Journal of Software Engineering Research and Development, 2024, 12:12, doi: 10.5753/jserd.2024.3651

Uniqueness of suspiciousness scores: towards boosting evolutionary fault localization

Willian de Jesus Ferreira [**Federal University of Goiás** | ***willianjferreira@gmail.com***]
Plinio S. Leitao-Junior [**Federal University of Goiás** | ***plinio@inf.ufg.br***]
Diogo Machado de Freitas [**Federal University of Goiás** | ***diogomfreitas92@gmail.com***]
Deuslirio Silva-Junior [**Federal University of Goiás** | ***deuslirio.junior@gmail.com***]
Rachel Harrison [**Oxford Brookes University** | ***rachel.harrison@brookes.ac.uk***]

Abstract

Context. Software is subject to the presence of faults, which impacts its quality as well as production and maintenance costs. Evolutionary fault localization uses data from the test activity (test spectra) as a source of information about defects, and its automation aims to obtain better accuracy and lower software repair cost. *Motivation.* Our analysis identified that the test spectra commonly used in research exhibit a high ratio of sample repetition, which impairs the training and evolution of Genetic Programming (GP) evolved heuristics. *Problem.* We investigate whether evolutionary training based on the uniqueness of suspiciousness scores can increase the ability to find software faults (defects), even in repeat-sample scenarios. Specifically, we examine whether the GP-evolved heuristic, which is based on the distinguishability of program elements in terms of faults, is really competitive. *Methodology.* The investigation formalized hypotheses, introduced three training strategies to guide the research and carried out an experimental evaluation, aiming to reach conclusions regarding the assessment of research questions and hypotheses. *Analysis.* The results have shown the competitiveness of all the proposed training strategies through evaluation metrics commonly used in the research field. *Conclusion.* Statistical analyses confirmed that the uniqueness of suspiciousness scores guides the generation of superior heuristics for fault localization.

Keywords: *Fault Localization, SBSE, Search Based Software Engineering, Software Debugging, Genetic Programming*

1 INTRODUCTION

The software debugging process requires costly resources, such as time and human effort, to find the causes of software failures. Fault Localization (FL) (Wong et al., 2016) is an automated software debugging technique that aims to guide human developers in identifying the cause of the failure so that it can be repaired. (Hailpern and Santhanam, 2002) stated software debugging involves finding and fixing the faulty code whose execution violates a known specification and causes software to fail.

The automation of the process of locating faults has been the subject of several studies in recent years, (Wong et al., 2016). Many techniques are based on heuristics that use the information obtained from the execution of tests (i.e. test spectra) in order to calculate how suspicious each program element is (i.e. how likely it is to be defective).

A test spectrum is obtained from the execution of a set of test cases and usually refers to the control flow coverage with respect to each program element for both negative tests (test cases that cause the program to fail) and positive tests (test cases where the program succeeds). Several methods are based on test spectra, as they are sources of knowledge about software faults. For example, the tester may keep an eye on those program elements covered in negative tests.

Search-based fault localization (SBFL) (Leitao-Junior et al., 2020) is the research field in which fault localization is perceived as a search problem, such that test spectra are used to train Genetic Programming (GP) evolved heuristics. In this context, scientific literature on the understanding of the use of search-based algorithms improves the capacity to locate faults, presenting results that would not be obtained by traditional methods (Leitao-Junior et al., 2020; Wang et al., 2011; Wong et al., 2016; Yoo, 2012).

Our investigation analyzed test spectra from *Defects4J* (Just et al., 2014), a fault localization benchmark composed of real faulty programs. This benchmark is representative and commonly used to evaluate search-based fault localization approaches.

One of our findings is that less than 12% of program elements from the analyzed test data are distinguishable from their training samples. This impacts the heuristics' ability to differentiate whether program elements are defective or not. We discovered that the training data limits the effectiveness of fault localization heuristics in experiments conducted within the research field. This limitation is mainly due to the redundancy of sample data, which leads to a consequent degradation in learning capacity during the evolution of GP-evolved heuristics using the canonical Genetic Programming metaheuristic.

The knowledge of the occurrence of repeated samples in the Genetic Programming training process prompted us to consider whether an exploration based on how distinguishable program elements are in terms of defectiveness can generate competitive fault localization heuristics.

As a consequence of this analysis, we hypothesize that the uniqueness of suspiciousness values (i.e. the uniqueness of scores that rank program elements related to being faulty) is a factor that potentially guides the generation of heuristics through Genetic Programming, a search-based evolutionary metaheuristic proposed by (Koza, 1992), in such a way that it outperforms canonical GP-evolved heuristics for locating software faults. In terms research questions: (i) Is the uniqueness of suspiciousness values related to the ability to localize faults in the context of GP-evolved heuristics? (ii) If so, is the approach competitive compared to the baselines in the research field?

To the best of our knowledge, this is the first time that *the uniqueness of suspiciousness values* has been explored as an element to potentially improve software fault localization.

The paper is structured as follows. Section 2 includes background and related work. Section 3 discusses the motivation for investigating data from the testing activity, which is commonly used as the information source in evolutionary methods of Fault Localization. The proposed method is introduced in Section 4. Section 5 describes the empirical evaluation conducted in order to evaluate the proposal. The results obtained are discussed in Section 6. Section 7 deals with the statistical analysis of the results, as well as the answers to research questions and the appraisal of hypotheses. Threats to validity are covered in Section 8. At the end, Section 9 addresses final comments.

2 BACKGROUND AND RELATED WORK

Test spectra are a collection of data that provide a specific view of the behavior of software during testing activities and offer information on the number of positive and negative tests in which program elements are executed. In this context, four coverage variables are used to characterize a program element:

- ***es*** and ***ns*** represent the number of positive test cases that do execute and not execute the observed element, respectively;
- ***ef*** and ***nf*** represent the number of negative test cases that do execute and not execute the observed element, respectively.

Spectrum-based Fault Localization (SFL) (Wong et al., 2016) techniques use such execution frequency of program elements to calculate how likely they are to be faulty. On this basis, each program element is assigned four values (related to es, ns, ef and nf coverage variables) as input to a suspiciousness score function (i.e. input to a fault localization heuristic). Then these scores are used to rank program elements with respect to being defective.

2.1 Fault localization heuristics based on test spectra

Fault localization heuristics based on test spectra make up one of the main research trends in automating the fault location process. They are equations (functions) grounded on studies that explore test spectra, and often called *heuristics developed by humans*.

The Tarantula heuristic is the forerunner in this category and was proposed by (Jones et al., 2009). The heuristic was built in order to penalize code elements that are executed more frequently by negative tests in relation to elements that are less frequently executed by this type of test.

Another heuristic widely referenced in the literature is the Ochiai coefficient, which was originally used to calculate genetic similarity in molecular biology. Ochiai was introduced as a fault localization heuristic by (Abreu et al., 2006). (Wong et al., 2016) present the main heuristics used in studies of the fault localization research field.

2.2 Evolutionary approaches to fault localization

The fault location challenge can be treated as an optimization problem, allowing search-based algorithms to be used to fully or partially automate solutions on fault localization (Leitao-Junior et al., 2020). Evolutionary approaches deal with populations of individuals, which evolve over generations, where each individual is a potential solution to the problem. Typically an individual represents a formula (i.e. a heuristic) that aims to calculate the suspiciousness score, that denotes the potential for a program element to be defective.

Researchers in the field often apply evolutionary approaches such as Genetic Algorithms (GA) (Wang et al., 2011), (Campos et al., 2013), (Silva-Junior et al., 2020) or Genetic Programming (GP) (De-Freitas et al., 2018), Sohn and Yoo (2017), (Yoo, 2012), to derive heuristics for the purpose of measuring the chances of each program element being defective. Both GA and GP are called metaheuristics.

Recent research seeks to enhance the effectiveness of evolutionary approaches in terms of their ability to locate faults. (Choi et al., 2018) formulated a multi-objective approach by using GP to generate classification heuristics that not only aim to place the failing program element in a higher position in the ranking but also aim to assign scores as low as possible to the program elements without faults. In addition to control flow spectra, the competitiveness of GP-evolved heuristics applied to mutation spectra is shown in (De-Freitas et al., 2018). In (Silva-Junior et al., 2020) an evolutionary algorithm to search sets of weights and combine heuristics from different data sources is proposed (control flow and data flow, as well as a hybrid strategy).

2.3 Genetic programming in fault localization

Genetic Programming (GP) is the most used metaheuristic algorithm in the fault localization research field, mainly because of its ability to generate suspicion formulas applied to the context (Leitao-Junior et al., 2020).

GP consists of the evolution of computer programs using analogies with many mechanisms used by the evolution of species and has evolutionary and mutative phenomena as a reference.

GP uses data from known inputs (e.g. test spectra) and outputs to generate GP-evolved heuristics. The first generation is created (in many cases, randomly), starting the GP cycle. As in biology, new individuals in each generation inherit characteristics from their parents. At each generation, individuals (i.e. fault localization heuristics) are evaluated according to a fitness function to determine how well the individual is adapted. The cycle repeats until the stopping criterion is reached. This criterion can be the maximum number of generations or the peak of the fitness of the new individuals.

New individuals are generated to be part of the evolutionary process using parameters such as population size, number of generations, type of selection, crossing, mutation and reproduction rate.

In summary, since GP-evolved heuristics are equations, they are used to infer the probability that a program element is defective. The GP-evolved heuristics take the test spectrum data as input to create the ranking of suspect elements.

3 MOTIVATION

Data from the testing activity - *test spectrum* - is frequently used as an information source in fault localization heuristics in 2 categories: developed by humans and evolved by evolutionary methods. Specifically, by applying a set of test cases, the values of spectrum variables (i.e. *ef*, *ep*, *nf*, *np*) are collected from each program element, as described in Section 2.

The test spectrum is an integral part of benchmarks, which are used in the development and evaluation of fault localization approaches. For instance, in (Leitao-Junior et al., 2020) the *benchmarks* preferred by studies on search-based methods are presented, and *Defects4J* (Just et al., 2014) was found to be one of the most popular in the search field.

This section focuses on *Defects4J*, which is a representative benchmark of real faults and real programs, and presents an analysis of the test spectrum of that benchmark. The objective is to identify factors within test spectrum entries that may potentially degrade the performance of heuristic-based approaches. By understanding these elements, we aim to explore and incorporate them into the development of new and improved heuristics.

3.1 Defects4J

Defects4J was proposed by (Just et al., 2014) as a *framework* that offers a database and a project structure with real faults in Java projects, in line with the statement by (Lu et al., 2005): a good *benchmark* should be able to demonstrate the strengths and weaknesses of each tool.

The goal of (Just et al., 2014) was to provide data for replicable studies, with robust and easy integration for software testing research. The benchmark consists of a collection of reproducible bugs and a support infrastructure to promote advances in software engineering. In its initial release, *Defects4J* has 357 real faults for five open-source programs, namely: *JFreeChart*, *Closure Compiler*, *Commons Math*, *Joda-Time* and *Commons Lang*.

Table 1 introduces some features of each *Defects4J* program in version 1.1.0. The table presents the number of defective versions per program and the average number of elements.

We adopted the *Defects4J* repository created and made available by (Pearson et al., 2017), as data source for the analysis and to carry out experiments in the present investigation.

Table 1. Defects4J programs

Program	Chart	Closure	Lang	Math	Mockito	Time
Number of versions	26	133	65	106	38	27
Mean of Elements	4168	16714	832	2266	1820	5078

3.2 Test spectrum analysis

This subsection analyzes the test spectrum, highlighting the uniqueness of its entries. A set of definitions is also introduced, which will be useful in clarifying the paper's proposal (Section 4).

Definition I - (***program version***). Several versions of a program are produced during the software cycle, as a consequence of its evolution. Let V^P be the set of versions of Program P. And let v_i be one of such versions (i.e. v_i is a program version), since $v_i \in V^P$.

Definition II - (***faulty program version***). A version of Program P is said to be a *faulty version* if it has behavior other than the specification of P.

Definition III - (***program element***). In software testing and debugging research, a program is decomposed into elements, in order to study properties of each element. Let E^P be the set of elements of Program P, and let e_j be one of such elements since $e_j \in E^P$ (i.e. e_j is a *program element*).

Definition IV - (***faulty element***). In a faulty version of Program P, there are one or more elements in P, called *faulty elements*, which are identified as the cause of the program's failure.

In the context of fault localization heuristics, a program element $e_j \in E^P$ is usually perceived in various granularities (perspectives), namely: command, control flow block, class method, data flow association, mutant, among others. The focus of the analysis will determine the perspective adopted for the program element. For example, the contents of Table 1 adopt *command* as the granularity for program element, so *mean of elements* denotes *average number of commands* among the various versions of the program.

Definition V - (***test spectrum entry***). A *test spectrum entry* is the quadruple $[ef, es, nf, ns]$ that refers to the values of spectrum variables. $TSE(e, v)$ is a function to compute the quadruple related to program element ***e*** of faulty program version ***v*** (*TSE* stands for *test spectrum entry*).

Definition VI - (***uniqueness of test spectrum entries***). The *uniqueness of test spectrum entries* refers to the number of distinct $[ef, es, nf, ns]$ quadruples related to all program elements. $uTSE(v)$ is a function to compute the uniqueness value to program version ***v*** (*uTSE* stands for *uniqueness of test spectrum entries*).

A test spectrum entry represents the data collected from spectrum variables, concerning a particular element of a program version, by applying a set of test cases. Thus, the number of test spectrum entries refers to the number of elements of the program version being analyzed. $uTSE$ is limited by the number of test spectrum entries, and usually there are repeated entries in the test spectrum, so a $uTSE$ value is typically lower than TSE. The reasons for this are diverse, such as: control flow dependence (i.e. two or more elements are always executed by the same test cases, regardless of the test cases) and low quality of the test cases (i.e. benchmark test cases can not distinguish two or more elements when they have no control flow dependence).

Definition VII - (***uniqueness ratio of test spectrum entries - uTSE ratio***). The *uTSE ratio* refers to how unique test spectrum are entries with respect to the number of program elements (the ratio ranges from 0.00 to 1.00, and higher values are better).

In the ideal scenario, the test spectrum consists of unambiguously distinct samples representing program elements. In other words, the best uniqueness ratio of test spectrum entries is 1.00, but that is theoretical due to control flow dependencies as well as non-executable program elements. This ideal means that each program element should have a unique test spectrum entry, so program elements are distinguishable from one another by the test case set.

Figure 1 shows the analysis of *Defects4J* regarding the uniqueness of test spectrum entries. Far from the ideal case, the figure portrays a real scenario of benchmarked programs (each circle represents a program). The abscissa axis denotes the number of the program element (i.e. the number of test spectrum entries), the ordinate axis refers to the uniqueness ratio of test spectrum entries, and size of each circle is proportional to the absolute number of distinct test spectrum entries to that program.

Figure 1 also shows the uniqueness ratio is less than 0.12 for all programs, which reveals that in general, most test spectrum entries are not able to locate software fault precisely. Related to circle sizes, *Closure* has the highest number of distinct test spectrum entries, but its uniqueness ratio is less than 0.06, which means less than 6% of elements have unique entries in the test spectrum. In that sense, *Mokito* and *Time* have better samples related to the others.

3.3 Suspiciousness values analysis

This subsection introduces definitions that support the understanding and the development of the proposal presented in Section 4.

Definition VIII - (***suspiciousness value***). As seen earlier, the *suspiciousness value* denotes how suspicious (likely) a program element is to be defective. It ranges from 0.00 to 1.00, and higher values mean greater suspiciousness. $SV(h, e, v)$ is a function to compute such a value for the heuristic ***h***, related to program element ***e*** of faulty version ***v*** (SV stands for *suspiciousness value*).

Definition IX - (***uniqueness of suspiciousness values***). The *uniqueness of a suspiciousness value* refers to the number of distinct suspiciousness values related to all program elements. $uSV(h, v)$ is a function to compute such a value for heuristic ***h***, related to faulty version ***v*** (uSV stands for *uniqueness of suspiciousness values*).

Table 2 illustrates some of the definitions introduced so far. In this example, the program has 10 elements (e_1, e_2, ..., e_{10}), where e_7 is the faulty one. The *uniqueness of test spectrum entries* is six ($uTSE = 6$), as there are six unique compositions of spectrum variables (quadruple $[ef, es, nf, ns]$). Distinctively, the *uniqueness of suspiciousness values* is five ($uSV = 5$), as there are five distinct suspiciousness values. In Table 2, Elements e_7 and e_8 share the same *suspiciousness value* (both are 0.89), despite their *test spectrum entries* being different ($[2, 10, 0, 77]$ and $[1, 11, 0, 76]$, respectively). That means the fault localization heuristic (i.e. the suspiciousness function) was not able to distinguish e_7 and e_8 with respect to the odds of being defective.

Table 2. Examples of *test spectrum entries* and *suspiciousness values*.

Element	*ef*	*es*	*nf*	*ns*	Susp Value
e_1	1	0	1	86	1.00
e_2	1	0	1	86	1.00
e_3	1	2	1	84	0.96
e_4	1	10	0	77	0.90
e_5	1	10	0	77	0.90
e_6	1	10	0	77	0.90
e_7	2	10	0	77	0.89
e_8	1	11	0	76	0.89
e_9	1	11	0	76	0.89
e_{10}	0	4	1	83	0.00

Definition X - (***uniqueness ratio of suspiciousness values - uSV ratio***). The *uSV ratio* refers to how unique suspiciousness values are with respect to the number of program elements (the ratio ranges from 0.00 to 1.00, and higher values are better).

Both uniqueness ratios (Definitions VII and X) represent proportions (i.e. values in the range [0, 1]) with respect to the number of program elements:

- ***uTSE ratio and uSV ratio of a program version*** are the values of $uTSE$ and uSV divided by the number of elements of that version, respectively;
- ***uTSE and uSV ratios of a program*** are averages over its versions.

In order to analyze the *uSV ratio*, five human-developed heuristics (see Table 3) were selected as they perform in recent works such as (Pearson et al., 2017) and (Zheng et al., 2018), and they are reference baselines in the research field according to (Wong et al., 2016), namely: $Tarantula$ (Jones et al., 2009), $Ochiai$ (Abreu et al., 2006), $Jaccard$ (Abreu et al., 2007), $Zoltar$ (Janssen et al., 2009), and $Barinel$ (Abreu et al., 2009). Furthermore, Genetic Programming-based heuristics (*GP-evolved* ones) (Yoo, 2012), (De-Freitas et al., 2018), (Sohn and Yoo, 2017) were applied as an evolutionary baseline.

Figure 2 presents the *uSV ratio* of some *Defect4j* programs: *Chart*, *Time*, and *Lang*. Note that heuristics compute a particular value of the $uSVratio$ for each program (e.g. the figure shows three different $uSVratios$ for *Zoltar*), and the *uSV ratio* of a program computed by human-developed heuristics are similar to each other. Even in the best scenario (i.e. GP-evolved heuristics applied to the *Time* program), less than 6% of the program elements are distinguishable as to whether they are suspected to be defective.

Figure 3 analyses the *uSV ratio* against the *uTSE ratio* for the *Time*, *Chart* and *Lang* programs, such that each series (line) refers to a program. That radar chart presents values

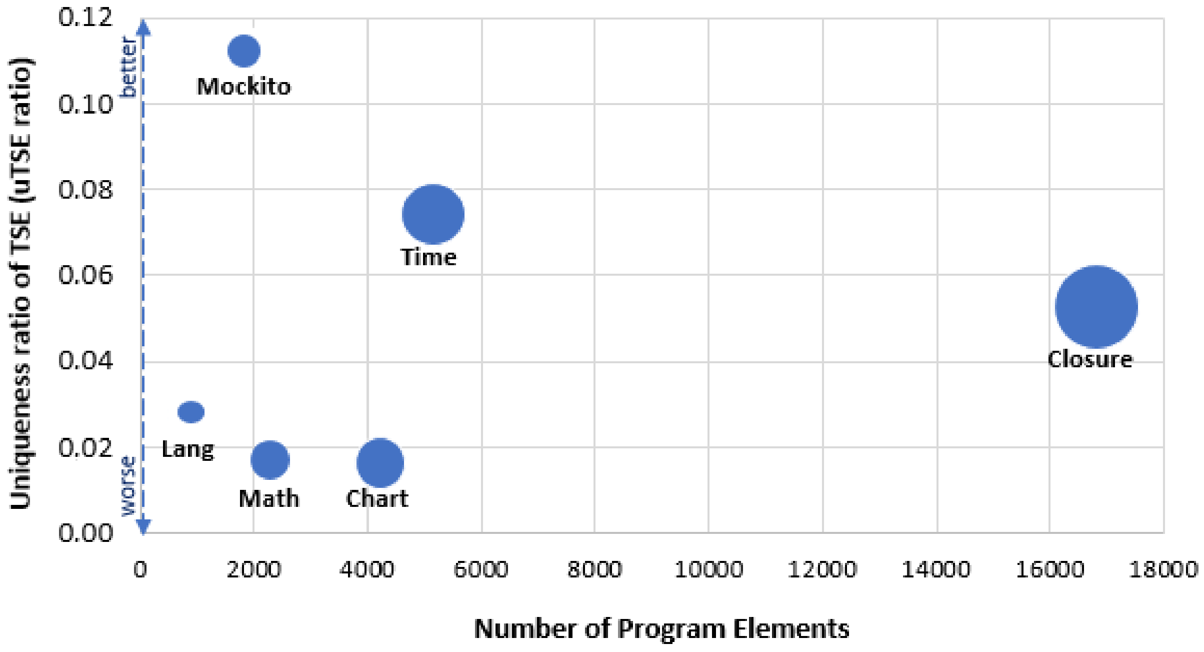

Figure 1. *uTSE ratio* of *Defects4J* Programs.

Table 3. Human-developed fault localization heuristics.

Heuristic	Equation
Tarantula	$\dfrac{\frac{ef}{ef+nf}}{\frac{ef}{ef+nf}+\frac{es}{es+ns}}$
Ochiai	$\dfrac{ef}{\sqrt{(ef+nf)\times(ef+es)}}$
Jaccard	$\dfrac{ef}{ef+nf+es}$
Zoltar	$\dfrac{ef}{ef+nf+es+\frac{10000\times nf\times es}{ef}}$
Barinel	$1-\dfrac{es}{es+ef}$

of $(uSV/uTSE)$, that shows how much the uSV *ratio* preserves $uTSE$ *ratio* (higher values are better).

In Figure 1 the *uTSE ratio* of the *Chart* program is 1.69%, i.e. any fault localization heuristic, which is based on the test spectrum entries, can obtain at most a distinct suspiciousness values of 1.69% in relation to the number of program elements (as a consequence, most program elements have the same suspiciousness value as other elements). However, according to Figure 3, *Ochiai* reduces the uniqueness ratio by 49%, that is, this heuristic is able to distinguish only 0.83% of the program elements with respect to being defective (49% of 1.69% is about 0.83%). In this case, as *Chart* has 4168 elements (see Table 1), there are just 35 distinct suspiciousness values for all program elements (0.83% of 4168 elements is equal to 35 elements). That scenario potentially reduces the fault localization effectiveness.

3.4 Hypotheses toward boosting GP-evolved heuristics

Based on the previous analysis, this subsection introduces hypotheses aiming to contribute to the effectiveness of evolutionary approaches for fault localization.

(Yoo et al., 2017) presented an empirical and theoretical study that provides evidence on the competitiveness of GP against the best heuristics developed by humans. The study claims that GP has developed a formula with virtually the best performance, and no human could ever design a formula that would outperform it.

In line with competitiveness of GP-evolved heuristics (Yoo et al., 2017), Figure 3 shows that GP preserves the uniqueness ratio of TSE better than human-developed heuristics, as computed values of $(uSV/uTSE)$ are the highest per program (81%, 79% and 65% of $uTSE$ for *Chart*, *Time* and *Lang* programs, respectively).

The analysis above leads us to the following hypotheses:

- $Hypothesis H_1$: The closer the *uSV ratio* is to the *uTSE ratio*, the better the fault localization ability. In other words, higher values of the fraction $(uSV/uTSE)$ improve the efficacy to locate faults.
- $Hypothesis H_0$: The value of the *uSV ratio* does not impact fault localization.

The following sections present an method aimed at evaluating hypotheses against the *Defects4J* benchmark, through a protocol that includes supporting research questions, empirical analysis and null-hypothesis significance testing.

4 METHOD

Our method involves the addition of a new component - *uniqueness suspiciousness value* (uSV) - to the learning process of the Genetic Programming (GP) metaheuristics. This

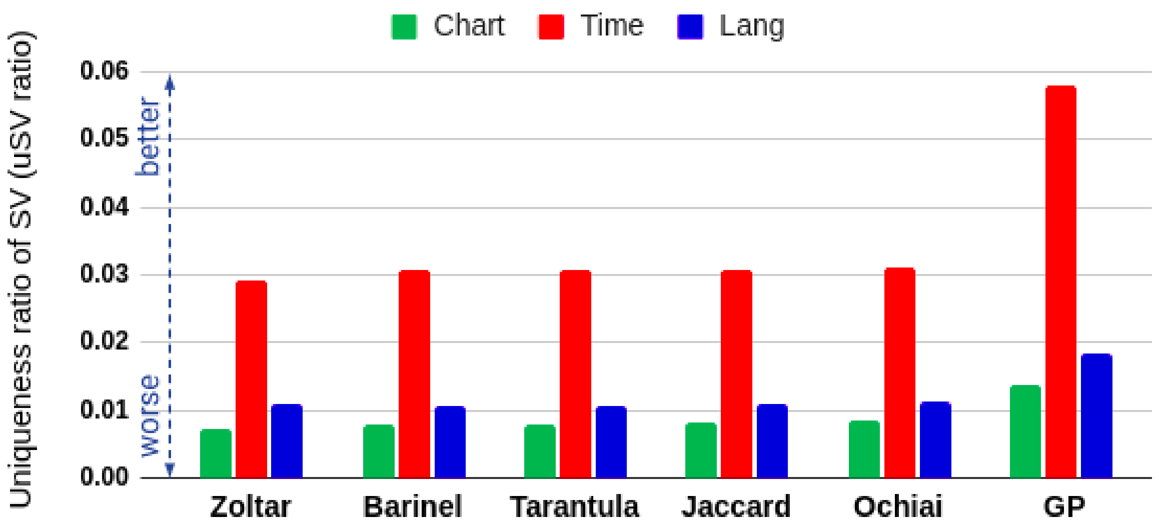

Figure 2. uSV *ratio* of *Defects4J* programs (*Chart*, *Time* and *Lang*).

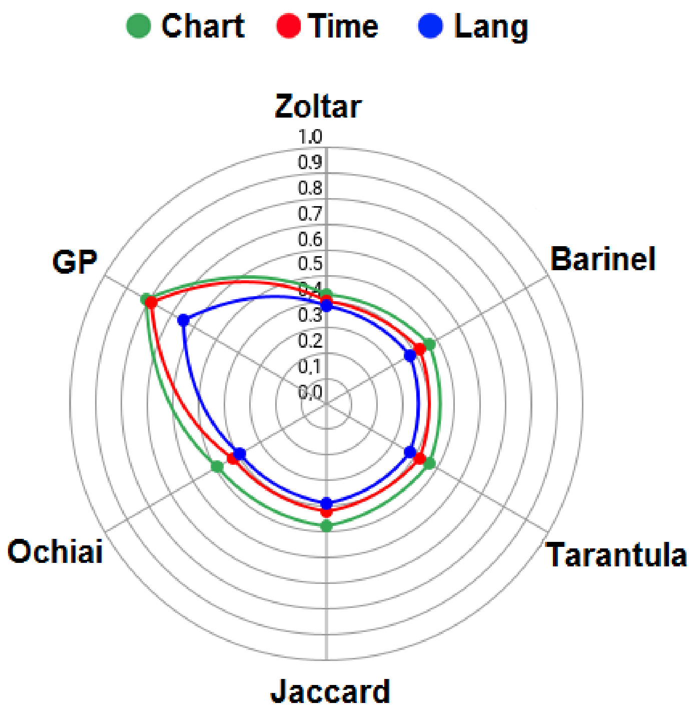

Figure 3. $(uSV/uTSE)$ of *Defects4J* programs (*Chart*, *Time* and *Lang*).

component is leveraged to guide the training of fault localization methods through the promotion of individuals with higher uSV and, consequently, improvement of GP-evolved heuristics performance concerning locating faults.

Basically, we use an evolutionary strategy in which individuals (solutions) with smaller fitness function are privileged for the genetic operators (crossover, mutation and selection). Figure 4 illustrates the basic flow of the canonical GP metaheuristic, but with the insertion of $uSV variable$ to the heuristics evolution.

On incorporating the $uSV variable$ effect to the fitness function, we use a common measure (e.g. number of program elements up to first fault location) but divided by $(uSV * w)$ where $w \in \{1, 3, 5, 7, uSV^2\}$. After preliminary analysis, we observed that values 3, 5, and 7 yielded results that were identified as more promising, prompting us to select them for set w. Then we have proposed three training approaches aiming to assess the raised hypotheses (H_0 and H_1):

- $GP/(uSV * 3)$, the canonical GP fitness function is divided by *three times* uSV.
- $GP/(uSV * 5)$, the canonical GP fitness function is divided by *five times* uSV.
- $GP/(uSV * 7)$, the canonical GP fitness function is divided by *seven times* uSV.

All approaches above promote individuals with higher uSV, since any denominator greater than one in the fitness formulae results in better-adapted individuals (i.e. individuals with lower fitness function are prioritized).

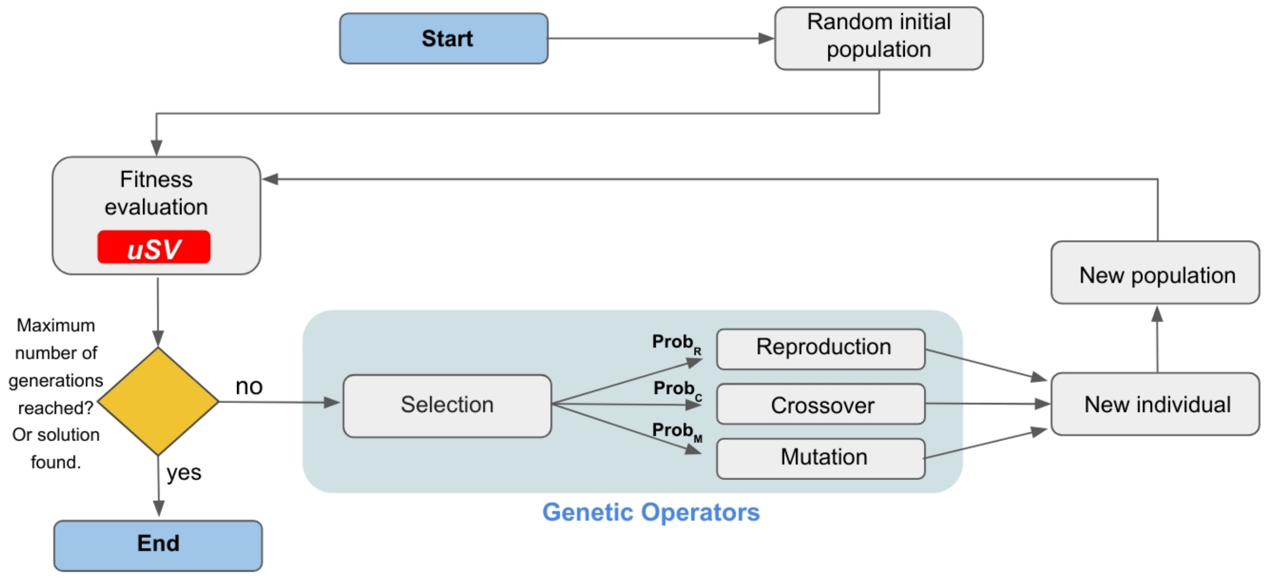

Figure 4. Basic flow of the GP metaheuristic with the new method.

5 EMPIRICAL EVALUATION

We performed an empirical analysis through an experiment to investigate the following research questions:

- ***RQ-1:*** *Is the proposed approach competitive for locating software faults?*
- ***RQ-2:*** *Is the uniqueness of suspiciousness values a factor that guides the generation of effective evolutionary approaches for fault localization?*

5.1 Experiment design

The following aspects were defined when designing the experiment:

Baseline. The research field has shown that Genetic Programming (GP) can generate more effective fault localization heuristics than those proposed by humans (De-Freitas et al., 2018), (Sohn and Yoo, 2017), (Yoo, 2012), (Yoo et al., 2017). So we chose GP-evolved heuristics as the baseline approach.

Benchmark. We use the *Dejects4J* benchmark as it represents real faults in real programs, and it has been explored in the research field, as justified in Subsection 3.1. We've selected *Chart*, *Time* and *Lang* programs, in line with the investigation in Subsection 3.3.

Environment. All experiments were performed on Debian GNU Linux version 10. We use DEAP [1] (Distributed Evolutionary Algorithms in Python) framework version 1.3.1 to implement the GP. The algorithms were run using Python [2] version 3.7.3.

GP parameters. We conduct experiments to explore the appropriate GP parameters regarding the trade-off between the effectiveness of fault localization and the cost of the training process until we reach the following. The population size is 100 individuals, and it was started randomly; Individuals have a tree with a minimum height of 4 and a maximum of 8; GP is configured with a size 3 tournament selection operator; a crossover operator with a rate of 0.8; a subtree replacement mutation operator with a rate of 0.07 and a point mutation operator with a rate of 0.03. The stopping criterion is set at 50 generations.

Tie-break of suspiciousness values. We have applied the worst-case strategy for the tie-break: if two or more program elements have the same suspiciousness score, then all of them are in the worst tied position. For example, if an element is ranked fifth, but its suspiciousness score ties with two other elements, then all three elements are ranked seventh.

Cross Validation and Repeated Runs. To deal with overfitting, we run the experiments by applying 10-fold cross-validation. To reduce the stochastic effect, the experiment was repeated 10 times.

5.2 Evaluation metrics

To assess the effectiveness of the proposed approach, we compared their results against those produced by the baseline (the GP metaheuristic). To this end, we use two evaluation metrics that are widely used in fault localization, according to the meaning described below. The lower the value computed by both metrics, the better the performance in locating faults.

Wasted Effort ($wef@n$): This denotes the number of elements investigated until locating a fault, but considering only n first positions of the suspiciousness ranking.

Exam: This represents the proportion of investigated elements, in relation to the number of program elements, until finding a fault (i.e. the program size impacts the metric value).

[1]http://deap.readthedocs.io
[2]http://python.org

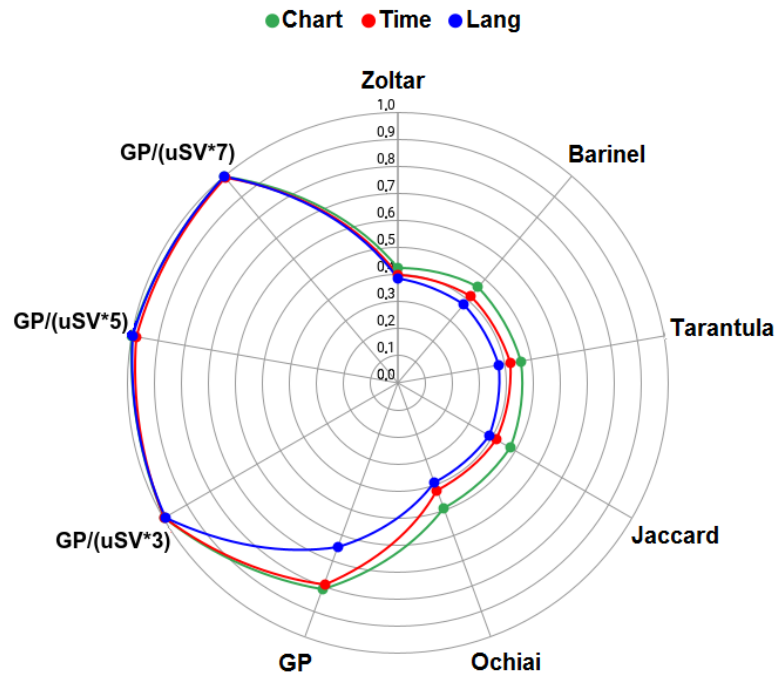

Figure 5. Approaches evaluation: $(uSV/uTSE)$ of *Defects4J* programs (*Chart*, *Time* and *Lang*).

6 RESULTS

Figure 5 extends Figure 3 by including the proposed approach: $GP/(uSV*3)$, $GP/(uSV*5)$ and $GP/(uSV*7)$ training approaches. As expected, all such GP-evolved heuristics improved the value of $(uSV/uTSE)$ as they included uSV maximization to guide the training process. So the proposed approach raises the value of $(uSV/uTSE)$ to close to 1.0, and beats the baseline approach (the GP metaheuristic).

The values computed by the *Exam metric* to the consolidated perspective of *Chart*, *Time* and *Lang* programs are shown in Figure 6. The abscissa axis lists the heuristics in descending order of performance. All three training approaches yield better results than the baseline approach. The values in the chart represent the average with respect to all programs' versions. For instance, in $GP/(uSV*3)$ about 3.6% of program elements are inspected until a fault is located, against 4.2% of the baseline approach.

In Figure 7 we present $wef@n$ (*wasted of effort*) of *Time* program, but considering $n \in \{5, 10\}$ (we omit other programs because of space concerns). The proposed approach again achieves better results related to the other approaches. For example, when inspecting five and ten elements of *Time* program versions, $GP/(uSV*7)$ wasted on average 3.77 and 6.73 elements until locating a fault, respectively (against 3.92 and 6.93 of the baseline approach).

7 STATISTICAL ANALYSIS

Although the results presented in the charts (Figures 5 to 7) seem promising, they need further analysis to increase confidence in the results obtained.

In order to deal with the stochastic effect that is inherent in the evolutionary approaches as well as to raise the findings reliability, two statistical tests were applied: Wilcoxon pair comparison test and Vargha & Delaney Â$_{12}$ test, as recommended by (Arcuri and Briand, 2014). We applied both tests to the results from the evaluation metrics used in the experiment: *Exam* and *Wasted effort* $(wef@n)$.

The Wilcoxon test is used to determine whether there is a statistically significant difference between the results produced by the different methods (p-value less than 0.05). The test has exposed that the three training approaches present results of both evaluation metrics with equivalent significance to the baseline (GP metaheuristic). This finding reveals the competitiveness of the approach.

Table 4. Statistical Analysis - Exam - Varga e Delaney $\hat{A}_{12}$

Program	Approach	Tarantula	Ochiai	Jaccard	GP
Chart	GP/(uSV*3)	0.47	0.51	0.50	0.48
	GP/(uSV*5)	0.60 ⇑	0.62 ⇑	0.62 ⇑	0.61 ⇑
	GP/(uSV*7)	0.57 ⇑	0.60 ⇑	0.60 ⇑	0.59 ⇑
Lang	GP/(uSV*3)	0.49	0.50	0.49	0.51
	GP/(uSV*5)	0.48	0.49	0.48	0.50
	GP/(uSV*7)	0.49	0.50	0.50	0.52
Time	GP/(uSV*3)	0.52	0.52	0.52	0.55
	GP/(uSV*5)	0.51	0.51	0.51	0.54
	GP/(uSV*7)	0.50	0.51	0.51	0.53

Table 4 presents the outcomes of Vargha & Delaney Â$_{12}$ test for the *Exam* metric, with respect to each *Defects4J* program. The symbol ⇑ indicates the cases where the approach is statistically superior to the baseline (the GP metaheuristic) and some human-developed heuristics (*Tarantula*, *Ochiai* and *Jaccard*). The table contents highlight the competitiveness of the proposed approach, and show its superiority for the $GP/(uSV*5)$ and $GP/(uSV*7)$ with respect to all evaluation metrics, when applied to *Chart* program.

Table 5 displays results for *Wasted effort* $(wef@n)$. The study considered scenarios where the faulty element is close

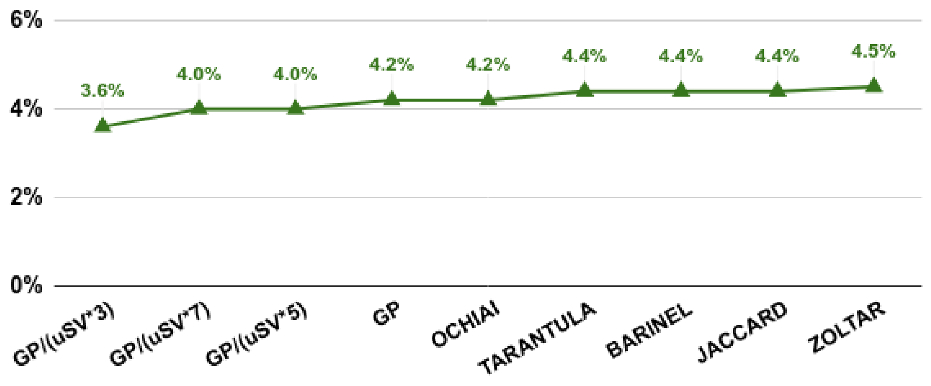

Figure 6. Approaches evaluation: *Exam measure* of *Defects4J* programs (*Chart*, *Time* and *Lang* - consolidated results).

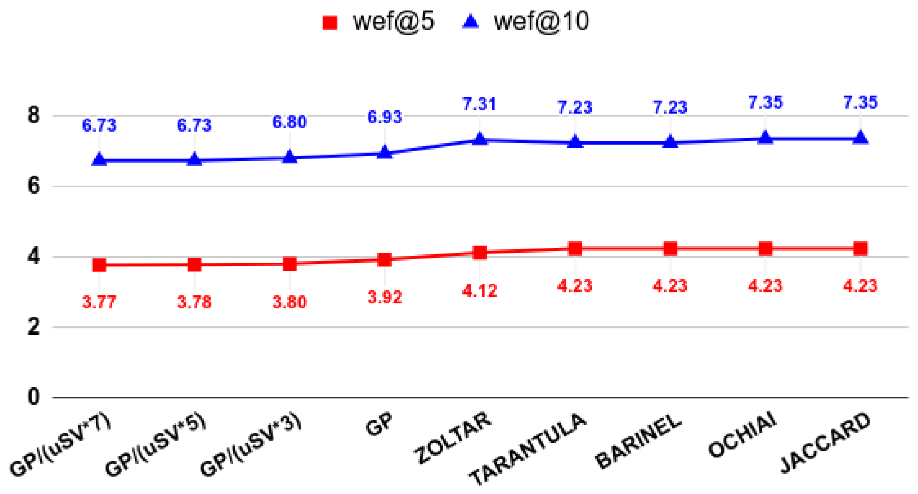

Figure 7. Approaches evaluation: $wef@n$ *measure* of the *Time* program ($n \in \{5, 10\}$).

Table 5. Statistical Analysis - wef@n - Varga e Delaney $\hat{A}_{12}$

Program	wef@n	GP/(uSV*3)	GP/(uSV*5)	GP/(uSV*7)
	$n = 5$	0.50	0.50	0.48
Chart	$n = 10$	0.50	0.50	0.50
	$n = 15$	0.51	0.54	0.50
	$n = 5$	0.50	0.49	0.54
Lang	$n = 10$	0.53	0.50	0.55
	$n = 15$	0.53	0.50	0.54
	$n = 5$	0.55	0.54	0.54
Time	$n = 10$	0.59 $\Uparrow$	0.60 $\Uparrow$	0.57 $\Uparrow$
	$n = 15$	0.58 $\Uparrow$	0.58 $\Uparrow$	0.57 $\Uparrow$

to the top in the suspiciousness ranking, namely $n \in \{5, 10, 15\}$. In all programs, the fault localization ability of the approach was statistically similar to the baseline. However, all training approaches have outperformed the baseline in most scenarios of the *Time* program.

Therefore, considering the discussion presented so far, we can answer the research questions:

- *RQ1: Is the proposed approach competitive for locating software faults?*
  Figures 6 and 7 show better effectiveness of the proposed approach in relation to baseline and human-developed approaches. Competitiveness was confirmed through statistical tests in all scenarios and superiority in several of them.
  Thus the proposed approach is competitive, performing similar or better than the baseline.
- *RQ2: Is the uniqueness of suspiciousness values a factor that guides the generation of effective evolutionary approaches for fault localization?*
  The three training approaches consistently (i.e. in all scenarios) guided competitive (i.e. similar or better) heuristics relative to the baseline, concerning fault locating.
  Thus the uniqueness of suspiciousness values is effective for the guidance of fault localization approaches.

On the hypotheses raised in Subsection 3.4: The analysis of Figure 5 refutes Hypothesis H_0, as we found a correlation between high values of uSV and the improvement of the ability to locate faults; Also, Hypothesis H_1 is then preliminarily confirmed through systematic empirical analysis and statistical tests. So the closer the $uSV\ ratio$ is to the $uTSE\ ratio$, the better for locating faults.

8 THREATS TO VALIDITY

We have adopted strategies to mitigate threats to the validity of the experimental evaluation and its findings. They are listed below and categorized based on (Barros and Dias-Neto, 2011).

Internal threats. To be reproducible, all experiment parameters were explicitly addressed. The open-source DEAP framework was used to implement the GP algorithm and its infrastructure. We use real programs with real faults as Software Engineering is a practical and world-connected science. **Construct threats.** Metrics commonly used in the research field were applied in order to foster the validity of the approaches' effectiveness measures. We adopted worst-case strategies to resolve ties of suspiciousness scores between program elements. **External threats.** To promote confidence and quality of findings, the training instances were carefully selected and studied (Section 3), but the experiment was conducted in real Java software (*Defects4j*), so new instances in other languages can promote greater generalization.**Conclusion threats.** Experiments were run at least 10 times for each instance (program version) in order to deal with stochastic variation and 10-fold cross-validation strategy for generating robust evolutionary GP-evolved heuristics. The analysis used the average of the observed results regarding the many execution cycles and program versions. Evaluations considered a robust benchmark (the canonical GP) as the baseline and several human-made heuristics. Hypothesis and statistical tests were used to show significance to the results.

9 FINAL REMARKS

The present study was based on the analysis of test spectra of *Defects4J* benchmark, which are commonly used as a source of information on existing software faults. For instance, we found out that less than 12% of the test spectra entries have unique values regarding distinguishing program elements. The analysis identified a high sample repetition in the training of fault location GP-evolved heuristics, potentially impacting their ability to locate faults.

Although the repetition of samples in the test spectra is disadvantageous concerning the training process, regarding the evolutionary generation of fault localization heuristics, this motivated us to use the uniqueness of suspiciousness value (scores) as a factor that can boost the fault-finding ability of these heuristics. In other words, we hypothesize that high values of the uniqueness of this score can generate more effective fault localization GP-evolved heuristic.

In this sense, hypotheses were formalized, research questions were stated, a proposal was defined and empirically evaluated, evidence was assessed as to its statistical significance, and threats to validity were addressed. The proposal includes three GP training approaches, all of which seek to promote heuristics that have higher uniqueness of suspiciousness scores.

On the empirical assessment, a state-of-the-art baseline (canonical Genetic Programming metaheuristic) and evaluation metrics widely used in the research field (*Exam* and *Wasted Effort*) were employed in order to reduce findings' threats of validity. The results showed the competitiveness of the training strategies and statistically confirmed that the uniqueness of suspiciousness scores guides superior heuristics for fault localization.

In future work, further experiments are pertinent, including the use of other benchmarks from the research field (Hirsch and Hofer, 2022) and the incorporation of additional data sources (e.g., data flow and mutation spectra). Expanding the dataset will not only validate the robustness of our findings but also uncover any potential limitations. Moreover, leveraging additional data sources could enhance the precision and reliability of suspiciousness scores, leading to more effective fault localization strategies.

Beyond the inclusion of diverse benchmarks and data sources, there is also potential for developing new algorithms that optimize the computation of suspiciousness scores.

References

Abreu, R., Zoeteweij, P., and c. Van Gemund, A. J. (2006). An evaluation of similarity coefficients for software fault localization. In *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*, pages 39–46.

Abreu, R., Zoeteweij, P., and Gemund, A. J. C. v. (2009). Spectrum-based multiple fault localization. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 88–99, Washington, DC, USA. IEEE Computer Society.

Abreu, R., Zoeteweij, P., and van Gemund, A. J. C. (2007). On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)*, pages 89–98.

Arcuri, A. and Briand, L. (2014). A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Softw. Test. Verif. Reliab.*, 24(3):219–250.

Barros, M. d. O. and Dias-Neto, A. C. (2011). 0006/2011 - threats to validity in search-based software engineering empirical studies. *RelaTe-DIA*, 5(1).

Campos, J., Abreu, R., Fraser, G., and d'Amorim, M. (2013). Entropy-based test generation for improved fault localization. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 257–267.

Choi, K., Sohn, J., and Yoo, S. (2018). Learning fault localisation for both humans and machines using multi-objective gp. In Colanzi, T. E. and McMinn, P., editors, *Search-Based Software Engineering*, pages 349–355, Cham. Springer International Publishing.

De-Freitas, D. M., Leitao-Junior, P. S., Camilo-Junior, C. G., and Harrison, R. (2018). Mutation-Based Evolutionary Fault Localisation. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8.

Hailpern, B. and Santhanam, P. (2002). Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12.

Hirsch, T. and Hofer, B. (2022). A systematic literature review on benchmarks for evaluating debugging approaches. *Journal of Systems and Software*, 192. Cited by: 0; All Open Access, Hybrid Gold Open Access.

Janssen, T., Abreu, R., and van Gemund, A. J. (2009). Zoltar:

A spectrum-based fault localization tool. In *Proceedings of the 2009 ESEC/FSE Workshop on Software Integration and Evolution @ Runtime*, SINTER '09, pages 23–30, New York, NY, USA. ACM.

Jones, J. A., Harrold, M. J., and Stasko, J. T. (2009). Visualization for Fault Localization. In *in Proceedings of ICSE 2001 Workshop on Software Visualization*, pages 71–75.

Just, R., Jalali, D., and Ernst, M. D. (2014). Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, page 437–440, New York, NY, USA. Association for Computing Machinery.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Leitao-Junior, P. S., Freitas, D. M., Vergilio, S. R., Camilo-Junior, C. G., and Harrison, R. (2020). Search-based fault localisation: A systematic mapping study. *Information and Software Technology*, 123:106295.

Lu, S., Li, Z., Qin, F., Tan, L., Zhou, P., and Zhou, Y. (2005). Bugbench: Benchmarks for evaluating bug detection tools. In *In Workshop on the Evaluation of Software Defect Detection Tools*.

Pearson, S., Campos, J., Just, R., Fraser, G., Abreu, R., Ernst, M. D., Pang, D., and Keller, B. (2017). Evaluating and improving fault localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 609–620.

Silva-Junior, D., Leitao-Junior, P., Dantas, A., Camilo-Junior, C., and Harrison, R. (2020). Data-flow-based evolutionary fault localization. In *Proceedings of the ACM Symposium on Applied Computing*, pages 1963–1970.

Sohn, J. and Yoo, S. (2017). FLUCCS: Using Code and Change Metrics to Improve Fault Localization. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2017, pages 273–283, New York, NY, USA. Association for Computing Machinery. event-place: Santa Barbara, CA, USA.

Wang, S., Lo, D., Jiang, L., Lucia, and Lau, H. (2011). Search-based fault localization. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings*, pages 556–559.

Wong, W., Gao, R., Li, Y., Abreu, R., and Wotawa, F. (2016). A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740.

Yoo, S. (2012). Evolving human competitive spectra-based fault localisation techniques. In Fraser, G. and Teixeira de Souza, J., editors, *Search Based Software Engineering*, pages 244–258, Berlin, Heidelberg. Springer Berlin Heidelberg.

Yoo, S., Xie, X., Kuo, F.-C., Chen, T. Y., and Harman, M. (2017). Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis. *ACM Trans. Softw. Eng. Methodol.*, 26(1).

Zheng, Y., Wang, Z., Fan, X., Chen, X., and Yang, Z. (2018). Localizing multiple software faults based on evolution algorithm. *Journal of Systems and Software*, 139:107 – 123.