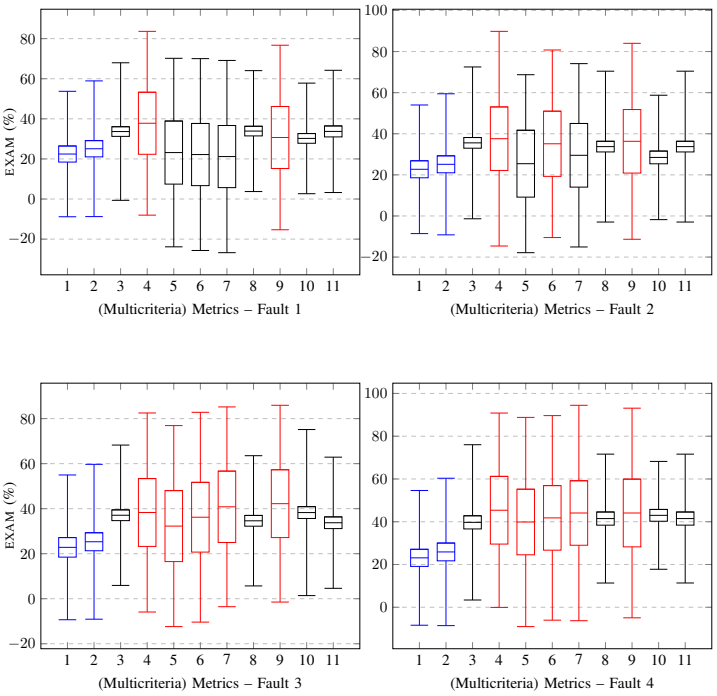

Fig. 5: Qualitative comparison for four faults programs. **(1): Active** AHP-LOC **(2): Passive** AHP-LOC **(3):** AHP-ROC **(4):** GP13 **(5):** TARANTULA **(6):** OCHIAI **(7):** JACCARD **(8):** NAISH2 **(9):** M2 **(10):** AMPLE **(11):** ER1B

[2] R. Abreu, P. Zoeteweij, R. Golsteijn, and J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.06.035

[3] J. A. Jones, M. J. Harrold, and J. T. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA*, 2002, pp. 467–477.

[4] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*, 2005, pp. 273–282.

[5] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the accuracy of spectrum-based fault localization," in *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007*, Sept 2007, pp. 89–98.

[6] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[7] V. Debroy and W. E. Wong, "Insights on fault interference for programs with multiple bugs," in *ISSRE 2009, 20th International Symposium on Software Reliability Engineering, Mysuru, Karnataka, India, 16-19 November 2009*. IEEE Computer Society, 2009, pp. 165–174.

[8] P. Agarwal and A. P. Agrawal, "Fault-localization techniques for software systems: a literature review," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 5, pp. 5:1–5:8, 2014.

[9] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," in *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, 2003, pp. 465–477. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2003.1201224

[10] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in parallel," in *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ser. ISSTA '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1273463.1273468

[11] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Software Eng.*, vol. 42, no. 8, pp. 707–740, 2016. [Online]. Available: https://doi.org/10.1109/TSE.2016.2521368

[12] P. Cellier, M. Ducassé, S. Ferré, and O. Ridoux, "Dellis: A data mining process for fault localization," in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, USA, July 1-3, 2009*, 2009, pp. 432–437.

[13] ——, "Multiple fault localization with data mining," in *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Eden Roc Renaissance, Miami Beach, USA, July 7-9, 2011*, 2011, pp. 238–243.

[14] M. Maamar, N. Lazaar, S. Loudni, and Y. Lebbah, "Fault localization using itemset mining under constraints," *Autom. Softw. Eng.*, vol. 24, no. 2, pp. 341–368, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10515-015-0189-z

[15] N. Aribi, M. Maamar, N. Lazaar, Y. Lebbah, and S. Loudni, "Multiple fault localization using constraint programming and pattern mining," in *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, 2017, pp. 860–867.

[16] S. Ali, J. H. Andrews, T. Dhandapani, and W. Wang, "Evaluating the accuracy of fault localization techniques," in *ASE 2009, 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, November 16-20, 2009*, 2009, pp. 76–87. [Online]. Available: http://dx.doi.org/10.1109/ASE.2009.89

[17] Y. Tian, D. Wijedasa, D. Lo, and C. Le Goues, "Learning to rank for bug report assignee recommendation," in *24th IEEE International Conference on Program Comprehension, ICPC 2016, Austin, TX, USA, May 16-17, 2016*, 2016, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2016.7503715

[18] J. Figueira, S. Greco, and M. Ehrgott, *Multiple Criteria Decision Analysis: State of the Art Surveys*. Boston, Dordrecht, London: Springer Verlag, 2005. [Online]. Available: http://www.springeronline.com/sgw/cda/frontpage/0,11855,5-165-22-34954528-0,00.html

[19] J. Xuan and M. Monperrus, "Learning to combine multiple ranking metrics for fault localization," in *ICSME*. IEEE Computer Society, 2014, pp. 191–200.

[20] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *European Journal of Operational Research*, vol. 48, no. 1, pp. 9 – 26, 1990, desicion making by the analytic hierarchy process: Theory and applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0377221790900571

[21] F. H. Barron and B. E. Barrett, "Decision quality using ranked attribute weights," *Manage. Sci.*, vol. 42, no. 11, pp. 1515–1523, Nov. 1996. [Online]. Available: http://dx.doi.org/10.1287/mnsc.42.11.1515

[22] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight bug localization with AMPLE," in *Proceedings of the Sixth International Workshop on Automated Debugging, AADEBUG 2005, Monterey, California, USA, September 19-21, 2005*, 2005, pp. 99–104. [Online]. Available: http://doi.acm.org/10.1145/1085130.1085143

[23] S. Yoo, "Evolving human competitive spectra-based fault localisation techniques," in *Search Based Software Engineering - 4th International Symposium, SSBSE 2012, Riva del Garda, Italy, September 28-30, 2012. Proceedings*, 2012, pp. 244–258. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33119-0_18

[24] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Engg.*, vol. 10, no. 4, pp. 405–435, Oct. 2005. [Online]. Available: http://dx.doi.org/10.1007/s10664-005-3861-2

[25] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria," in *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press, 1994, pp. 191–200.

[26] F. Steimann, M. Frenkel, and R. Abreu, "Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA)*, ACM. Lugano, Switzerland: ACM, July 2013, p. 314–324.

[27] 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. 11:1–11:32, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2000791.2000795