Noname manuscript No.
(will be inserted by the editor)

FTMES@r: A Mutation Execution Strategy Method to Improve the Top@r Positions of Spectrum-based Fault Localization Techniques

André Assis Lôbo de Oliveira
· **Celso Gonçalves Camilo-Junior**
· **Eduardo Noronha de Andrade Freitas**
· **Auri Marcelo Rizzo Vincenzi**

Received: date / Accepted: date

Abstract Fault localization has been one of the most manual and costly software debugging activities. The spectrum-based fault localization is the most studied and evaluated fault localization approach. Mutation-based fault localization is a promising approach but with a high computational cost. We propose the FTMES@r: a fault localization method that uses the MBFL approach to improve the top@r ranking positions of SBFL techniques efficiently. Our assumption is that the SBFL can select the fault until the top@r position from all ranked statements and the MBFL approach improves the efficacy of localization this smaller ranking efficiently. Our proposed approach and eight other baselines were evaluated against 221 real faults. The results show that FTMES@r has the best cost-benefit relationship in terms of runtime approximation and efficacy improvements.

1 Introduction

Terms
Efficiency: the capacity of a technique to reduce the amount of mutant executions.
Efficacy: the capacity of a technique to rank faulty statements in the initial positions of the suspicious list.

According to the National Institute of Standards and Technology (NIST), software faults incur more than $59 billion in annual economic losses to U.S.

André Assis Lôbo de Oliveira
Alameda Palmeiras, Quadra D, Câmpus Samambaia,
Instituto de Informática (INF), Universidade Federal de Goiás
CEP 74690-900 - Goiânia - GO
Phone: +55 (62) 3521-1181
Fax: +55 (62) 3521-1181
E-mail: andre.assis.lobo@gmail.com

users (NIST, 2002), and is increasing annually (LIU et al., 2017). Faced with this, various debugging approaches have been proposed to find and fix bugs of software (OLIVEIRA et al., 2018). Numerous researchers have focused on fault localization techniques that use a given test set to identify defective program elements (PAPADAKIS; TRAON, 2014). However, it is one of the most expensive, tedious and time-consuming process in the whole debugging activity, driving research community to automate these activities.

Amongst fault localization (FL) techniques (JONES; HARROLD, 2005; WONG et al., 2012, 2014; XUAN; MONPERRUS, 2014), the most studied and evaluated are Spectrum-based (SBFL) (FREITAS et al., 2018a; ABREU; ZOETEWEIJ; GEMUND, 2007). Most fault localization techniques generate a suspiciousness score $S(s)$ for each element s. Based on this score, a decreasing suspiciousness list is generated from software elements such as statements. The technique is more effective when faulty statements lie on the top of this ranking.

Another fault localization approach is the Mutation-based Fault Localization (MBFL) which is based on the Mutation Analysis technique (DEMILLO; LIPTON; SAYWARD, 1978) and aims efficacy improvement of suspicious elements list for faulty software debugging (FREITAS et al., 2018b; MOON et al., 2014; PAPADAKIS; TRAON, 2012, 2014, 2015). However, Mutation Analysis is a high computational cost technique. Hence, it is important to investigate optimization opportunities in MBFL techniques.

MBFL techniques face two conflicting objectives: improving early fault localization (efficacy), and reducing computational cost (efficiency). For example, the MCBFL-hybrid-avg (PEARSON et al., 2017) is a hybrid technique (SBFL and MBFL) and is a technique with high efficacy. However, this MBFL technique spent between 32 to 168 CPU hours (PEARSON et al., 2017) to perform complete Mutation Analysis on some real scenarios (JUST; JALALI; ERNST, 2014).

Several cost reduction techniques have been proposed aiming at improving MBFL (PAPADAKIS; TRAON, 2014, 2012; GONG; ZHAO; LI, 2015; LIU et al., 2017). The Dynamic Mutation Execution Strategy (DMES) (GONG; ZHAO; LI, 2015) is a recent approach that aims to make MBFL more efficient by optimizing the number of mutant executions (we note that DMES was validated only with artificial faults).

Another shared problem among the fault localization techniques is the usability of suspiciousness ranking. In practice, the programmers not use the ranking sequentially, as suggested. This phenomena is called as “zigzag effect” (PARNIN; ORSO, 2011). This happen due the statements dependency and the size of ranked list. 有用的、灵验的

The SBFL techniques are less efficacious compared to the best MBFL technique (MCBFL-hybrid-avg). However, the SBFL techniques reached good results for many real scenarios positioning the fault statements in the top of ranking with little distance compared to the best MBFL technique (PEARSON et al., 2017). Furthermore, the SBFL is more scalable and efficient technique. Before this, other no explored opportunities exist for optimization. For exam-

ple, the use of the top statements ranked by SBFL technique to be improved with the further application of MBFL technique considering the approximated effectiveness of SBFL.

In this context, we propose the FTMES@r: a method that uses the MBFL approach to improve the top@r ranking positions of SBFL techniques efficiently. Our assumption is that the SBFL can select the fault until the top@r position from all ranked statements and the MBFL approach improves the efficacy of locating this smaller ranking efficiently. We call SFilter@r such process that uses the SBFL's efficacy of localization to filter the top@r statements from SBFL's ranking. Furthermore, we purpose the MBFL application over top@r statements to generate a new smaller and improved Ranking@r providing it to the programmer. The FTMES@r framework reduces the Mutation Analysis cost with two components: i) SFilter@r and ii) FTMES. SFilter@r is responsible to filter the statements and FTMES is the mutation execution strategy that optimizes the mutants executions.

We conducted empirical studies of our approach considering two analysis: i) the FTMES component's evaluation as mutation execution strategy that optimizes the MBFL approaches comparing with other baselines and; ii) the FTMES@r method's analysis presenting results for SFilter@r and FTMES interaction. The comparison considers the following techniques: the mutation execution strategy DMES, the SBFL class and; as well as the best current MBFL techniques (PEARSON et al., 2017). We use the Defects4J benchmark (JUST; JALALI; ERNST, 2014) to evaluate the solutions of all techniques.

The main contributions of this article are:

1. A novel statement selection approach (SFilter@r) that uses the SBFL technique's efficacy to improve the MBFL's cost reduction.
2. The use of mutation execution strategy to improve the efficacy of localization for SBFL's ranking.
3. We perform empirical experiments to evaluate the FTMES@r that demonstrates a better cost-benefit relationship among the MBFL's techniques in terms of runtime approximation and efficacy improvement compared to SBFL technique performance.

We formulate three research questions to guide the experiments:

(RQ1) Is FTMES more efficient than the current mutation execution strategies?

(RQ2) Is FTMES really as effective as other techniques while being more efficient?

(RQ3) Does FTMES@r improve the fault localization activity producing:
- **(RQ3.1)** - a lower computational cost for MBFL techniques?
- **(RQ3.2)** - a highest efficacy of fault localization for SBFL's Ranking?
- **(RQ3.3)** - a better cost-benefit relationship for MBFL techniques compared to SBFL's performance?

We answer the RQ1 and RQ2 evaluating the FTMES component before the current mutation execution strategies. The results show that FTMES com-

ponent is the best current mutation execution strategy and is an efficacious technique compared to the best ones.

The RQ3 evaluates the FTMES@r improvements to SBFL's efficacy and MBFL's cost by interaction between SFilter@r and FTMES components. The answer to RQ3.1 shows that SFilter@r made MBFL more efficient reaching a higher reduction with FTMES component, as FTMES@r guides. The answer to RQ3.2 describes the SBFL's Ranking improvement was performed by MFBL techniques. Finally, the RQ3.3 presents the FTMES@r approach reaching the best cost-benefit relationship in terms of runtime approximation and efficacy SBFL's improvement.

This paper is structured as follows: Section 2 presents FL techniques and the MBFL techniques are explained in Subsections 2.2 e 2.3. Section 3 presents our approach. The experiments and its configurations are presented in Section 4, and the Sections 5 and 6 show the results and related works, respectively. Section 7 describes the threats of validity. Finally, Section 8 summarizes the proposed ideas and suggest future works.

2 Fault Localization Techniques

2.1 Spectrum-based Fault Localization techniques

Spectrum-based Fault Localization (SBFL) techniques make use of the frequency of executions from statements of a program to generate an associated *suspiciousness* value. In general, when there is a high frequency of execution in failed test cases and a low frequency of execution in passed test cases, the suspiciousness $S(s)$ of a statement is high. Table 1 presents five SBFL techniques largely studied in the literature. $totalpassed$ is the number of passed test cases and $passed(s)$ is the number of test cases which execute a statement s. Likewise, $totalfailed$ and $failed(s)$ are defined.

这里有问题，表1只列举了3个经典的频谱算法

2.2 Mutation-based Fault Localization Approach

The Mutation-based Fault Localization (MBFL) technique applies *Mutation Analysis (MA)* (DEMILLO; LIPTON; SAYWARD, 1978) in fault localization process. It involves assigning suspiciousness score to mutants in the hypothesis that test cases that kill mutants contribute more to the accuracy of values for the rank of statements. A test case *kills* a mutant when it detects a difference between the outputs produced by the original program and the mutant program after its execution. Despite the variations among different existing MBFL techniques, an MBFL approach can be implemented following these four steps: i) classify test cases; ii) generate and execute mutants; ii) compute suspiciousness; and iv) create the rank of statements.

Classify test cases: initially, T executes P, where P is a Program Under Test (PUT) and T its set of test. Next, the coverage information and test

Table 1: SBFL and MBFL formulas.

Technique	Formula
Ochiai:	$S(s) = \frac{failed(s)}{\sqrt{totalfailed.(failed(s)+passed(s))}}$
Dstar:	$S(s) = \frac{failed(s)^*}{\sqrt{passed(s)+(totalfailed-failed(s))}}$
Op2:	$S(s) = failed(s) - \frac{passed(s)}{totalpassed+1}$
Metallaxis:	$\mathrm{S(m)} = \frac{failed(m)}{\sqrt{totalfailed.(failed(m)+passed(m))}}$

results (FAIL or PASS) are collected. Then, T is split into two sets: T_p and T_f, where T_p are the passed test cases and T_f are the failed test cases. Besides that, $cov(T_f)$ is the set of covered statements by T_f.

Generate and Execute mutants: the MBFL techniques use $cov(T_f)$ to generate the mutants. In general, a statement s has a set of mutants, denoted by $M(s)$. The test cases are executed against each mutant, and the killing information are stored. Then, T is split into two groups: T_n and T_k, where T_n is the set that *does not kill the mutant* m,and T_k *kills* m. A mutation execution strategy (GONG; ZHAO; LI, 2015) optimizes the step of test case executions.

Compute the suspiciousness: the suspiciousness of a mutant m can be computed considering different SBFL formulas. However, some of the coverage information variables need to be replaced by mutant information variables. This transformation can be performed considering four main metrics: $a_{np} = |T_n \cap T_p$—, $a_{kp} = |T_k \cap T_p$—, $a_{nf} = |T_n \cap T_f$— and $a_{kf} = |T_k \cap T_f$—. These metrics also satisfy: $a_{np} + a_{kp} = totalpassed = |T_p|$ and $a_{nf} + a_{kf} = totalfailed = |T_f|$.

Equation 1 presents the application of four metrics in Ochiai formula. This application can also be performed in other formulas listed in the Table 1 after that transformation. The next step consists of computing $S(m)$ for each mutant $m \in M(s)$.

$$S(m) = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf}).(a_{kf} + a_{kp})}} \quad (1)$$

The method to aggregate the mutant value to suspiciousness of a statement s is given by the maximum value (Agg. Max): $S(s) = Max(S(m_1), ..., S(m_n))$, where $m_1, \ldots, m_n$ are all mutants in $M(s)$ and the value $S(s)$ is the probability of a statement s be faulty.

Creation of statement ranking: all statements are sorted in descending order by the value $S(s)$. After that, the order of the statements in the rank will guide the manual verification of the fault location and later its repair.

2.3 Variations of MBFL Techniques

首创的

Metallaxis (PAPADAKIS; TRAON, 2012) is the pioneering MBFL technique. For this reason, we consider the mechanism of killing mutants adopted by Metallaxis technique to represent the traditional MBFL.

Subsequently, other techniques have emerged that combines the SBFL and MBFL approaches to enhance fault localization activity. These techniques are called hybrid techniques and we highlight the following: MRSBFL-hybrid-max (Mutant-resolution Spectrum-based FL) and MCBFL-hybrid-avg (Mutant and Coverage-based FL). The advantage of hybrid techniques over traditional MBFL techniques is that they can assign suspiciousness value to non-mutable statements.

In general terms, there are two aspects that differentiate MBFL techniques: i) condition for a test case to kill a mutant and; ii) suspicious formula.

2.3.1 Condition for a test case to kill a mutant

In the case of the Metallaxis and MCBFL-hybrid-avg approaches, a test case kills a mutant m when there is a difference between the mutant and faulty program outputs. Such a difference is detected by failure at a different point or by a different error message.

没有oracle时，MCBFL的应用

In the case of the MRSBFL-hybrid-max approach, the test cases are not executed, only the coverage information of the mutants are considered. Thus, a test case kills a mutant m when it *cover* its mutation line in the original program P that has already had its coverage mapped by a test case set run. Therefore, the cost of the MRSBFL-hybrid-max technique is comparable to the cost of the SBFL techniques, and is much less than the cost of the other MBFL techniques that run the mutants.

2.3.2 Suspiciousness formula:

Table 1 shows the suspiciousness formulas of the techniques.

Metallaxis considers as $failed(m)$ the number of failed test cases that differentiate the outputs between the program P and its mutant m, that is, $failed(m) = akf$. The $totalfailed$ element is the size of the set T_f. The elements $passed(m)$ and $totalpassed$ are defined similarly. Then, Metallaxis assigns to the statement s the highest suspiciousness value between $M(s)$ applying the formula Ochiai adapted to MBFL (Equation 1).

3 The Fault Localization Method

The use of a spectrum-based fault localization (SBFL) technique requires: i) a faulty program P; ii) the test set T for P and; iii) the chosen SBFL technique. All test cases of T run over P. The coverage information is collected and the suspiciousness value of each statement is calculated as SBFL's formula. Finally, the statements ranking is generated in decreasing order that guides the programmer to find and repair the fault.

The ranking's accuracy has a great importance because real programs are big, having thousands of source code lines. Furthermore, the programs' statements have many dependencies. These two factors contribute to "zigzag effect" (PARNIN; ORSO, 2011): the programmers uses the rank non-sequentially. In practice, the first positions guide the identification process to repair the fault and the other positions are verified by relationship of the first positions' statements.

In the fault localization area, there are many techniques that claim to be better than others (WONG et al., 2016) for a variety of scenarios. The suspicious calculation also considers many SBFL formulas' combination. The Mutation-based Fault Localization techniques (MBFL) are more accurate than SBFL, but for some problems the SBFL ranks the faulty statements with highest accuracy at lower cost than MBFL techniques (PEARSON et al., 2017).

Facing this, we purpose the FTMES@r method that provides a smaller and more accurate SBFL ranking improved by MBFL technique. The purposed approach has two main steps: i) the statements filter and; ii) the mutants execution. The Figure 2 shows the general vision of our approach.

3.1 Statements Filter

Firstly, the programmer gives the following inputs: i) the faulty program P; ii) the test set T for P; iii) the first top-r rank's statements (@r) that will be evaluated; iv) the SBFL and; v) MBFL techniques.

The @r value is the size of SBFL's Ranking that the programmer determines to inspect after the FTMES@r execution. The set T runs over P and the coverage information is collected. Next, the SBFL technique performs the fault localization using the coverage information. The output is the entire Ranking of P statements in decreasing order.

After this, a new and smaller rank *Ranking@r* is filtered considering the @r value given by the programmer. The Mutation Analysis (MA) tool generates a set of mutants M from *Ranking@r* statements. We called this step as SFilter@r due the statements selection based in the SBFL accuracy (SFilter) and the size of ranking (@r) given by the programmer.

The SFilter@r and SBFL's costs are equivalent. The @r value helps the programmer to focus efforts on a ranking size according to the domain of the problem and in its reality to perform the activity of fault localization. The @r value filters the highest suspicious statements depending of SBFL

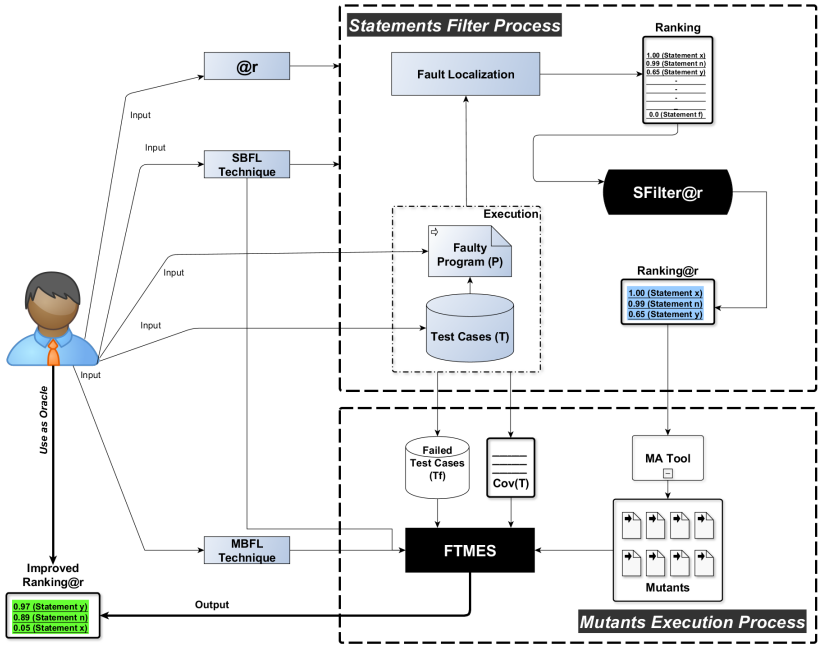

Fig. 1: The FTMES@r Framework.

accuracy but also contributes to the mutants set reduction considering the mutants generation based Ranking@r. Consequently, the MBFL technique is performed more quickly due to such reduction. The next Section describes the mutants execution step.

3.2 Mutants Execution

The FTMES@r's mutants execution step considers: i) the Mutants set M generated from the statements selected by SFilter@r; ii) the set of failed test cases T_f classified in previous tests execution over P; iii) the coverage information $cov(T)$ of failed and passed test cases; iv) the SBFL and; v) the MBFL techniques. The FTMES component is the mutation execution mechanism responsible to run, calculate the suspiciousness of statements and provides the improved *Ranking@r* to the programmer.

The FTMES and SFilter@r are two FTMES@r's components, but also works independently. The SFilter@r generates mutants to other mutation execution strategy. Similarly, the FTMES runs test cases over generated mutants from entire statements of a faulty program P covered by failed test cases set (T_f). Such techniques incorporate our approach because the following reasons:

i) the SBFL techniques have high accuracy for the most real problems ii) the FTMES is the current best MBFL's mutation execution strategy that runs mutants at low cost maintaining the localization efficacy and; iii) the union of this two techniques can turn the fault localization more useful and with a high fault localization power.

The Section 3.3 describe more details of FTMES component, as a general mutation execution strategy.

3.3 The Failed-Test-Oriented Mutant Execution Strategy (FTMES)

Failed-Test-Oriented Mutant Execution Strategy (FTMES) aims at obtaining efficacy of ranked lists in a more efficient way. FTMES uses only the set of failed tests to generate and execute mutants and avoids the execution of passed tests replacing their killing information with their coverage data.

In the process of fault localization, T executes P, where P is a Program Under Test (PUT) and T its set of test. Thus, the coverage information and test results (FAIL or PASS) are collected. Next, T is split in two sets: T_p and T_f, where T_p are the passed test cases and T_f are the failed test cases. Then, $cov(T_f)$ is the set of covered statements by T_f. The $cov(T_f)$ is used generate mutants for MBFL techniques. In most cases, generated mutant programs are a large set (M). Depending on the size of test suite, the mutant execution could be impractical.

The insight behind of FTMES approach is related to the test suit execution. The entire running of the failed test set (T_f) over the mutants programs set (M) seems to be necessary and justified by the negative behavior capture from the T_f. However, the execution of passed test cases (T_p) over the mutants (M) may not provide a significant contribution for localization the fault, since they do not capture any other helpful information of the fault and we already have coverage information from the original program. Further, we consider that $|T_p|$ is significantly larger than $|T_f|$, in practice. Therefore, we generate the mutants from the statements covered by T_f.

The Figure 2 presents the FTMES' workflow and the main steps are highlighted. The whole workflow has to be repeated for each statement s in the faulty program. In the end, the FL mechanism ranks the statements based on their suspiciousness. In the process of FTMES suspiciousness calculation of a statement s, we highlight two steps: i) calculate ak_f and an_f and; ii) calculate $aCov_p$ and $anCov_p$.

In the third step of Figure 2, T_f executes the $M(s)$ mutants by measuring both ak_f and an_f metrics, as stated in Section 2.2. The fourth step of Figure 2 does not run mutants, it only uses coverage information from the previous runs of T against P. In this step, the ***coverage information replaces killing information*** in order to avoid run the mutants. Thus, $ak_p = aCov_p$, where $aCov_p$ is the number of test cases in T_p that cover a mutant $m \in M(s)$. Similarly, $an_p = anCov_p$, where $anCov_p$ is the number of test cases in T_p that do not cover a mutant $m \in M(s)$. FTMES returns an SL list of suspiciousness

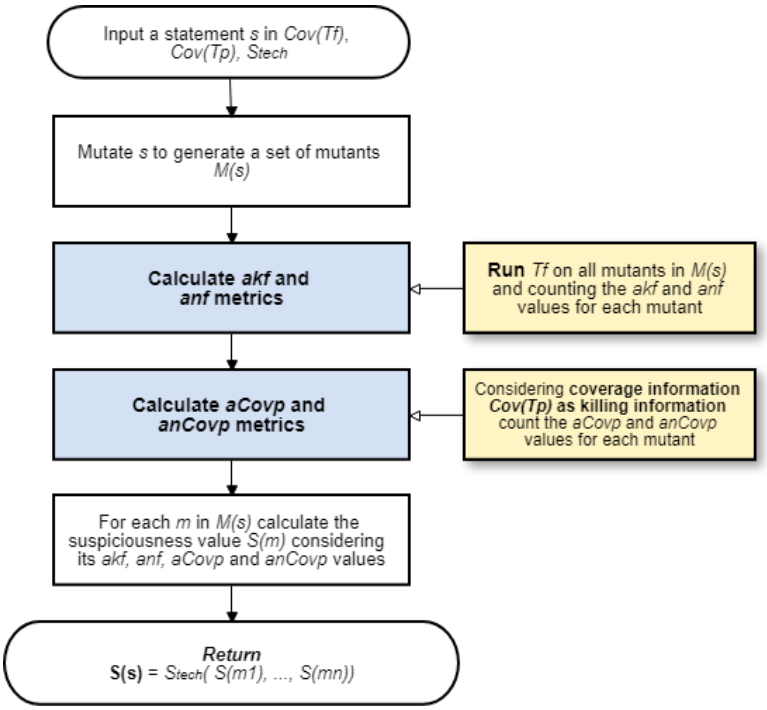

Fig. 2: The FTMES workflow

of the mutants related to statement s, according to the last step expressed in Figure 2. This list is used by MBFL techniques for assigning the final $S(s)$ value of suspiciousness to a given statement s.

The Algorithm 1 describes FTMES suspiciousness calculation process for a given statement s. The entries of Algorithm 1 include program P, test suite T, statement s covered by failed test cases, coverage information C of s, test result R, suspicious formula S_{form} and the S_{Tech} is the MBFL technique mechanism to assigns the $S(s)$ considering the SL (list of suspiciousness of each mutant $M(s)$ related to the statement s. C is a vector that every element $C(t)$ has two possible values, Covered and Uncovered, which indicates whether test t covers statement s or not. R is also a vector, with passed or failed values. $R(t) =$ Passed represents test case t passing on P, otherwise $R(t) =$ Failed represents t failing on P.

算法实体都有些什么

再描述算法的过程

Next, we explain the algorithm in detail. Line 1 mutates statement s to get the set of mutants $M(s)$ by mutation operators. Line 2 divides T into two sets: T_f is the set of failed test cases and T_p is the set of passed test cases. Lines 4-7 iterate over the $M(s)$ to get ak_f and an_f values for each mutant in $M(s)$ (step 3 of FTMES workflow, as Figure 2). Similarly, the lines 8-10 iterate over the M(s) to get $aCov_p$ and $anCov_p$ values (step 4 of FTMES framework, as Figure 2). The line 11 calculates the suspiciousness values for each mutant

Algorithm 1: Algorithm of FTMES Approach

Input: PUT P; test suite T; test result R; statement s covered by failed test cases; coverage data C of s; the MBFL suspicious formula S_{form}, and S_{Tech} as the MBFL mechanism to assigns the suspiciousness value to s

Output: *Susp(s)*

1: $M(s) \leftarrow Mutate(s)$
2: $Tf, Tp \leftarrow$ Partition T using R
3: **for** $m \in M(s)$ **do**
4: **for** $(t_f \in T_f)\&\&(C(t_f) == Covered)$ **do**
5: $Execute < m, t_f >$
6: Count ak_f and an_f
7: **end for**
8: **for** $t_p \in T_p\&\&C(t_p) == Covered$ **do**
9: Count $aCov_p$ and $anCov_p$
10: **end for**
11: $SL_m = S_{form}(ak_f, an_f, aCov_p, anCov_p)$
12: **end for**
13: **return** $Susp(s) = S_{Tech}(SL_M(s))$

m as S_{form} storing in SL list. Finally, the $Susp(s)$ is returned to assign the suspiciousness statement s, as S_{Tech} guides.

In this work, the S_{Tech} can be MBFL or MCBFL-hybrid-avg. For MBFL, FTMES signs the value of the statement s with the maximum value in SL, that is, $S_{MBFL}(s) = max(SL)$. For the hybrid MCBFL-hybrid-avg technique, FTMES uses the average: $S_{MCBFL-hybrid-avg}(s) = avg(max(SL), S_{SBFL}(s))$, as detailed in (PEARSON et al., 2017).

The FTMES does not add computational cost to the traditional MBFL implementation unlike the DMES baseline (GONG; ZHAO; LI, 2015). In terms of MTP values, utilizing the working example explained in Section II-B in work (GONG; ZHAO; LI, 2015), FTMES assigns *S(s)* with $MTP = 14$ (2 failed test cases against 7 mutants). In this same example, the DMES mutation execution strategy assigns *S(s)* with $MTP = 17$. Thus, our approach reduces the quantity of mutant execution compared to the baseline.

The effect of the FTMES approach is evaluated considering eight MBFL techniques, including a comparison with DMES baseline (GONG; ZHAO; LI, 2015).

4 Experimental Setup

4.1 Coverage and Mutation Framework Tools

The Major mutation framework tool (JUST, 2014) provides the MA data to techniques evaluation. Major generates mutants considering all available mutation operators. GZoltar tool is frequently used in FL context (CAMPOS et al., 2012). This tool provides the coverage data needed by our research.

4.2 Subject Programs

Defects4J is a database and extensible framework providing real bugs to enable reproducible studies in software testing research (JUST; JALALI; ERNST, 2014). We used Defects4J because it is a large, peer-reviewed and structured dataset of real-world Java bugs; it is considered the largest open database of well-organized real-world Java bugs (CAMPOS; MAIA, 2017). Our experiments consider the programs in the Defects4J dataset (v1.1.0): JFreeChart (26 faults), Apache Commons Lang (65 faults), Apache Commons Math (106 faults), Mockito (38 faults), and Joda-Time(27 faults). Defects4J also provides faulty and fixed program versions with a minimized change that represents the isolated bug fix.

为什么没有用Closure

The column 2 of Table 2 is the *Number of Faulty Versions*. This column has two numbers: the total provide by Defects4J and the total considered in the experiments. We were unable to obtain the following information from some program versions: (i) identification of defective lines; ii) identification of candidate lines for the correction of faults of omission and; iii) versions with only failed test cases. Hence we exclude these versions.

不明白

Table 2: Subject Programs.

Subject Program	Number of Faulty Version	KLOC	Average of $\|T_f\|$	Average of $\|T_p\|$	Average Number of Mutants
Chart	26 (24)	50	3.8	235.9	5284
Lang	65 (49)	6	1.8	169.8	2153
Math	106 (93)	19	3.5	186.0	9664
Mockito	38 (29)	22	3.9	661.5	2353
Time	27 (26)	53	3.5	2541.4	11031

4.3 Evaluation Metrics

In this paper, we consider *efficiency* as the capacity of the technique to reduce the amount of execution of mutants against test cases. Similarly, we consider

efficacy as the technique capacity to rank the faulty statements in the initial positions of the suspicious program statements list.

4.3.1 Metrics of Efficiency

An MTP value counts the number of mutant runs for test cases. This metric has been used to measure the cost reduction techniques for MBFL(LIU et al., 2018; GONG; ZHAO; LI, 2015; LIU et al., 2017). The idea of the metric is that the number of executions of mutants are linked to the computational cost required to obtain the rank of statements. That is, the lower MTP value of a MBFL technique, the greater its efficiency. We use this metric to evaluate the techniques that run mutants.

In addition, we also collect the runtime of the test cases. This value is the total time for execution of test cases on the faulty programs and mutants. The execution time does not consider compilation values or additional time for the use of a specific technique. It is worth noting that FTMES approach does not add additional time in the localization process, differently of DMES (GONG; ZHAO; LI, 2015).

4.3.2 Metrics of Efficacy

The goal of an FL strategy is the ranking of statements such that the element with the highest chance of being faulty appears in the first position. The need for a highly effective ranking encourages the development of automatic FL techniques, even though they are more computationally costly, in order to reduce the later manual effort.

Considering the importance of the efficacy of ranked list, the techniques studied were evaluated taking 3 metrics: i) *EXAM score* (WONG et al., 2008); ii) Accuracy (acc@n) and; iii) *wef* (Wasted Effort) (SOHN; YOO, 2017).

The *EXAM score* (called only as *score*) is the most popular metric to evaluate the efficacy of ranked list of the FL techniques (PEARSON et al., 2017; WONG et al., 2016). The EXAM score is n/N, where n denotes the number of statements that need to be inspected before finding the faulty statement of Program Under Test (PUT) and N is the number of statements in the program. The score ranges between 0 and 1, and *smaller* numbers are better. Our study evaluates the fault localization technique in the best case debugging scenarios: any faulty statement needs to be localized to understand and repair the fault. It multiple statements have the same suspiciousness score, we treat all of them as being the *nth* element in the output, where n is their average rank (PEARSON et al., 2017; STEIMANN; FRENKEL; ABREU, 2013; WONG et al., 2016).

The metrics *acc@n* and *wef* are based on the idea of counting program elements, instead of presenting percentage values, following the guideline in the literature (PARNIN; ORSO, 2011). The *acc@n* counts the number of faults that have been localized within *top n places* of the ranking (SOHN; YOO, 2017). We use 1, 3, 5, 10 and 20 for number n to evaluate the techniques and

count the number of corresponding faults per project and also overall. *Larger acc@n* values are better. The *wef* measures the amount of wasted effort looking at non-faulty program elements. Essentially, *wef* can be interpreted as the absolute count version of traditional Expense metric (JONES; HARROLD, 2005). *Smaller wef* values are better.

4.4 Techniques of Investigation

The proposed approach deals specifically with classes of MBFL and Hybrid techniques. However, the experimentation also presents results for the class of SBFL techniques as a baseline for MBFL techniques. Thus, our experimentation encompasses 3 classes of FL techniques.

Table 3 presents the techniques' configuration. The first column of Table 3 shows the *short name* of each technique to differentiate the subjects of investigation. The mutation execution strategies used for each are in second column and the classes of techniques are in third column.

The lines 1 and 2 of Table 3 corresponds to SBFL (B1) and MRSBFL-hybrid-max (B2) techniques. These techniques does not run mutants. Hence, the mutation execution strategies does not apply to them, and their computational cost corresponds to one execution of the entire test set over the faulty program. The subsequent lines of Table 3 show the MBFL techniques that run mutants.

Table 3: Fault Localization Techniques.

Short Name	Mutation Exec Strategy	Technique
B1	-	***SBFL*** (Dstar using variable* = 2)
B2	-	***MRSBFL-hybrid-max***
B3	SAM	***MBFL*** (killing a mutant like Metallaxis)
B4	SAM	***MCBFL-hybrid-avg***
B5	DMES	***MBFL***
B6	DMES	***MCBFL-hybrid-avg***
B7	DMES_SAM	***MBFL***
B8	DMES_SAM	***MCBFL-hybrid-avg***
B9	FTMES	***MBFL***
B10	FTMES	***MCBFL-hybrid-avg***

The SBFL (B1) technique is Dstar (WONG et al., 2014). It has been reported as the best SBFL technique, and was validated using real faults (PEARSON et al., 2017; PEARSON, 2016). For this reason, it is in all presented experiments, including the MBFL and Hybrid techniques. The MRSBFL-hybrid-max (B2) is shown in the next row. The MBFL (B3) technique considers the

mutant killing mechanism of the Metallaxis, as detailed in Section 2.3. The difference with the conventional Metallaxis is that we consider Dstar as suspiciousness calculation formula. Both (B3 and B4), skip runs according to *Skipping Alive Mutants (SAM)* strategy. It is worth mentioning that B3 and B4 techniques run the same amount of test cases presenting the same cost in terms of MTP and test cases' runtime.

The *dynamic mutation execution strategy (DMES)* (GONG; ZHAO; LI, 2015) operates on two distinct MBFL techniques: MBFL and MCBFL-hybrid-avg. When this strategy is canonically applied in conjunction with MBFL and MCBFL-Hybrid-avg, we refer to these combinations as B5 and B6, respectively. The strategy named as DMES_SAM is a small variation that we produced on canonical DMES (GONG; ZHAO; LI, 2015). We simply add the SAM strategy to the DMES mechanism. The idea of this combination is to compare the quality of the solutions between the strategies: DMES, SAM, and DMES_SAM. In Table 3, B7 and B8 correspond to the DMES_SAM application over MBFL and MCBFL-hybrid-avg, respectively. It is worth mentioning that DMES and DMES_SAM skip runs dynamically. Hence, DMES and DMES_SAM produce different amount of test case executions between MBFL and MCBFL-hybrid-avg.

Finally, The FTMES approach is also applicable to the MBFL and MCBFL-hybrid-avg techniques tabulated as B9 and B10 in Table 3, respectively. The FTMES' amount of test case executions are equivalent to MBFL (B9) and MCBFL-hybrid-avg (B10), because the same test set (T_f) is run over their respective covered mutants.

The mutation execution strategies aim to reduce the computational costs for MBFL techniques. The MBFL (B3) and MCBFL-hybrid-avg (B4) as the main techniques to be optimized by the studied mutation execution strategies: i) DMES; ii) DMES_SAM and; FTMES. Thus, it is expected that mutation execution strategies application makes the MBFL and MCBFL-hybrid-avg more efficient without loss in fault localization efficacy. Further, we show that for SBFL (B1) the MBFL techniques are actually more efficacious than SBFL techniques.

4.5 Research Questions

In order to evaluate the quality of FTMES solutions and other FL techniques, we considered the following research questions:

(RQ1) Is FTMES more efficient than the current mutation execution strategies?

(RQ2) Is FTMES really as effective as other techniques while being more efficient?

(RQ3) Does FTMES@r improve the fault localization activity producing:
- **(RQ3.1)** - a lower computational cost for MBFL techniques?
- **(RQ3.2)** - a highest efficacy of fault localization for SBFL's Ranking?

(RQ3.3) - a better cost-benefit relationship for MBFL techniques compared to SBFL's performance?

RQ1 and RQ3.1 use the Mutant Test Pair (MTP) value of each technique for efficiency evaluation. This metric has been used by other researchers to measure the cost of their MBFL techniques (GONG; ZHAO; LI, 2015; LIU et al., 2017, 2018).

To answer RQ2 and RQ3.2, we use the *EXAM Score*: the metric most used in the literature in the comparison of different techniques (WONG et al., 2008), as detailed in Section 4.3. Following the recent literature (PARNIN; ORSO, 2011), the efficacy of the solutions are also measured by the metrics *acc@n* and *wef*, also detailed in Section 4.3. Finally, we answered RQ3.3 with both metrics (efficiency and efficacy) performing a Pareto Frontier analysis.

5 Evaluation

5.1 Analysis of RQ1

RQ1: Is FTMES more efficient than the current mutation execution strategies? To answer the **RQ1**, we used the test cases' runtime and the *Mutant-Test Pair (MTP)* values of each technique. Figure 3 shows the efficiency of each technique in terms of *Mutant-Test Pair (MTP)*. The techniques' bars are ordered from lower to higher values in each subject program of Figure 3. For all subject programs, our approach was more efficient than all MBFL techniques. Regarding other mutation execution strategies, we highlight that the DMES_SAM variation performed better than canonical DMES.

The Table 4 gives the average of techniques' runtime for each program in minutes. For example, in the *Chart* program, the SBFL (B1) and MRSBFL-hybrid-max (B2) techniques spent 0.79 minutes on average. The SBFL (B1) and MRSBFL-hybrid-max (B2) approaches have the shortest runtime because they do not execute mutant programs. Among the mutation execution strategies, FTMES presents the shortest execution time (B9 and B10). Next, DMES_SAM and DMES strategy presents the second and third least time, respectively. The most costly strategies SAM.

FTMES and DMES are two mutation execution strategies. SAM is an optimization performed in previous researches (PAPADAKIS; TRAON, 2012; PEARSON et al., 2017). In this way, SAM is present as the canonical mechanism of baseline techniques: MBFL (B3) and MCBFL-hybrid-avg (B4). So, MBFL (B3) and MCBFL-hybrid-avg (B4) are considered baselines to evaluate the reduction percentage that FTMES and DMES can produce in the MBFL and MCBFL-hybrid-avg approaches. Table 5 presents the reduction percentage for the FTMES and DMES combination.

A Tabela 5 apresenta as reduções de custo realizadas pelas Strategys de execução de mutantes em relação às Techniques MBFL convencionais. Em outras palavras, a Tabela 5 descreve a redução produzida pela aplicação de

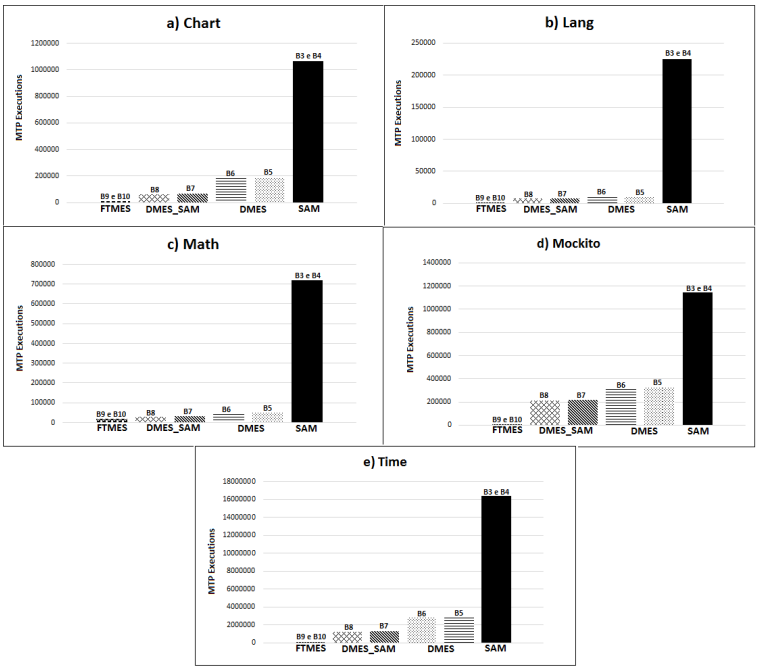

Fig. 3: Comparison of Techniques' *Mutant-Test Pair (MTP)*.

Table 4: Runtime of Test Case (minutes).

	Mutation Execution Strategies									
	SAM		*DMES*		*DMES_SAM*		*FTMES*			
	SBFL	MRSBFL*	MBFL	MCBFL*	MBFL	MCBFL*	MBFL	MCBFL*	MBFL	MCBFL*
Programs	(B1)	(B2)	(B3)	(B4)	(B5)	(B6)	(B7)	(B8)	(B9)	(B10)
Chart	0.79		801.43		141.79	140.91	51.50	48.93	8.91	
Lang	0.34		82.04		4.18	4.15	3.09	3.00	0.52	
Math	2.52		1034.47		71.94	70.77	46.67	42.40	27.23	
Mockito	4.04		2958.89		839.19	839.25	567.78	551.30	13.00	
Time	9.79		11303.18		1995.08	1950.85	907.57	858.34	16.98	
Average	**3.50**		**3233.00**		**610.44**	**601.19**	**315.32**	**300.79**	**13.33**	

DMES, DMES_SAM e FTMES nas Techniques MBFL (B3) e MCBFL-hybrid-avg (B4), em termos de execuções MTP. Por exemplo, o programa *Chart* (linha 3), a Strategy FTMES (coluna 4) reduziu 88% da quantidade de execuções MTP comparada às Techniques MBFL convencionais (B3 e B4).

According to Table 5, the canonical DMES was less efficient than its variation DMES_SAM. On average, the DMES strategy applied to MBFL (B5) produced 85,02% of average reduction while DMES with MCBFL-hybrid-avg (B6) produced 85,15%. Similarly, the DMES_SAM strategy applied to MCBFL (B7) reduced 91,80% while its application to MCBFL-hybrid-avg (B8) reduced

Table 5: MTP Average Execution Ratios Reduced by Mutation Execution Strategies.

	DMES		*DMES_SAM*		*FTMES*
Programs	**(B5)**	**(B6)**	**(B7)**	**(B8)**	**(B10)**
Chart	82,39%	82,50%	93,67%	93,99%	98,99%
Lang	95,30%	95,33%	96,64%	96,74%	99,77%
Math	93,27%	93,39%	95,72%	96,14%	97,61%
Mockito	71,74%	71,73%	80,92%	81,48%	99,70%
Time	82,42%	82,81%	92,05%	92,49%	99,94%
Average Reduction	**85,02%**	**85,15%**	**91,80%**	**92,17%**	**99,20%**

92,17%. Therefore, as canonical DMES as DMES_SAM performed a great efficiency when combined to MCBFL-hybrid-avg technique.

Answer to RQ1: FTMES reduced 99,20% of MBFL computational cost. Such reduction is 14% higher than the current mutation execution strategy: DMES. Therefore, FTMES is the current best mutation execution strategy for Mutation-based Fault Localization techniques.

5.2 Analysis of RQ2

RQ2: Is FTMES really as effective as other techniques while being more efficient? To evaluate **RQ2**, we compared each technique by considering three metrics that evaluate the efficacy of ranked lists. The first column of Table 6 lists the various considered values of *EXAM Score (Score)*, and the other columns are the percentages of PUT whose *Score* is smaller than the corresponding value for each considered technique. Table 6 presents the performance of the techniques according to the EXAM Score, independently of the project. For the value of EXAM Score = 0.01, the technique B1 reported values less than or equal to 0.01 in 48% (0.489) of the different versions.

The MCBFL-hybrid-avg (B4) technique exhibits better localization efficacy in terms of EXAM Score, according to Table 6. Our approach, FTMES (B10) appears with the second best performance. The third and fourth ones are SBFL (B1) and MRSBFL-hybrid-max (B2) techniques, respectively.

Table 7 shows the statistical comparison between MBFL techniques. We compared FTMES with all studied MBFL techniques that run mutants aiming to verify if FTMES is statistically better than others. We used the EXAM Score metric because its distribution is normal. We performed a paired t-test and Cohen's d (effect size) between FTMES and the other techniques. Emphasis on whether our evaluation *agree* indicates p-value: **p¡0.01**, p¡0.05, *(p>0.05)*. Emphasis on Cohen's d indicates effect size: **large**, medium, (small), *(negligible)*. The column "95% CI" gives the confidence interval for the difference in meaning. The column "(b – eq – w)" gives the counts for: per defect, was the winner better, equal to, or worse compared to the loser, ignoring the magnitude of the difference.

Table 6: Techniques' Efficacy Comparison with *EXAM Score* metric.

			Mutation Execution Strategies							
			SAM		***DMES***		***DMES_SAM***		***FTMES***	
	SBFL	**MRSBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***	**MBFL**	**MCBFL***
Score <=	(B1)	(B2)	(B3)	(B4)	(B5)	(B6)	(B7)	(B8)	(B9)	(B10)
0,01	0,489	0,471	0,385	0,548	0,403	0,376	0,412	0,430	0,131	0,534
0,05	0,778	0,756	0,688	0,851	0,624	0,656	0,661	0,701	0,570	0,796
0,10	0,905	0,896	0,738	0,928	0,670	0,742	0,715	0,792	0,742	0,910
0,15	0,950	0,941	0,756	0,968	0,697	0,783	0,742	0,828	0,819	0,946
0,20	0,968	0,968	0,769	0,977	0,701	0,796	0,751	0,842	0,837	0,973
0,30	0,982	0,986	0,774	0,982	0,701	0,796	0,751	0,851	0,864	0,982
0,40	0,982	0,986	0,774	0,991	0,701	0,801	0,751	0,855	0,864	0,986
0,50	0,982	0,986	0,774	0,991	0,701	0,801	0,751	0,855	0,864	0,986
0,60	0,991	0,995	0,977	1	0,991	0,986	0,986	0,986	0,995	0,995
0,70	1	1	0,991	1	1	0,995	1	0,995	0,995	1
0,80	1	1	0,995	1	1	0,995	1	0,995	0,995	1
0,90	1	1	0,995	1	1	0,995	1	0,995	0,995	1
1	1	1	1	1	1	1	1	1	1	1

Table 7: Statistical MBFL techniques comparison.

Comparisons	**EXAM Score**			
Winner >loser	agree?	d (eff.size)	95% CI	(b - eq - w)
FTMES (B10) >MBFL (B3)	**yes**	0.624	[-0.136, -0.073]	(218-1-3)
FTMES (B10) >MCBFL* (B4)	*(no)*	*(0.065)*	[-0.009, 0.0189]	(85-37-99)
FTMES (B10) >DMES (B5)	**yes**	0.767	[-0.166, -0.101]	(112-13-96)
FTMES (B10) >DMES (B6)	**yes**	0.589	[-0.123, -0.064]	(133-1-87)
FTMES (B10) >DMES_SAM (B7)	**yes**	0.669	[-0.143, -0.080]	(110-12-99)
FTMES (B10) >DMES_SAM (B8)	**yes**	(0.470)	[-0.095, -0.041]	(122-2-97)

Except for MCBFL-hybrid-avg, FTMES (B10) is better than all other techniques with 99% of confidence level ($p \leq 0.001$) and with a *medium effect size*. MCBFL-hybrid-avg is better than FTMES, but **such result has low statistical significance ($p = 0.4902$) with a negligible effect size**. This result shows that FTMES improves the efficiency of MCBFL-hybrid-avg (B4) without significant loss of the efficacy to fault localization activity.

Table 8 shows the performance of the techniques in terms of *acc@n* and *wef* for each subject program providing a project dimension. The first column lists the short name of each studied technique. The second one shows the number of bugs considered. From the second to the seventh column are shown the n values for the metrics *acc@n*. Lastly, the two last columns show the average (wef_avg) and standard deviation (wef_std) for the metric *wef*, respectively. The n list of acc@n values are: 1, 3, 5, 10 and 20. The *wef_avg* and *wef_std* are the average and standard deviation of *wef*, respectively. The **first best** technique result has a **black background**. The **second best** technique result has a **gray background**.

The Table 8 shows the ***first best*** technique result has a cell with a ***black background*** following the respective parameters. Similarly, the ***second best***

Table 8: Techniques' Efficacy Comparison with *acc@n* and *wef*.

Technique	Prog.	Def.	acc@1	acc@3	acc@5	acc@10	acc@20	wef_med	wef_desv
Name:	Chart	24	0	2	3	7	12	**155,00**	551,75
SBFL	Lang	49	1	5	13	24	30	32,02	43,82
Strategy:	Math	93	6	22	27	34	43	**60,47**	**125,06**
without	Mockito	29	**4**	**9**	**13**	**13**	**15**	324,83	912,52
(B1)	Time	26	0	4	8	11	13	97,77	151,87
	Total	**221**	**11**	**42**	**64**	**89**	**113**	**670,09**	**1785,02**
Name:	Chart	24	0	2	3	7	12	155,00	551,75
*MRSBFL**	Lang	49	1	5	13	24	30	32,02	43,82
Strategy:	Math	93	6	21	26	33	42	107,54	264,91
without	Mockito	29	**4**	**9**	**12**	12	14	**234,83**	776,37
(B2)	Time	26	0	4	8	**11**	13	97,77	151,87
	Total	**221**	**11**	**41**	**62**	**87**	**111**	**627,15**	**1788,71**
Name:	Chart	24	**1**	4	7	9	12	976,04	2441,83
MBFL	Lang	49	5	12	19	23	30	218,53	504,91
Strategy:	Math	93	8	18	22	34	44	540,61	1286,40
without	Mockito	29	1	**5**	6	9	10	247,48	**610,64**
(B3)	Time	26	**6**	**7**	**10**	**13**	**16**	395,65	1124,89
	Total	**221**	**21**	**46**	**64**	**88**	**112**	**2378,32**	**5968,68**
Name:	Chart	24	**2**	**7**	**9**	**11**	**14**	**60,54**	**133,22**
*MCBFL**	Lang	49	7	**17**	**22**	**31**	**37**	**19,39**	**36,04**
Strategy:	Math	93	**10**	**27**	**35**	**44**	**53**	**53,53**	**134,84**
without	Mockito	29	**5**	**9**	9	**15**	**18**	186,83	771,41
(B4)	Time	26	**5**	**10**	**10**	**14**	**16**	**70,73**	**115,01**
	Total	**221**	**29**	**70**	**85**	**115**	**138**	**391,01**	**1190,52**
Name:	Chart	24	**1**	3	5	8	9	1166,46	2780,58
MBFL	Lang	49	**9**	12	15	20	27	346,86	626,07
Strategy:	Math	93	8	20	25	37	43	673,46	1472,27
DMES	Mockito	29	1	2	2	4	6	492,93	938,39
(B5)	Time	26	**5**	6	8	11	**17**	442,58	1296,42
	Total	**221**	**24**	**43**	**55**	**80**	**102**	**3122,29**	**7113,72**
Name:	Chart	24	**1**	3	6	8	10	1219,21	2729,04
*MCBFL**	Lang	49	**10**	**14**	15	23	32	233,02	556,31
Strategy:	Math	93	8	15	22	37	43	467,10	1317,12
DMES	Mockito	29	2	2	2	4	4	432,59	886,10
(B6)	Time	26	**5**	6	6	8	11	483,88	1232,15
	Total	**221**	26	**40**	**51**	**80**	**100**	**2835,80**	**6720,71**
Name:	Chart	24	**1**	**5**	7	9	11	1117,21	2792,23
MBFL	Lang	49	7	13	18	22	27	304,43	612,71
Strategy:	Math	93	7	19	21	34	43	592,61	1427,29
DMES_SAM	Mockito	29	0	3	3	6	8	462,66	938,26
(B7)	Time	26	**5**	**5**	8	10	17	448,69	1294,96
	Total	**221**	**20**	**45**	**57**	**81**	**106**	**2925,60**	**7065,45**
Name:	Chart	24	**1**	**5**	**8**	**10**	12	1064,38	2744,75
*MCBFL**	Lang	49	**9**	**17**	**20**	**25**	31	119,02	434,45
Strategy:	Math	93	7	15	18	36	45	440,62	1304,56
DMES_SAM	Mockito	29	0	3	3	7	8	435,31	900,07
(B8)	Time	26	**5**	5	8	10	15	435,12	1236,62
	Total	**221**	**22**	**45**	**57**	**88**	**111**	**2494,44**	**6620,46**
Name:	Chart	24	0	0	0	0	1	3625,75	5765,03
MBFL	Lang	49	0	0	0	0	0	756,55	624,14
Strategy:	Math	93	0	0	0	0	0	2051,02	1748,26
FTMES	Mockito	29	0	0	0	0	0	1401,69	**613,13**
(B9)	Time	26	0	0	0	0	0	4497,19	1528,61
	Total	**221**	**0**	**0**	**0**	**0**	**1**	**12332,20**	**10279,18**
Name:	Chart	24	0	**5**	7	**10**	**13**	**53,46**	**89,84**
*MCBFL**	Lang	49	1	5	16	**25**	**36**	**25,47**	**38,74**
Strategy:	Math	93	**12**	**27**	**30**	**40**	**50**	75,35	242,29
FTMES	Mockito	29	3	**9**	**12**	12	**15**	**210,79**	774,94
(B10)	Time	26	0	4	**9**	11	14	**74,92**	**107,29**
	Total	**221**	**16**	**50**	**74**	**98**	**128**	**440,00**	**1253,09**

technique result has a ***gray background***. This characteristic shows the MCBFL-hybrid-avg with and FTMES with the ***first*** and ***second best results*** among the subject programs, respectively.

Answer to RQ2: The FTMES' efficacy of localization is close to MCBFL-hybrid-avg, the best performance in terms of: i) EXAM Score; ii) Accuracy (acc@n) and; iii) Wasted Effort (wef). The statistical analysis reveals a negligible effect size of between the difference of FTMES and MCBFL-hybrid-avg techniques. Contrasting, DMES strategy was worse than SBFL's efficacy. Therefore, we consider that FTMES maintain the MFBL's efficacy of localization.

5.3 Analysis of RQ3

The research question RQ3 aims to evaluate the interaction between SFilter@r and FTMES considering the following dimensions: RQ3.1) MBFL's cost reduction; RQ3.2) increase of SBFL's efficacy of localization and; RQ3.3) the cost-benefit analysis. The selected techniques to these analyses were:

1. **SBFL**: represented by Dstar (WONG et al., 2012), as Section 2.
2. **MCBFL**: represented by MCBFL-hybrid-avg technique (B4 - Table 3). Such technique was selected to SFilter@r application because it is the MBFL technique with the highest efficacy of localization, as Section 5.2.
3. **FTMES**: represented by FTMES (B10 of Table 3). Such technique was selected to SFilter@r application because it is the best mutation execution strategy, as Sections 5.1 e 5.2.

5.3.1 MBFL's Cost Reduction

RQ3.1: Does FTMES@r improve the fault localization activity producing a lower computational cost for MBFL techniques? The goal is to evaluate the interaction between SFilter@r and the different MBFL techniques observing the produced reduction rates. The Table 9 presents the MCBFL and FTMES's computational costs considering different sizes of r to SFilter@r application. The techniques' runtime is in minutes and the general average is in hours.

FTMES has the highest computational cost comparing with all programs and considering different values of @r. Beyond, less values of @r produces less computational costs to FTMES and MCBFL techniques. The SFilter@r application reduces considerably the MBFL's runtime. Such reduction appear in comparison to different @r results with the column ***wihtout*** (MBFL's techniques results without SFilter@r use).

The Figure 4 presents the MTP reduction rates by SFilter@r application in the FTMES and MCBFL compared to traditional MFBL (without the mutation execution and SFilter@r application).

SFilter@r produces lower amounts of mutant executions when as FTMES@r guides the MCBFL technique, considering all values to @r. The Figure 4 also

Table 9: MBFL techniques' runtime with different sizes of @r in SFilter@r application.

Techniques	Prog.	@r 10	20	30	50	100	200	without
*MCBFL**	Chart	7,94m	15,23m	21,46m	39,42m	71,33m	136,7m	801,43m
	Lang	1,97m	3,54m	4,67m	6,51m	12,66m	26,27m	82,04m
	Math	16,65m	30,18m	43,88m	74,3m	115,26m	219,41m	1034,47m
	Mockito	210,83m	442,6m	640,8m	992,26m	1595,01m	2146,56m	2958,89m
	Time	123,91m	241,49m	361,64m	598,85m	1187,21m	2408m	11303,18m
	Average	**1,2h**	**2,44h**	**3,57h**	**5,7h**	**9,94h**	**16,46h**	**53,93h**
FTMES	Chart	0,89m	0,98m	1,05m	1,29m	1,67m	2,49m	8,91m
	Lang	0,35m	0,37m	0,38m	0,39m	0,44m	0,49m	0,52m
	Math	2,82m	3,18m	3,47m	4,12m	5m	7,75m	27,23m
	Mockito	4,76m	5,6m	6,27m	7,52m	9,47m	11,21m	13m
	Time	9,89m	9,99m	10,09m	10,3m	10,79m	11,73m	16,98m
	Average	**0,06h**	**0,07h**	**0,07h**	**0,08h**	**0,09h**	**0,11h**	**0,22h**

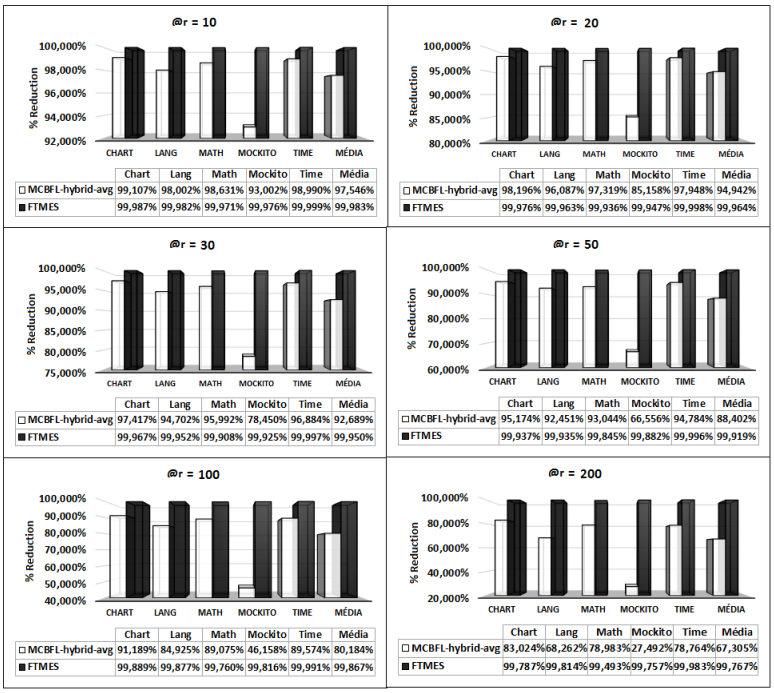

Fig. 4: A comparison of reduction rates of MTP executions with SFilter@r uses.

presents that values of @r less or equal than 30 (@$r \leq 30$) show reduction higher than 92% compared to traditional MBFL, considering the SFilter@r to FTMES and MCBFL.

Considering only the SFilter@r application to MCBFL, @r values between [50,100] produce a reduction of 80% on average. The reduction of 67% is the lower average to @$r = 200$. *Mockito* was the program with lower reduction rate compared to the reduction performed to the other programs. The SFilter@r applied with FTMES produced an average reduction always higher than 99% to any value of @r. Such result demonstrates the higher reduction of FTMES@r approach.

As described in evaluation of RQ1 (Section 5.1), the SBFL has a computational cost much lower than MBFL techniques. Thus, the reduction tecniques' goal is to become the MBFL cost close to SBFL class. The Figure 5 aims to show the SBFL's approximation rates that SFilter@r produces with FTMES and MCBFL techniques. How great the rate is great is the SBFL runtime approximation. In other words, the higher rate is the higher cost reduction produced in FTMES and MCBFL.

The Figure 5 represents that the higher SBFL cost approximation is produced by SFilter@r application to FTMES considering all studied programs. The approximation rates is higher than 50% when @$r \geq 50$ to all programs. On the other side, the SFilter@r application to MCBFL generates less approximation rates in *Mockito* and *Time* programs.

Answer to RQ3.1: FTMES@r produced the highest cost reduction to MBFL considering all scenarios. The interaction between SFilter@r and MCBFL also produced considerable reductions. However, the FTMES@r's reduction was significantly higher.

5.3.2 Increase of SBFL's Efficacy of Localization

RQ3.2: Does FTMES@r improve the fault localization activity producing a highest efficacy of fault localization for SBFL's Ranking? The goal of RQ3.2 is to evaluate the degree of improvement produced per SFilter@r interaction with different MBFL techniques when considering the SBFL's Ranking. According to the presented, SFilter@r reduce the MBFL cost. The Table 9 presents accuracy (acc@n) and wasted effort (wef) with SFilter@r application. Aiming a reliable comparison, we evaluate the SBFL's Ranking with FTMES and MCBFL when used and also when it doesn't use SFilter@r.

The first column of Table 10 are the @r values: $\{30, 50, 100, 200\}$. The second and third columns show the techniques and programs studied. The fourth column (bug) is the total bugs per program. We highlight that the amount of bugs depend of the @r values because some programs versions have a Ranking size lower than *@r* value. The version 25 of Chart program has $|Ranking| = 25$, i.e, the failed test cases covers only 25 program statements for this respective version. Thus, the value @$r = 30$ exclude such version *Chart*

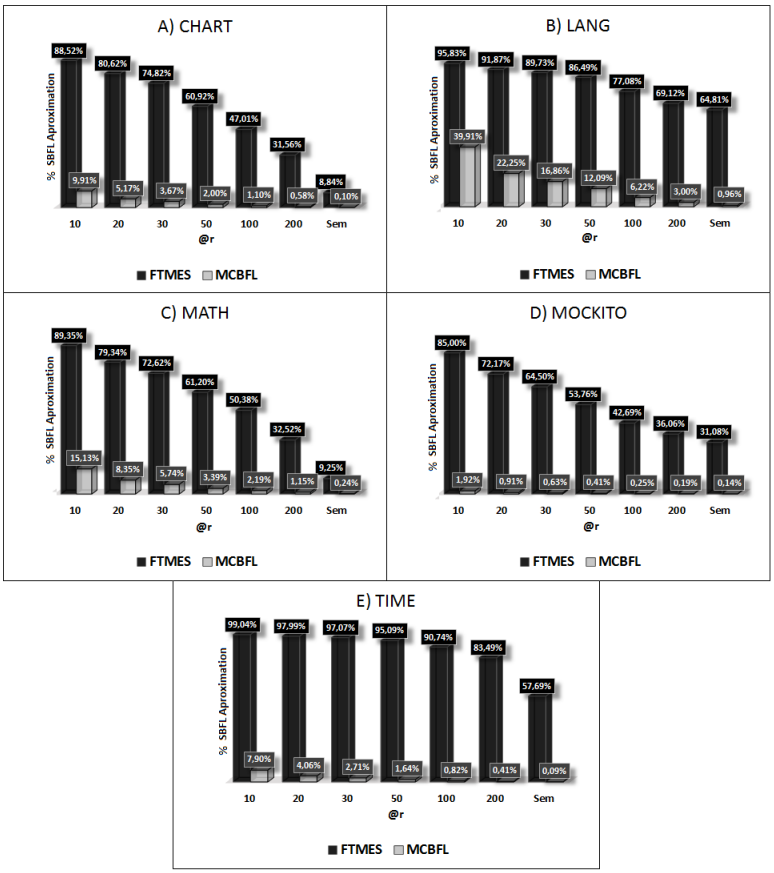

Fig. 5: Rates of Approximation to SBFL' Runtime.

and this bug is not considerate. Similarly, the values of $@r = 100$ and $@r = 200$ selected 211 and 198 bugs, respectively.

The columns from 5 to 9 of Table 10 present different values to n for accuracy calculation ($acc@n$). Starting from fourth to a sixth line of each experiment shows the techniques' performance, similarly the Table 8. For example, considering the experiment with $@r = 50$ of FTMES, was considered 220 bugs and the $acc@1 = 28/5\%$, i.e, 28 faults were located what is 5% of 220 bugs. The last column is the average of wasted effort (wef_avg).

In general, the SFilter@r increases the SBFL's accuracy ($acc@n$) with all $@r$ and n values. The improvement also happens in terms of wasted effort (wef_avg) presenting fewer values.

The values $n = 20$ and $@r = 50$ set the best accuracy produced by SFilter@r application with MCBFL: 136 located faults / 62% of 220. With the same value of n the best SFilter@r and FTMES interaction happened with $@r = 30$:

Table 10: SBFL's Efficacy Improvement (acc@n and wef) with MBFL techniques (FTMES, MCBFL and SFilter@r applications).

@r	Technique	Prog.	bugs	acc@1	acc@3	acc@5	acc@10	acc@20	wef_avg
30	**Name:**	Chart	23	0	2	2	6	11	92,70
	SBFL	Lang	49	1	5	13	24	30	17,14
	Strategy:	Math	93	6	22	27	34	43	40,17
	without	Mockito	29	4	9	13	13	15	193,45
		Time	26	0	4	8	11	13	61,00
	Total		**220**	**11/5%**	**42/19%**	**63/29%**	**88/40%**	**112/51%**	**404,46**
	Name:	Chart	23	2	7	8	10	12	89,30
	MCBFL-hybrid-avg	Lang	49	7	16	20	31	34	13,67
	Strategy:	Math	93	11	26	34	44	54	37,32
	without	Mockito	29	5	8	9	13	17	193,10
		Time	26	5	11	12	14	16	57,23
	Total		**220**	**30/14%**	**68/31%**	**83/38%**	**112/51%**	**133/60%**	**390,63**
	Name:	Chart	23	0	4	6	9	12	90,00
	MCBFL-hybrid-avg	Lang	49	1	5	16	26	36	15,35
	Strategy:	Math	93	12	28	30	40	52	37,48
	FTMES	Mockito	29	3	9	12	13	15	193,72
		Time	26	0	4	9	12	14	59,96
	Total		**220**	**16/7%**	**50/23%**	**73/33%**	**100/45%**	**129/59%**	**396,52**
50	**Name:**	Chart	23	0	2	2	6	11	127,04
	SBFL	Lang	49	1	5	13	24	30	23,96
	Strategy:	Math	93	6	22	27	34	43	44,02
	without	Mockito	29	4	9	13	13	15	233,93
		Time	26	0	4	8	11	13	93,42
	Total		**220**	**11/5%**	**42/19%**	**63/29%**	**88/40%**	**112/51%**	**522,38**
	Name:	Chart	23	2	7	8	10	12	123,22
	MCBFL-hybrid-avg	Lang	49	7	16	20	31	34	18,65
	Strategy:	Math	93	10	27	35	46	55	38,55
	without	Mockito	29	5	9	9	14	19	230,62
		Time	26	4	9	11	14	16	89,73
	Total		**220**	**28/13%**	**68/31%**	**83/38%**	**115/52%**	**136/62%**	**500,77**
	Name:	Chart	23	0	4	6	9	12	124,09
	MCBFL-hybrid-avg	Lang	49	1	5	16	25	35	21,39
	Strategy:	Math	93	12	28	31	41	50	39,83
	FTMES	Mockito	29	3	9	12	13	15	234,03
		Time	26	0	4	9	11	14	91,85
	Total		**220**	**16/7%**	**50/23%**	**74/34%**	**99/45%**	**126/57%**	**511,18**
100	**Name:**	Chart	23	0	2	2	6	11	208,57
	SBFL	Lang	41	0	4	10	17	22	33,32
	Strategy:	Math	92	6	22	27	34	43	50,87
	without	Mockito	29	4	9	13	13	15	240,79
		Time	26	0	4	8	11	13	160,65
	Total		**211**	**10/5%**	**41/19%**	**60/28%**	**81/38%**	**104/49%**	**694,20**
	Name:	Chart	23	2	7	8	10	13	203,87
	MCBFL-hybrid-avg	Lang	41	5	14	16	24	28	24,44
	Strategy:	Math	92	10	27	34	45	53	42,57
	without	Mockito	29	5	8	8	13	16	246,93
		Time	26	4	9	11	13	15	158,12
	Total		**211**	**26/12%**	**65/31%**	**77/36%**	**105/50%**	**125/59%**	**675,92**
	Name:	Chart	23	0	4	6	9	12	205,00
	MCBFL-hybrid-avg	Lang	41	0	4	12	18	27	28,78
	Strategy:	Math	92	12	28	31	41	49	45,04
	FTMES	Mockito	29	3	9	12	12	15	244,38
		Time	26	0	4	9	11	14	159,08
	Total		**211**	**15/7%**	**49/23%**	**70/33%**	**91/43%**	**117/55%**	**682,28**
200	**Name:**	Chart	21	0	2	2	6	9	146,76
	SBFL	Lang	39	0	4	10	17	21	37,36
	Strategy:	Math	84	4	18	22	28	35	61,06
	without	Mockito	29	4	9	13	13	15	213,28
		Time	25	0	3	7	10	12	184,20
	Total		**198**	**8/4%**	**36/18%**	**54/27%**	**74/37%**	**92/46%**	**642,66**
	Name:	Chart	21	1	6	7	9	11	139,33
	MCBFL-hybrid-avg	Lang	39	4	12	15	23	27	22,77
	Strategy:	Math	84	8	20	27	36	46	52,33
	without	Mockito	29	4	8	8	14	17	215,24
		Time	25	5	9	10	13	15	168,00
	Total		**198**	**22/11%**	**55/28%**	**67/34%**	**95/48%**	**116/59%**	**597,68**
	Name:	Chart	21	0	4	5	8	10	143,33
	MCBFL-hybrid-avg	Lang	39	0	4	12	18	26	29,44
	Strategy:	Math	84	10	23	26	34	42	53,32
	FTMES	Mockito	29	3	9	12	12	15	212,24
		Time	25	0	3	8	10	13	182,16
	Total		**198**	**13/7%**	**43/22%**	**63/32%**	**82/41%**	**106/54%**	**620,49**

129 located faults / 59% of 220. The Figure 6 shows the improvement rate that SFilter@r performed with MBFL techniques for SBFL's Ranking.

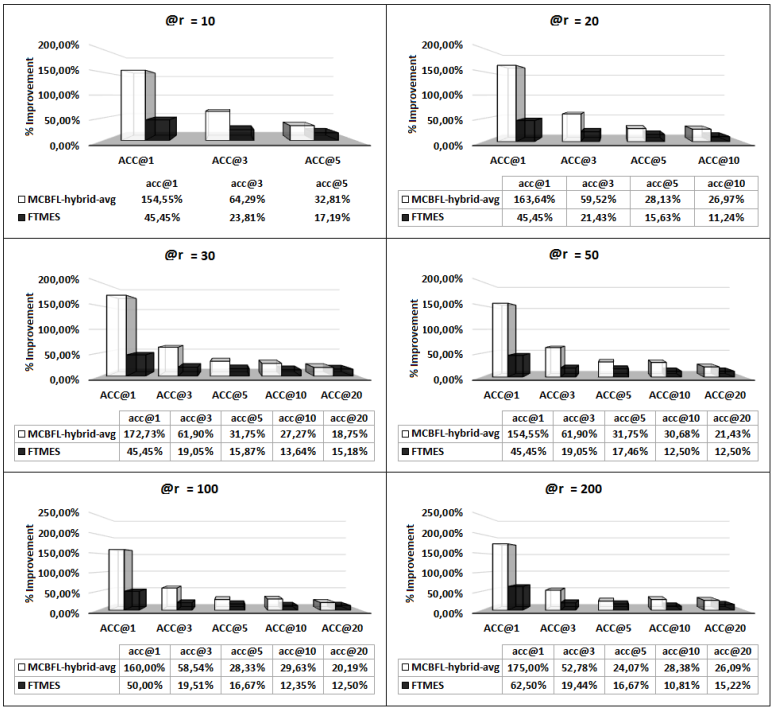

Fig. 6: Rates of SBFL's Ranking Accuracy Improvement performed by SFilter@r and MBFL Techniques Interaction.

The Figure 6 shows that the $@r = 200$ value produces greater rates of improvement. However, the highest amount of located faults was found considering the $@r = 30$ e $@r = 50$ values to interactions SFilter@r with FTMES and MCBFL, respectively. The SFilter@r with MCBFL produced highest accuracy in all experiments. Considering $n = 20$ such interaction produced 18% and 26% of improvement rates. The SFilter@r with FTMES produced 12% e 15% of improvement rates.

The Figure 7 shows general results of accuracy and wasted effort considering all programs as one. In terms of accuracy, it is visible the MCBFL and MCBFL@50 techniques ($SFilter$@50 and MCBFL interaction). Contrasting, the SBFL technique has the worst performance.

The FTMES@30 ($SFilter$@50 and FTMES interaction) and FTMES present results between MCBFL and SBFL accuracy performances. Considering the wasted effort (wef), the MCBFL and FTMES@30 show the best results. Observing accuracy ($acc@n$) and wasted effort (wef), the MCBFL presents the

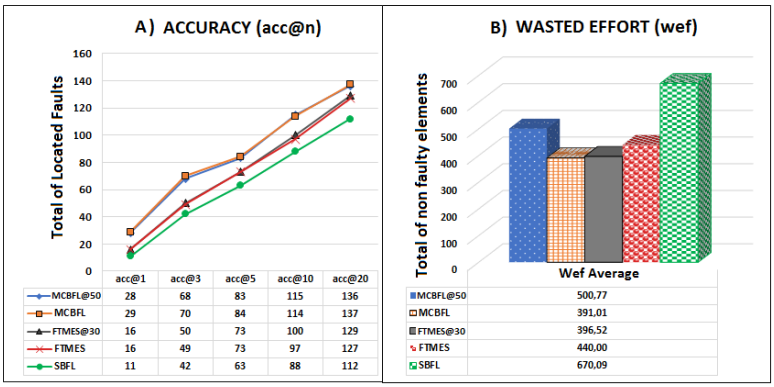

Fig. 7: General performance of Techniques Efficacy.

best results followed by the FTMES@30 technique. However, the FTMES@30 performance is close to MCBFL.

Answer to RQ3.2: FTMES@r improves the SBFL's Ranking producing higher efficacy of localization in all experiments. However, the SFilter@r and MCBFL-hybrid-avg interaction produced better values for *acc@n* and *wef*, despite of FTMES@r demonstrates very close results.

5.3.3 Cost-benefit Relationship

RQ3.3: Does FTMES@r improve the fault localization activity producing a better cost-benefit relationship for MBFL techniques compared to SBFL's performance? In some scenarios, the efficacy gain of a MBFL technique is so small that the computational cost aggregate turns its use implausible. Therefore, an MBFL requires the cost-benefit evaluation.

The analysis of Figure 8 suggests the Pareto Frontier with two objectives: runtime and accuracy (*acc@20*); similarly to multi-objective techniques comparison (YOO; HARMAN, 2010).

Considering the Figure 8, one technique dominates another when it has a lower runtime and a higher accuracy. Thus, the closer to zero of horizontal axis (runtime) and the higher it is in vertical axis (acurácia), the better is the tecnique. For example, the FTMES@30 dominates FTMES and MCBFL@50 technique in the *Chart* program. The idea is to visualize the dominance relationship based in different techniques' performance. Depending of the context, some objectives can have more importance than others and such vision subsidize the techniques selection.

In general terms, the SBFL technique is note dominates in terms of runtime. However, it is dominates to all other considering efficacy of localization. FTMES@30 dominates FTMES in most experiments. Furthermore, FTMES@30 dominates MCBFL@50 in two programs: *Chart* and *Lang*. This result strength-

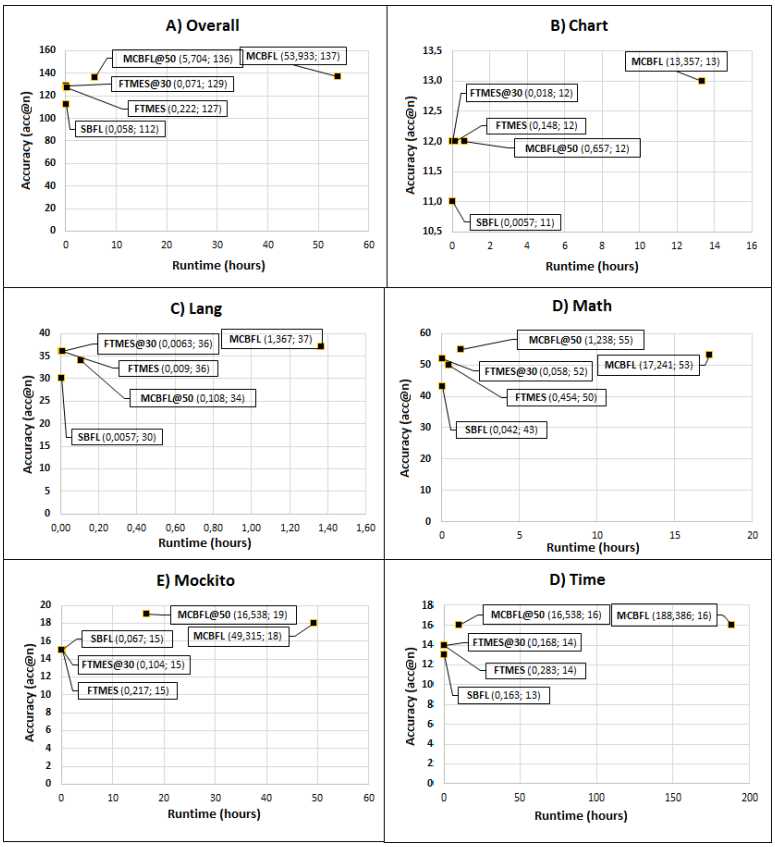

Fig. 8: Pareto Frotier: Runtime and Accuracy (*acc@20*).

ens the FTMES@r approach because FTMES@30 is 70% and 99% less costly than FTMES and MCBFL@50, respectively.

The same relationship of dominance is not perceived between MCBFL@50 and MCBFL. In general terms, MCBFL@50 dominates MCBFL in runtime. Considering accuracy, MCBFL@50 dominates MCBFL in *Math* and *Mockito* programs. MCBFL dominates MCBFL@50 in the *Chart* and *Lang* programs and a tie happens to *Time*. Considering the great computational cost of MCBFL orginal, the MCBFL@50 is a better choice in terms of runtime and accuracy.

The Figure 9 that suggests the same analysis is the same of Figure 8, but considering wasted effort. FTMES@30 dominates FTMES and MCBFL@50 in all programs. FTMES@30 also dominates the MCBFL original in three of five studied programs (*Lang*, *Math* e *Time*). Considering the runtime and wasted effort, FTMES@30 is a good choice considering the great runtime of MCBFL approach.

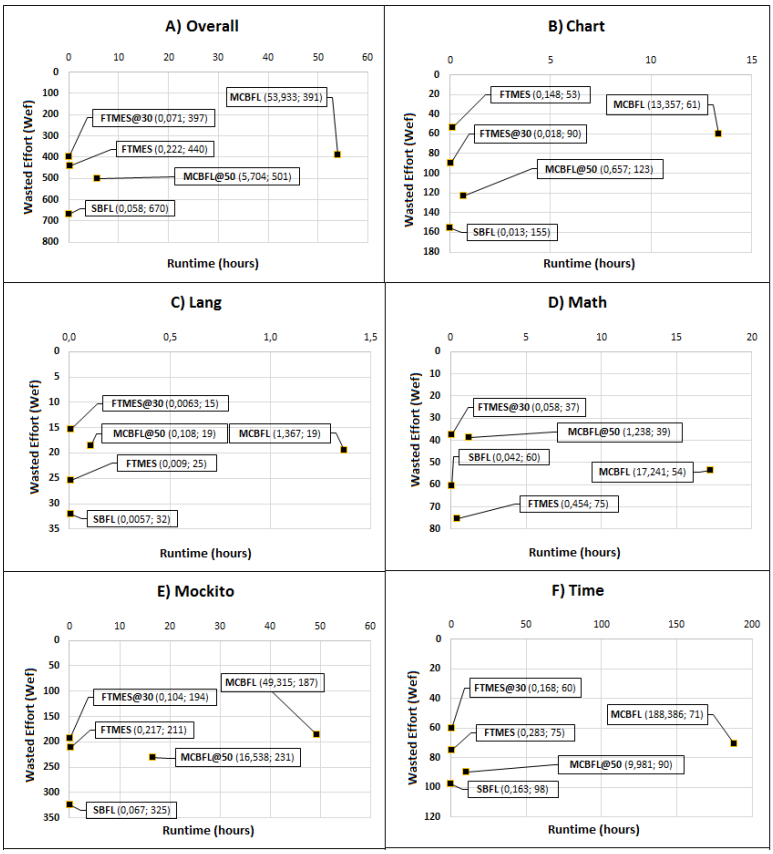

Fig. 9: Pareto Frontier: Runtime and Wasted Effort (wef).

FTMES@30 dominates with higher superiority all MBFL techniques when considering only runtime as goal. FTMES@30 also dominates MCBFL@50 in all studied programs considering two objectives: runtime and wasted effort. About Accuracy (acc@20), FTMES@30 dominates MCBFL@50 in two considered programs. However, FTMES@30 was dominated per MCBFL in terms of efficacy of localization with low difference rate.

The goal's priority determines the techniques choice when the dominance relationship does not happen over all considered objectives. In real scenarios, there are great contexts varieties, for example, the accuracy to be more important than wasted effort or time. Beyond benefits priorities, the time is considered with higher priority.

Considering the computational cost, there is the expectation that MBFL techniques overcome the SBFL in terms of efficacy. When the reduction strategy is used, another expectation is the approximation of runtime approximation and efficacy improvement of SBFL class. The Equation 2 and 3 aims to establish a percentage p to prioritize the MBFL choice. The value α is the

rate that prioritizes the cost approximation (runtime). The ω value is the rate that prioritizes the efficacy's improvement (acc@n or wef). Such Equations is useful for different scenarios. For example, when different weights are needed (ex.: $\alpha = 30\%$ e $\omega = 70\%$) or the same weights (ex.: $\alpha = 50\%$ e $\omega = 50\%$), considering accuracy and wasted effort.

$$p_{acc} = \alpha * runtime_{aprox} + \omega * acc@n_{improvement} \quad (2)$$

$$p_{wef} = \alpha * runtime_{aprox} + \omega * wef_{improvement} \quad (3)$$

The Figure 10 suggests the Pareto Frontier analysis of: i) the p_{acc} and ii) p_{wef} rates. The Figure shows the techniques performance considering different weights to: i) cost: $\alpha = 50\%, 30\%, 20\%, 10\%$) and ii) benefit: $\alpha = 50\%, 70\%, 80\%, 90\%$. In the charts, the greater are the percentages the better is the technique performance considering the two objectives.

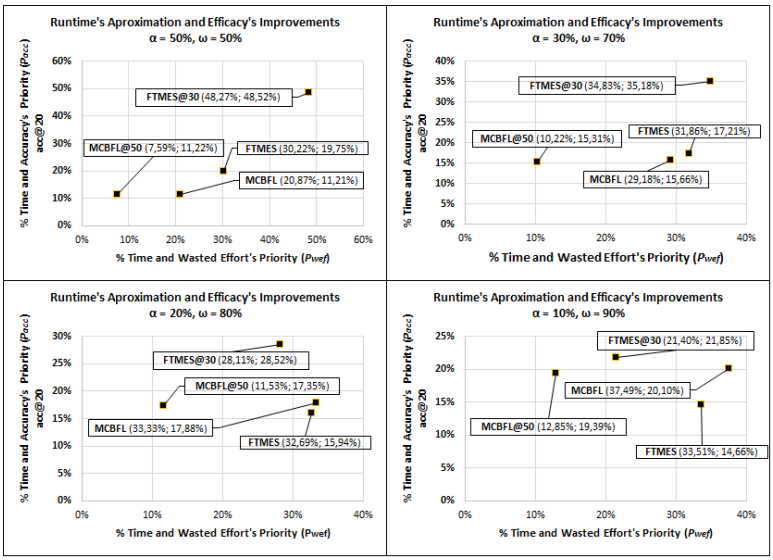

Fig. 10: Pareto Frontier to Prioritizes Cost-Benefit.

The FTMES@30 dominates all techniques with significant distance to scenarios with the same priority to cost and benefit: $\alpha = 50\%$ e $\omega = 50\%$. The FTMES component also dominates the MCBFL and MCBFL@50. Such conclusion is the same to $\alpha = 30\%$ e $\omega = 70\%$.

The FTMES@30 dominates only MCBFL@50. MCBFL dominates FTMES but does not dominate FTMES@30. Such conclusion is the same to $\alpha = 10\%$ e $\omega = 90\%$.

Answer to RQ3.3: FTMES@30 shows a best cost-benefit relationship among the MBFL's techniques when the same priority is given to runtime and efficacy. Even when reducing the cost priority to 30% and increasing the benefit to 70%, FTMES@30 maintained the dominance over all techniques. Therefore, we consider the FTMES@r as the best cost-benefit relationship in terms of runtime approximation and efficacy improvement compared to SBFL technique.

6 Related works

Among the fault localization techniques (WONG et al., 2016), the most studied and tested fault localization strategies are (PEARSON et al., 2017; WONG; DEBROY; CHOI, 2010): spectrum-based (ABREU; ZOETEWEIJ; GEMUND, 2007), (JONES; HARROLD, 2005), (WONG et al., 2012, 2014), slice-based (XUAN; MONPERRUS, 2014), and mutation-based (MOON et al., 2014; PAPADAKIS; TRAON, 2012, 2014, 2015).

6.1 Fault Localization Techniques

Our research considers Dstar as a spectrum-based fault localization baseline (SBFL) because it was indicated as the best spectrum-based technique that working with mutation-based approaches (PEARSON et al., 2017). In Mutation-based Fault Localization context, MUSE (MUtation-baSEd) (MOON et al., 2014) and Metallaxis (PAPADAKIS; TRAON, 2012) are two MBFL pioneer techniques. Previous Works indicate that the Matallaxis technique overcome the MUSE in the efficacy of ranked list terms (PAPADAKIS; TRAON, 2012, 2015, 2014). For this reason, we considered the mechanism for killing mutants of Matallaxis as a representant of traditional MBFL. We also considered a class of hybrid techniques (the best hybrid techniques validated in real faults) (PEARSON et al., 2017; PEARSON, 2016).

In terms of efficacy, MCBFL-hybrid-avg (PEARSON et al., 2017) is currently the best technique that uses mutants for fault localization, evaluated against real faults. However, it still presents a high computational cost compared to SBFL techniques. In an experiment with real faults, Pearson et al. (PEARSON et al., 2017) reports a runtime of mutants around 32 to 168 hours for some faults.

The MRSBFL-hybrid-max (PEARSON et al., 2017) technique emerges for a more scalable MBFL, where, without performing mutants and using only mutant coverage. It is interesting to demonstrate efficacy in the rank of list a little higher than other SBFL techniques. However, it is much lower than the MCBFL-hybrid-avg technique.

MCBFL-hybrid-avg overcomes our approach (FTMES) in the efficacy of ranked lists, but with negligible statistical significance, as Section 5.2. But, if the cost for obtaining the rank is considered, FTMES stands out by reducing in 90% of the cost of MCBFL-hybrid-avg on average producing solutions very close to MCBFL-hybrid-avg in terms of efficacy.

6.2 Reduction Strategies for MBFL

The Mutation-based Fault Localization techniques can be as costly as Mutation Analysis when it is added mutant execution to perform the FL activity. In this context, approaches have been proposed in distinct steps of location process when mutants are used: i) the strategy of reducing mutation operators (PAPADAKIS; TRAON, 2014); ii) strategies of reducing mutants (PAPADAKIS; TRAON, 2012) and (LIU et al., 2017), and iii) the dynamic mutation execution strategy (GONG; ZHAO; LI, 2015). Our approach fits in the mutation execution strategies scope, once the other approaches can be used in previous steps.

Liu et al. (2017) purposed the reduction strategy called SOME (Statement-Oriented Mutant Reduction Strategy) for MBFL. Our approach reduces the mutation cost at statement level. Despite SOME is named "Statement", our approach is different. SOME is a simple mutant sampling approach applied to MBFL context. The SFilter@r component of our approach is a genuine statement selection performed before generating mutants. Furthermore, our approach uses the SBFL's suspiciousness to select only one portion of failed tests' covered statements. Thus, it is possible to apply together: i) statement, ii) a reduction mutants or operator approach and iii) the mutation execution strategy (ex.: FTMES). For this reason, SOME is note used as baseline.

The dynamic mutation execution strategy (DMES) (GONG; ZHAO; LI, 2015) is an approach that aims to turn the MBFL more efficient, optimizing the form in which the mutants are executed. In an empirical study, the authors employed a Siemens software set with (DO; ELBAUM; ROTHERMEL, 2005) with artificially inserted faults, showing a reduction in the number of mutant executions from 32% to 87% with the same effectiveness of the original MBFL (GONG; ZHAO; LI, 2015; LIU et al., 2017). DMES is a directly baseline in our research that uses the FTMES component to optimizes the mutants execution. Differing of previous study (GONG; ZHAO; LI, 2015) with artificial faults, we apply DMES on Defects4J benchmark (JUST; JALALI; ERNST, 2014) containing projects with real faults. Further, we apply the mutation execution strategies in MBFL traditional and hybrid MBFL, as explained in Section 2.2.

We concluded that canonical DMES a significant reduction of 85,02% and 85,15% applied to MBFL and MCBFL-hybrid-avg, respectively. Our optimization in canonical mechanism of DMES (DMES_SAM) reaches better reductions 91,80% and 92,17% (MBFL and MCBFL-hybrid-avg, respectively) when compared with its canonical approach. The FTMES incorporated to the MCBFL-hybrid-avg technique reaches 99,20% of cost reduction, on average. Compared to canonical DMES, FTMES was 14% superior and 7% most efficient than DMES_SAM. Thus, the FTMES is most efficient than all actual mutation execution strategies.

The use of MBFL approach brings a high cost and, for this reason, the expectation is that MBFL overcome the SBFL's techniques in terms of efficacy. The use of mutation execution strategy does not reduce significantly the efficacy of MBFL approaches. Our study reveals that DMES approach re-

duces the computational cost but also reduces the MBFL's efficacy. The SBFL technique had higher efficacy than DMES technique. In other words, DMES was more costly and less efficacy. Contrasting, the FTMES technique demonstrates better efficacy than SBFL studied technique. We believe that DMES' drawback is that it was validated in software with artificial faults.

Our complete approach is FTMES@r purposed with two components: i) SFilter@r and ii) FTMES. We explore the SBFL scalability and accuracy using some positions of SBFL ranking to be further improved by the MBFL technique. Further, we use the FTMES as the mutation execution strategy component to optimize the mutant executions.

The best of our knowledge, our approach is first to use of SBFL technique to select the smaller ranking for MBFL improvement integrated with a mutation execution strategy. In this sense, the FTMES@r's statements selection level is a new level of cost reduction techniques for MBFL.

7 Threats of Validity

The quality of FTMES@r approach is linked to the SBFL's efficacy and the test set quality. If the given @r value is not sufficient to rank the fault among the Ranking@r statements, the FTMES@r does not have improvements' power. In this sense, if it is need a big value for @r the difference of reduction between FTMES@r and FTMES performance can be insignificant.

Furthermore, the proposed approach uses the FTMES component to improve the cost of MBFL techniques. FTMES is linked to the quality of the test set, similar to many tests and debugging techniques. The FTMES component presented good solutions when the size of the failed test case set is smaller than the successful test case set. This feature is perceived on mostly real scenario and benchmark on this area. Although unusual, there may be scenarios where this case does not happen.

FTMES was compared with DMES, the current mutant execution strategy form MBFL (GONG; ZHAO; LI, 2015; LIU et al., 2017, 2018). DMES authors published results considering mutants generated from Proteum (DELAMARO; MALDONADO, 2001), a C Programming Language tool with 77 mutation operators. Our evaluation considered mutants generated from Major (JUST, 2014), a tool proposed with only 8 mutation operators for Java language. Moreover, mutants generated by Major tool does not consider redundant mutants (JUST; SCHWEIGGERT, 2015).

Despite the fact that DMES accept mutant reduction on earlier stages, as well as its framework being independent of language or mutation tool, it is possible that DMES have a better performance on real fault when utilizing Proteum tool on C Programming Language context. Furthermore, it is possible that our approach shows unsatisfactory results depending on the nature of the fault and the mutation operators used.

8 Conclusion and Future Works

This paper presents the FTMES@r method to improve the top@r ranking positions for Spectrum-based Fault Localization techniques. The FTMES@r framewok has two main components: i) SFilter@r and ii) FTMES that filter the top@r statements and execute the generated mutants set efficiently, respectively. The idea of SFilter@r is to reduce the amount of statements to be improved by MBFL approaches. The insight behind FTMES component is that of avoiding the execution of mutants against passed test cases and thus replacing the killing information from their runs with the coverage information of those test cases against the original program.

For our experiments, we used 221 real faults, ten fault localization techniques and five metrics for evaluating the solutions. We compared the efficiency and efficacy of our approach FTMES with two other mutation execution strategies.

In terms of efficiency, FTMES component outperformed others considering the test cases' runtime and *Mutant-Test Pair (MTP)* executions metrics. The MBFL (PAPADAKIS; TRAON, 2012) and MCBFL-hybrid-avg (PEARSON et al., 2017) are the conventional MBFL techniques. FTMES achieved lower executions than conventional techniques in terms of MTP when compared to these conventional techniques. Before this, the FTMES component is the best current mutation execution strategy of literature. Furthermore, the FTMES@r cost reduction improved the efficiency more than FTMES when it works alone and also compared to SFilter@r and MCBFL-hybrid-avg's interaction.

Analyzing the FTMES component, in terms of fault localization efficacy, the MCBFL-hybrid-avg (PEARSON et al., 2017) had the best results to Exam Score (WONG et al., 2008), Accuracy and Wasted Effort metrics (PARNIN; ORSO, 2011). However, we highlight that FTMES obtained similar results to MCBFL-hybrid-avg. That is, MCBFL-hybrid-avg was better than FTMES on average, but such result is statistically non-significant. Thus, the FTMES component reached a better cost reduction compared to MCBFL-hybrid-avg without significant loss of solutions' efficacy.

FTMES@r's framework improves the SBFL's Ranking producing higher efficacy of localization in all experiments. For example, the FTMES@30 reached better results than FTMES when it works alone, in terms of runtime reduction and efficacy improvement. Beyond, the interaction between SFilter@r and MCBFL-hybrid-avg also reached considerable cost reduction with approximated efficacy. However, the FTMES@r was most efficient with efficacy's results close to MCBFL-hybrid-avg: the best efficacy of localization with worst computational cost. Furthermore, FTMES@r shows a better cost-benefit relationship among the MBFL's techniques in terms of runtime approximation and efficacy improvement compared to SBFL technique performance.

For future work, our proposal could be combined with other techniques (FU et al., 2017; OLIVEIRA; CAMILO-JUNIOR; VINCENZI, 2013) aiming at improving the efficacy of program elements ranking as well as reducing the computational cost.

Acknowledgments

The authors would like to thank CAPES (Finance Code 001), the Federal University of Mato Grosso do Sul (UFMS) and, the Federal Institute of Education, Science and Technology of Mato Grosso (IFMT) for their financial support.

References

ABREU, R.; ZOETEWEIJ, P.; GEMUND, A. J. C. van. On the accuracy of spectrum-based fault localization. In: *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*. Washington, DC, USA: IEEE Computer Society, 2007. (TAICPART-MUTATION '07), p. 89–98. ISBN 0-7695-2984-4. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=1308173.1308264⟩.

CAMPOS, E. C.; MAIA, M. d. A. Common bug-fix patterns: A large-scale observational study. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.: s.n.], 2017. p. 404–413.

CAMPOS, J. et al. Gzoltar: an eclipse plug-in for testing and debugging. In: *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2012. p. 378–381.

DELAMARO, M. E.; MALDONADO, J. C. Mutation testing for the new century. In: WONG, W. E. (Ed.). Norwell, MA, USA: Kluwer Academic Publishers, 2001. cap. Proteum/IM 2.0: An Integrated Mutation Testing Environment, p. 91–101. ISBN 0-7923-7323-5.

DEMILLO, R.; LIPTON, R.; SAYWARD, F. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, v. 11, n. 4, p. 34–41, abr. 1978. ISSN 0018-9162.

DO, H.; ELBAUM, S. G.; ROTHERMEL, G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering: An International Journal*, v. 10, n. 4, p. 405–435, 2005.

FREITAS, D. et al. Evolutionary composition of customized fault localization heuristics. In: *Proceedings of the 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. [S.l.: s.n.], 2018.

FREITAS, D. et al. Mutation-based evolutionary fault localisation. In: *Proceedings of the IEEE Congress on Evolutionary Computation*. [S.l.]: IEEE, 2018.

FU, W. et al. A test suite reduction approach to improving the effectiveness of fault localization. In: *2017 International Conference on Software Analysis, Testing and Evolution (SATE)*. [S.l.: s.n.], 2017. p. 10–19.

GONG, P.; ZHAO, R.; LI, Z. Faster mutation-based fault localization with a novel mutation execution strategy. In: *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.: s.n.], 2015. p. 1–10.

JONES, J. A.; HARROLD, M. J. Empirical evaluation of the tarantula automatic fault-localization technique. In: *in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2005. p. 273–282.

JUST, R. The major mutation framework: Efficient and scalable mutation analysis for java. In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2014. (ISSTA 2014), p. 433–436. ISBN 978-1-4503-2645-2. Disponível em: ⟨http://doi.acm.org/10.1145/2610384.2628053⟩.

JUST, R.; JALALI, D.; ERNST, M. D. Defects4J: A Database of existing faults to enable controlled testing studies for Java programs. In: *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*. San Jose, CA, USA: [s.n.], 2014. p. 437–440. Tool demo.

JUST, R.; SCHWEIGGERT, F. Higher accuracy and lower run time: Efficient mutation analysis using non-redundant mutation operators. *Softw. Test. Verif. Reliab.*, John Wiley and Sons Ltd., Chichester, UK, v. 25, n. 5-7, p. 490–507, ago. 2015. ISSN 0960-0833. Disponível em: ⟨http://dx.doi.org/10.1002/stvr.1561⟩.

LIU, Y. et al. Statement-oriented mutant reduction strategy for mutation based fault localization. In: *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. [S.l.: s.n.], 2017. p. 126–137.
LIU, Y. et al. An optimal mutation execution strategy for cost reduction of mutation-based fault localization. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 422, n. C, p. 572–596, jan. 2018. ISSN 0020-0255. Disponível em: ⟨https://doi.org/10.1016/j.ins.2017.09.006⟩.
MOON, S. et al. Ask the mutants: Mutating faulty programs for fault localization. In: *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*. Washington, DC, USA: IEEE Computer Society, 2014. (ICST '14), p. 153–162. ISBN 978-1-4799-2255-0.
NIST. National institute of standards and technology. https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf. 2002.
OLIVEIRA, A. A. L. de; CAMILO-JUNIOR, C. G.; VINCENZI, A. M. R. A coevolutionary algorithm to automatic test case selection and mutant in mutation testing. In: *2013 IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2013. p. 829–836. ISSN 1089-778X.
OLIVEIRA, V. et al. Improved representation and genetic operators for linear genetic programming for automated program repair. *Empir Software Eng*, Springer, v. 23, n. 5, p. 2980–3006, 2018. ISSN 1573-7616.
PAPADAKIS, M.; TRAON, Y. L. Using mutants to locate "unknown" faults. In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. [S.l.: s.n.], 2012. p. 691–700.
PAPADAKIS, M.; TRAON, Y. L. Effective fault localization via mutation analysis: A selective mutation approach. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2014. (SAC '14), p. 1293–1300. ISBN 978-1-4503-2469-4.
PAPADAKIS, M.; TRAON, Y. L. Metallaxis-fl: Mutation-based fault localization. *Softw. Test. Verif. Reliab.*, John Wiley and Sons Ltd., Chichester, UK, v. 25, n. 5-7, p. 605–628, ago. 2015. ISSN 0960-0833.
PARNIN, C.; ORSO, A. Are automated debugging techniques actually helping programmers? In: *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2011. (ISSTA '11), p. 199–209. ISBN 978-1-4503-0562-4. Disponível em: ⟨http://doi.acm.org/10.1145/2001420.2001445⟩.
PEARSON, S. Evaluating and improving fault localization techniques. In: *Technical report UW-CSE-16-08-03, August 2016*. [S.l.: s.n.], 2016.
PEARSON, S. et al. Evaluating and improving fault localization. In: *Proceedings of the 39th International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2017. (ICSE '17), p. 609–620. ISBN 978-1-5386-3868-2. Disponível em: ⟨https://doi.org/10.1109/ICSE.2017.62⟩.
SOHN, J.; YOO, S. Fluccs: Using code and change metrics to improve fault localization. In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2017. (ISSTA 2017), p. 273–283. ISBN 978-1-4503-5076-1. Disponível em: ⟨http://doi.acm.org/10.1145/3092703.3092717⟩.
STEIMANN, F.; FRENKEL, M.; ABREU, R. 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*. New York, NY, USA: ACM, 2013. (ISSTA 2013), p. 314–324. ISBN 978-1-4503-2159-4.
WONG, E. et al. A crosstab-based statistical method for effective fault localization. In: *2008 1st International Conference on Software Testing, Verification, and Validation*. [S.l.: s.n.], 2008. p. 42–51. ISSN 2159-4848.
WONG, W. E.; DEBROY, V.; CHOI, B. A family of code coverage-based heuristics for effective fault localization. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 83, n. 2, p. 188–208, fev. 2010. ISSN 0164-1212.
WONG, W. E. et al. The dstar method for effective software fault localization. *IEEE Transactions on Reliability*, v. 63, n. 1, p. 290–308, March 2014. ISSN 0018-9529.
WONG, W. E. et al. Software fault localization using dstar (d*). In: *2012 IEEE Sixth International Conference on Software Security and Reliability*. [S.l.: s.n.], 2012. p. 21–30.

WONG, W. E. et al. A survey on software fault localization. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 42, n. 8, p. 707–740, ago. 2016. ISSN 0098-5589.
XUAN, J.; MONPERRUS, M. Test case purification for improving fault localization. In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: ACM, 2014. (FSE 2014), p. 52–63. ISBN 978-1-4503-3056-5. Disponível em: ⟨http://doi.acm.org/10.1145/2635868.2635906⟩.
YOO, S.; HARMAN, M. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 83, n. 4, p. 689–701, abr. 2010. ISSN 0164-1212.