Data-Flow-Based Evolutionary Fault Localization

Silva-Junior D, Leitao-Junior P, Dantas A, Camilo-Junior C, Harrison R

ABSTRACT

Fault localization is the activity of precisely indicating the faulty commands in a buggy program. It is known to be a highly costly and tedious process. Automating this process has been the goal of many studies, showing it to be a challenging problem. The coverage-spectrum based approaches commonly apply heuristics grounded on the execution of control-flow components to calculate the odds of each program element to be the defective one. The present study aims to investigate another source of fault information by assessing how data-flow analysis are useful to compute suspiciousness scores; and how the combination of scores from different sources impacts fault localization. We present an approach to calculate the suspiciousness score for each program command by using the execution of data-flow components. Then we use an evolutionary algorithm to search sets of weights to combine heuristics from distinct sources of fault data (both control-flow and data-flow as well as a hybrid strategy). The approach was applied in programs with seeded faults and real faults and evaluated by using absolute metrics to asses its efficacy to locate faults. Furthermore, we introduce a new metric to investigate the dependence of tie-break strategies in building the ranking of suspicious commands. Data-flow based methods demonstrate high effectiveness but increase the need for tie-breaks, unlike the evolutionary hybrid method that keeps competitive the effectiveness and depends less on tie-break strategies.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**; **Software testing and debugging**;

KEYWORDS

Fault Localization, SBSE, Software Debugging, Data-Flow Analysis

ACM Reference Format:

. 2020. Data-Flow-Based Evolutionary Fault Localization. In *Proceedings of ACM SAC Conference (SAC'20).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Software is progressively adopted to manage critical tasks, such as health and security systems. On the other hand, it is almost impossible to avoid the insertion of bugs. Such defects have the intrinsic potential to generate damages, to resources or humans life [10]. Software testing and debugging are topics that concentrate efforts on ensuring reliability in software artifacts, preventing, detecting, and repairing faults [4].

Within the software debugging process, Fault Localization (FL) is the process of precisely indicating the faulty element in the defective code. Such activity is known to be too costly and monotonous. In this context, several approaches seek to automate this task. The most popular methods are FL heuristics that utilize the information obtained from the coverage spectrum, to compute suspiciousness values to each program element[25]. It is possible to obtain coverage spectrum from different sources of fault information, for example, by analyzing the execution of control-flow or data-flow elements.

The current study aims to analyze if the suspiciousness score obtained from data-flow can contribute positively to the location of faults. We also present an evolutionary approach to combine suspiciousness scores calculated from different sources of information as control- and data-flow coverage spectra. We explore a set of 24 suspiciousness values, using 12 heuristics with each of the two coverage spectra. Thus, we use a genetic algorithm to search sets of weights that when applied to the suspiciousness values, generate linear combinations of the known heuristics. The method was evaluated in 112 faulty versions obtained from the *Siemens Suite* programs, and 36 faulty versions from *jsoup* program. We use *Accuracy* and *Wasted Effort* to assess the method, we also introduce a *Absolute Critical Tie* metric to investigate the number of ties generated by FL methods.

Heuristics using data-flow information outperform other methods with respect to accuracy, but a hybrid approach using control- and data-flow demonstrates a comparable accuracy and a lower number of ties compared with them, giving a lower wasted effort.

In Section 3, we present our proposals and useful definitions to understand the approach better. Section 4 presents baselines, metrics, genetic algorithm parameters, and the experimentation process. The research questions are treated in Section 5. Finally, Sections 6 and 7, report potential validity threats and our conclusions.

2 BACKGROUND AND RELATED WORK

The main proposals for automating FL task are based on the execution of test cases to infer the probability of being responsible for the incorrect behavior for each program command. The program spectrum is a set of data that expresses its execution behavior. Then, it allows the visualization of which program elements are executed (covered) by the test cases and how they are related to the defect.

The coverage spectrum is defined as a matrix $M \times N$, derived from M executions that can cover N program commands, and a separate column represents the execution success or failure information. From the data of this coverage matrix, it is possible to measure a suspiciousness score for any element N_i [3], as shown in Figure 1.

Traditionally, the spectrum coverage is shown with control-flow program elements, as nodes (commands or block of commands), and edges (decisions), this control-flow coverage spectrum may express which commands were executed by the test cases, providing useful information for FL.

SAC'20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/xx.xxx/xxx_x

$$
\begin{array}{c} \\ M \text{ executions} \end{array}
\begin{array}{c} N \text{ Program Elements} \\ \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \end{array}
\quad
\begin{array}{c} \text{Success/Fail} \\ \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \end{array}
$$

Figure 1: Example of a spectrum coverage matrix.

Coverage spectrum-based heuristics are equations that use variable values obtained from the coverage matrix to give program elements a suspiciousness value. We will use the following notation to indicate these coverage variables that one can obtain for each N_i element, (i) *es* - Number of successful test cases that execute (cover) the element under observation; (ii) *ns* - Number of successful test cases that do not execute (cover) the element under observation; (iii) ef - Number of failed test cases that execute (cover) the element under observation, and (iv) nf - Number of failed test cases that do not execute (cover) the element under observation.

The *Tarantula* heuristic [14] was one of the first coverage spectrum-based FL techniques presented. A node is a control-flow element, and one can use the *Tarantula* heuristic to calculate the suspiciousness score to a given node of the investigated program, as presented in Equation 1.

$$\text{Tarantula(node)} = \frac{\frac{ef(node)}{ef(node)+nf(node)}}{\frac{ef(node)}{ef(node)+nf(node)} + \frac{es(node)}{es(node)+ns(node)}} \quad (1)$$

The results obtained by heuristics can be used to guide the localization of bugs. Thus, several studies such as Ochiai [1], Dstar [26], and OP2 [16] present other heuristics aiming a better accuracy.

Software fault localization techniques face several difficulties. In addition to highlighting the defective command, they should also avoid giving high values of suspiciousness score to non-defective commands.

2.1 Search-Based Fault Localization

Search-Based Software Engineering (SBSE) techniques attack software engineering problems by reducing them to optimization problems, which can, in turn, be addressed by a variety of known, populational or local, search strategies. Harman *et al.* [11] highlight two essential aspects when applying search strategies to software engineering problems, namely, *problem representation* and the *fitness function* to evaluate possible solutions.

Wang *et al.* [23] use SBSE for the FL problem. This study represents a solution as a set of weights that should be applied to traditional heuristics, generating a linear combination of suspiciousness score values. To evaluate the solutions found by the search methods, Wang *et al.* define a *fitness function* as the average code proportion that someone should investigate whilst finding the faulty elements, the goal is to minimize this proportion using Simulating Annealing (SA) and Genetic Algorithm (GA).

Yoo *et al.* [29] and De-Freitas *et al.* [7] applied genetic programming (GP) strategies to the FL problem. The former uses GP to generate new equations using the same set of variables used in heuristics described in Section 2, and the solutions are trained with a set of projects. The latter generates equations with information based on mutation tests using different versions of the same programs to train the solutions.

Search-based fault localization methods show promising results, outperforming several traditional heuristics such as *Ochiai* and *Tarantula* when analyzed for effectiveness.

2.2 Data-Flow Analysis for Fault Localization

Data-flow analysis [12] has been used to define testing criteria, and refers to the analysis of dynamic interactions between a memory definition (*e.g.* change a value in a memory address) and subsequent uses of such a definition during the program execution.

In program code, a definition (*def*) happens when a command changes the value of a variable, and a "use" happens when that variable is accessed. Two types of use are distinguished: *c-use* (computational use), when the variable is used to compute a value; and *p-use* (predicate use), when the variable is used to compute a predicate. So *c-uses* happen on nodes and *p-uses* on edges, as we can see in Figure 2.

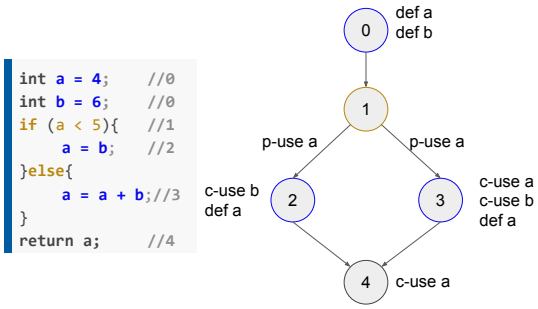


Figure 2: Data-flow example.

A *def-use association* (*dua*) with respect to (*w.r.t.*) a variable x occurs when there exists at least one path free of definition (*def-clear path*) from a definition of x to a subsequent use of x. One can represent a *dua* as the triple $< variable, def\_site, use\_site >$, where *variable* is defined in node def_site, and use_site refers both to a node where a *c-use* of *variable* occurs or an edge of a *p-use* of *variable*. For instance from Figure 2, we derive the *duas* $< a, 0, (1-2) >$, $< a, 0, (1-3) >$, $< b, 0, 2 >$, $< a, 0, 3 >$, $< b, 0, 3 >$, $< a, 2, 4 >$ and $< a, 3, 4 >$.

In data-flow analysis, a test case covers a *dua* if at least one clear path w.r.t. the variable is exercised during its execution. The purpose of data-flow criteria is to exercise def-use associations in various ways. Some examples of the data-flow criteria are:

- *all-defs*: requires the coverage of at least one of the *duas* derived from each variable definition.
- *all-uses*: requires the coverage of all the *duas* derived from each variable definition.

Data-flow coverage spectra are derived from the test cases execution, making available the following variables related to a specific *dua*: $ef(dua)$ and $nf(dua)$ are the number of failing test cases that covered and did not cover the *dua* respectively; $es(dua)$ and $ns(dua)$

are the number of sucessful test cases that covered and did not cover the *dua* respectively.

Data-flow based approaches share the belief data-flow analysis may improve the effectiveness to locate faults and obtain good results when compared to pure control-flow analysis. In this sense the suspiciousness scores of *duas* have been similarly computed as the original proposals to the control-flow coverage spectra [20–22]. Equation 2 presents the *Tarantula* heuristic applied to a *dua*.

$$Tarantula(dua) = \frac{\frac{ef(dua)}{ef(dua)+nf(dua)}}{\frac{ef(dua)}{ef(dua)+nf(dua)} + \frac{es(dua)}{es(dua)+ns(dua)}} \quad (2)$$

Ribeiro *et al.* [20, 21] propose JAGUAR, a tool to assist the debugging expert visually. The search for the faulty program element is guided by the suspiciousness ranking of *duas*: for each investigated *dua*, it considers that two elements were inspected to compute how many elements were checked until finding the faulty one. The analysis was applied in ten FL heuristics such as *Tarantula* and *Ochiai*, among others. The results have shown that data-flow coverage leads the debugging expert to inspect less code than control-flow with statistical significance for all heuristics experimented.

Santelices *et al.* [22] apply data-flow analysis to locate faults and rank *duas* based on *Tarantula*. However, they consider the definition part of data-flow associations as the site of the faulty element. The approach also employs three different sources of fault (nodes, branches, and *duas*) to obtain three suspiciousness scores for each program node. They conclude that no fault source excels the others for fault localization, and propose the combination of them such as calculating for each program node the highest and average values related to the suspiciousness scores.

Regarding the two aforementioned approaches, the first one does not calculate suspiciousness scores for program elements but just builds a ranking of *duas* to guide the inspection of elements. The latter does not consider the "def" and "use" parts of data-flow associations equally to locate the potential fault site. Our proposal includes an evolutionary method and copes with such gaps as well as encompassing some benefits of these approaches.

3 APPROACH

Heuristics based on control-flow spectra commonly measure the intrinsic suspiciousness score for each node in a program. However, on applying these heuristics to data-flow associations, a pertinent question is: how to set suspiciousness scores to nodes from the scores of *duas*. Moreover, the same node can participate in several *duas*, and there may still be different suspiciousness scores in each of these *duas*. Therefore, the suspiciousness of *duas* is not trivially comparable with the traditional suspiciousness score of nodes.

A data-flow association is defined in Section 2.2 as the triple $< variable, def\_site, use\_site >$: def_site refers to the node where $variable$ is defined, and use_site refers both to a node where a *c-use* of $variable$ or an edge of a *p-use* of $variable$ occurs.

In our proposal, the nodes related to the definition site and the use site in a data-flow association should inherit the suspiciousness score of the *dua*. As the same node can participate in several *duas*, and there may still be different suspiciousness scores in each of these *duas*, the suspiciousness score of a node is the highest score among the *duas* in which the node participates. In the following this is formalized as *dua-to-node score*, the algorithm for calculating the suspiciousness score of a node:

Definition 1 - **allduas(node) set**. This set contains all *duas* that a specific node is in.

$$) = \bigcup dua_i \quad | \, node \in dua_i \quad (3)$$

Definition 2 - **dua-to-node score** ($\widehat{H}(node)$). This score denotes the suspiciousness of a *node*, obtained from the data flow analysis of all *duas* in *allduas(node)*.

$$\widehat{H}(node) = max\big(H(dua_i)\big) \quad | \, dua_i \in alldua(node) \quad (4)$$

For instance, suppose that *dua_1* is the triple *<var_1, node_10, (node_11–node_12)>* and *dua_2* is *<var_2, node_5, (node_11–node_13)>*. Also let the suspiciousness scores be *OP2(dua_1) = 0.55* and *OP2(dua_2) = 0.9*. Based on these data $\widehat{OP2}$ of nodes *node_5*, *node_10*, *node_11*, *node_12* and *node_13* are 0.9, 0.55, 0.9, 0.55 and 0.9 respectively.

3.1 Evolutionary Fault Localization Based on Data-Flow Coverage Spectra

Wang *et al.*[23] explore the combination of control-flow suspiciousness scores using a genetic algorithm, modeling the FL problem as an optimization problem.

A genetic algorithm (GA) is a bio-inspired meta-heuristic used for optimization problems. It has the main characteristic of allowing the "evolution" of a population of solutions. To achieve this, it use operators such as *crossover* to combine solutions, and *mutation* to reach new solutions from other already evaluated [19].

In their approach, Wang *et al.* try to find a linear combination of n heuristics, and assign the obtained value of suspiciousness to the investigated nodes. The applied strategy is to use search weights w_i for each heuristic H_i. This way, the method has as a solution a weighted sum HC of the suspiciousness score obtained from n heuristics, as we can see in Equation 5.

$$HC(node) = w_1 \times H_1(node) + w_2 \times H_2(node) + \ldots + w_n \times H_n(node) \quad (5)$$

The solutions are represented as a binary string with seven bits reserved for each weight w_i, with the domain $0.0 \leq w_i \leq 1.0$. The fitness of each possible solution is evaluated calculating the average proportion of code investigated until finding the fault site; also, it is a metric related to the number of faulty versions investigated.

Similarly, we can investigate the linear combination by applying in control- and data-flow suspiciousness scores. For all heuristics H_i, we calculate the control-flow ($H_i(node)$) and data-flow ($H_i\_ndua(node)$)) suspiciousness for each node. Thus, we can generate three different compositions: a simple combination using only the control-flow suspiciousness score, a simple combination using only the data-flow suspiciousness score, and a hybrid combination using both sources of data, as shown in Equation 6.

$$\begin{aligned} HC_{hyb}(node) = {} & w_1 \times H_1(node) + \ldots + w_n \times H_n(node) + \\ & w_{n+1} \times \widehat{H}_1(node) + \ldots + w_{n+m} \times \widehat{H}_n(node) \end{aligned} \quad (6)$$

We use a genetic algorithm to generate each combination, using several heuristics from the literature, and evaluate the approach

using absolute metrics commonly used in similar studies. We hypothesize that the linear combination of control-flow and data-flow information improves the effectiveness of fault localization.

4 EXPERIMENTS

We conduct an empirical experiment aiming to investigate the following Research Questions:
RQ1: Is the proposed method competitive for locating software faults?
RQ2: Does the evolutionary combination of control-flow and data-flow coverage spectra improve fault localization ability?

To answer these questions, we analyze how node-to-dua scores perform in comparison to traditional methods with respect to effectiveness, and if the evolutionary approach offers improvements to fault localization when combining the different sources of data.

4.1 *Baselines*

As baselines we use 12 well known heuristics, that are commonly used in recent FL studies, they are: *Tarantula*, *Ochiai*, *OP*, *Dstar*, *Ample* , *Jaccard* , *GP*13 , *OP*2, *Wong*3 , *Zoltar* , *Kulczynski*2, e *Barinel* [2, 14, 15, 25, 27, 29]. Table 1 shows the heuristics used in the experiment. For each heuristic, we calculate the suspiciousness score for each node using control-flow and data-flow elements. Additionally, we have used the approach described by Wang *et al.*[23] as an evolutionary baseline.

4.2 Experimentation Process and Genetic Algorithm

We use a genetic algorithm to search weights for a linear combination of suspiciousness values. The utilized method of search takes advantage of the known faulty command of some buggy versions using such information to train candidate solutions. So, it builds new compositions that also perform well in different faulty versions, *i.e.*, the possible solutions are trained to locate faults better using the localization of known faults.

When we are training new solutions with a set of faults, achieved solutions may become so well fitted to the training set, that their result is not generalizable anymore, such a behavior is called *Overfitting*. Besides that, the genetic algorithm has an intrinsic stochastic nature, *i.e.*, it can result in a different solution in each execution.

To deal with *Overfitting*, we conducted the experiments by applying a *Three-Fold Cross Validation*. We randomly divided the faulty versions into three disjointed sets. In each step of cross-validation, one set is selected to be the validation set, whilst the remaining sets compose the training set. Consequently, we do not evaluate a solution by the same information that was used to generate itself.

To ease the GA stochasticity effects, we performed 30 executions of the cross-validation process and applied statistical tests on resulting samples. The results in the next sections refer to the validation set executions, that do not exert influence in the search.

The individual genotype is a binary string with seven bits for each heuristic suspiciousness score, that is, 84 bits for simple compositions and 168 bits for hybrid compositions. We adopted a *bitflip* mutation operator (rate 0.01), two-points crossover (rate 0.6), and tournament 3 selection strategy. The *fitness* function is also the same used by Wang *et al.*[23], the average proportion of code investigated before finding the faults in the validation set. The GAs were executed with 250 generations, 50 individuals as population size.

We used DEAP (*Distributed Evolutionary Algorithms in Python*) [8] to implement the GA and the R Project [18] for statistical analysis.

4.3 Subject Programs

As Table 2 shows, the experiment was conducted using seven C programs from the *Siemens Suite* [13], obtained from the SIR - *Software-artifact Infrastructure Repository*[9] and one Java program called *jsoup*[1] with real faults. We utilized 112 versions with seeded faults from the C programs and 36 versions from the java one. We obtained the control and data-flow coverage spectra from the *Siemens* programs through manual code instrumentation, and the *jsoup* data were collected from the JAGUAR tool[20].

5 RESULTS

We use two absolute metrics to evaluate the effectiveness of the proposal: *Accuracy* and *Wasted Effort* [17], which are also used in other FL experiments [6, 7, 24].

Accuracy (acc@n) counts the number of faults located looking at the top n elements in the suspiciousness ranking. This metric represents a debugging expert inspecting a limited number of elements with the highest suspiciousness score. Higher values are better in this metric. We use the average case strategy for the tie-break, *i.e.*, if two or more program elements have the same suspiciousness score, we consider the average position in the ranking.

Wasted effort (wef@n) counts the number of elements investigated before locating the faults, limited to n elements for each faulty version. This metric represents the non-defective elements inspected by a debugging expert whilst locating the first fault or stopping. Smaller values are better in this metric. We also use the average case strategy for the tie-break.

In the following discussion we present *acc* and *wef* results (only *@5* because of space concerns) of the traditional heuristics $H(node)$ represented by "control-flow" and dua-to-node scores $\widehat{H}(node)$ represented by "data-flow". We also present GAs with simple combinations for each source of information (GA-cf and GA-df) and the "*GA_hyb*" representing the genetic algorithm using data obtained from both data-flow and control-flow coverage spectra.

Figure 3a shows the *acc@5* results for 112 faulty versions of the *Siemens Suite* programs. In this chart, as in the following ones, the X-axis is FL methods, and Y-axis is the metric value, *acc@5* in this case. For each method, we have two bars of results, the left one shows the results using control-flow information, and the right one shows the data-flow results. The one bar in *GA_hyb* shows the results combining both information.

Looking at the *acc@5* results on *Siemens Suite*. It is possible to see the apparent slight predominance of dua-to-node scores compared with traditional methods. Heuristics using data-flow information were able to locate up to 35 faults against 33 of the control-flow based suspiciousness score. GA-cf performed better than the heuristics using only control-flow with 34 faults located.

[1] https://github.com/jhy/jsoup

Table 1: Heuristics used as baseline and combined using evolutionary approach

Heuristic	Name	Equation	Heuristic	Name	Equation	Heuristic	Name	Equation
H_1	Tarantula	$\frac{\frac{ef}{ef+nf}}{\frac{ef}{ef+nf}+\frac{es}{es+ns}}$	H_5	Ample	$\left\lvert\frac{ef}{ef+nf}-\frac{es}{es+ns}\right\rvert$	H_9	Wong3	$\begin{cases} es \le 2 & ef-es \\ es > 2 \ \& \ es \le 10 & ef-2+0.1\times(es-2) \\ es < 10 & ef-2.8+0.01\times(es-10) \end{cases}$
H_2	Ochiai	$\frac{ef}{\sqrt{ef+nf\times(ef+es)}}$	H_6	Jaccard	$\frac{ef}{ef+nf+es}$	H_{10}	Zoltar	$\frac{ef}{ef+nf+es+\frac{10000\times nf\times es}{ef}}$
H_3	OP	$\begin{cases} nf > 0 & -1 \\ nf \le 0 & ns \end{cases}$	H_7	GP13	$ef\times(1+\frac{1}{2\times es+ef})$	H_{11}	Kulczynski2	$\frac{1}{2}\times(\frac{ef}{ef+nf}+\frac{ef}{ef+es})$
H_4	Dstar	$\frac{ef^2}{nf+es}$	H_8	OP2	$\frac{ef-es}{es+ns+1}$	H_{12}	Barinel	$1-\frac{es}{es+ef}$

Table 2: Subject programs of the experiments

Program	LOC	Buggy Versions	Test Cases
printtokens	472	6	4030
printtokens2	399	9	4415
replace	512	26	5542
schedule	292	8	2650
schedule2	301	7	2710
tcas	141	35	1608
tot_info	440	21	1051
jsoup	10K	36	468

Similarly, GA-df performed better than the heuristics using only data-flow with 38 faults located. Finally, *GA_hyb* outperfoms all other methods analyzed, locating 39 faults inspecting only the top-5 elements of the suspiciousness ranking.

Figure 3b reports *acc@5* results with respect to the 36 faulty versions of *jsoup*. As well as in *Siemens Suite*, heuristics with dua-to-node scores also outperform control-flow based methods, locating up to 14 faults against 11 of the same heuristics using control-flow information. GA, in its turn, performs better than heuristics when comparing using only dua-to-node scores, locating 15 faults, but does not improve fault localization effectiveness using only control-flow information, locating just eight faults. The hybrid approach does not find *acc@5* better than GA-df, but maintains the results of dua-to-node heuristics with 14 faults located.

Concerning *acc@5*, methods using data-flow information produce results significantly better than when using control-flow. We conjecture that by containing information about relations of program variables definition and use, the data-flow allows a more careful investigation than control-flow, that uses only commands executed in the program — implying a more accurate suspiciousness score for the faulty element.

Figure 4a presents *wef@5* results with respect to *Siemens Suite* programs. In this metric lower results are better. So, in general, methods using control-flow information perform slightly better than the same methods using data-flow. For instance, the *OP2* heuristic would require the inspection of 440 elements using control-flow against 455 using data-flow. The GA-cf presents an almost irrelevant improvement in comparison with heuristics by reducing just one element inspected, GA-df reduces ten inspected elements in comparison with heuristics. *GA_hyb* shows the biggest improvement, reducing to 416 the number of inspected elements, outperforming all other analyzed methods.

Figure 4b illustrates *wef@5* results related to the *jsoup* program. Unlike the *Siemens Suite* results, for *jsoup*, methods using data-flow information perform better in comparison with the same method using control-flow. For example, the heuristic Ample-data-flow inpected 147 elements, against 156 of Ample-control-flow. The GA-cf produced a worse result than the heuristic, and GA-df again shows an almost irrelevant improvement. The *GA_hyb* keeps the advantage presented in *Siemens Suite*, with 140 elements inspected, also outperforming all other analyzed methods.

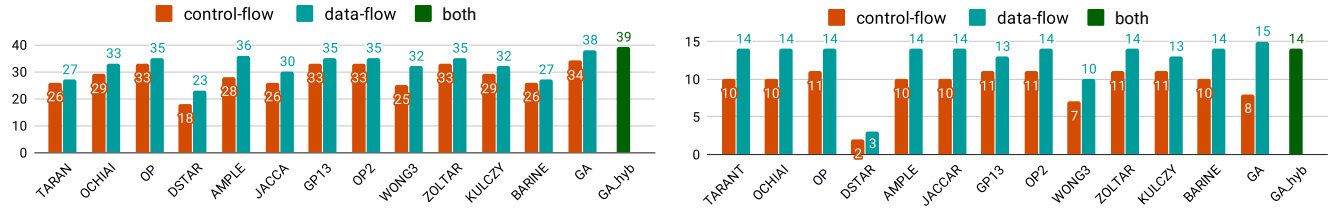
The *wef* results are lower when the number of faults located (*acc*) is higher. This behavior may explain why data-flow methods have lower *wef* than control-flow in *jsoup*. However, in *Siemens Suite*, this is apparently not true. So, to clarify this interpretation it is important to point out that the dua-to-node score has the intrinsic characteristic of attributting the same suspiciousness score to all nodes that participate in a *dua*. So, in some cases when heuristics using control-flow address the highest suspiciousness score to a node, using data-flow it may share the same score with another three elements. This way, in this case, the *wef@5* that would be 1 analyzing control-flow, becomes 3 (in the worst case). This scenario probably happens in *Siemens Suite* and *jsoup*, but in the first one, the slight difference of *acc@5* does not compensate the *wef@5* for the additional inspected elements.

5.1 Analysis

From the *acc* results presented earlier, the ability to locate faults presented by methods using data-flow information is notable. In *Siemens Suite*, and *jsoup* the dua-to-node score makes traditional heuristics achieve results that without it were impossible to obtain. The GA approach also has produced more significant improvements when applied using data-flow information. Although the hybrid approach demonstrates a slight advantage for the *Siemens Suite*, we do not notice this behavior in *jsoup*, where it only maintains comparable results.

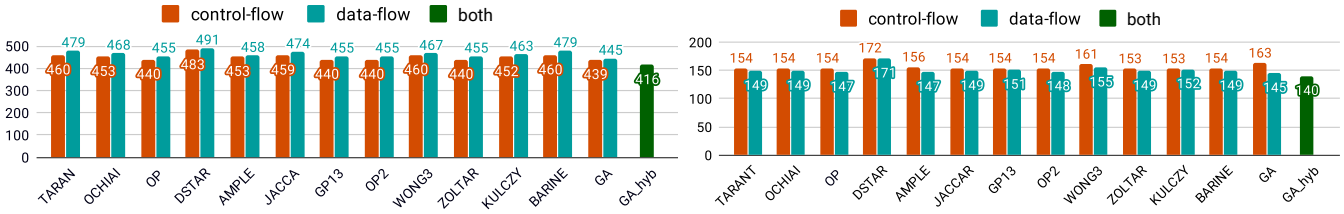
On the other hand, *GA_hyb* expresses a considerable contribution when analyzing the *wef* results, so, the hybrid approach presents an *acc@5* value which is at least as good as the heuristics using data-flow, but would require less inspected code to locate the same number of faults. In our investigation, we detected that this behavior is also valid for *n* not equal to 5, (n = 3 or 10, say).

As aforementioned, the dua-to-node score has as an intrinsic behavior, to address the same suspiciousness score for many elements. We conduct a tie investigation aiming to understand this phenomenon deeply.



(a) *Siemens Suite.* **(b) *jsoup.***

Figure 3: Accuracy (*acc@5*) results.



(a) *Siemens Suite.* **(b) *jsoup.***

Figure 4: Wasted effort (*wef@5*) results.

Regarding the faulty suspiciousness of program elements, Xu *et al.* [28] defines a *tie* as a set of statements, each of which has been assigned the same suspiciousness score, and therefore shares the same position in the suspiciousness ranking. Due to that, a faulty statement may be tied with non-faulty statements. In this sense, the authors also define a *critical tie* as a tie that contains a faulty statement, and the statements in a critical tie are called *critically tied statements*.

A critical tie impacts the suspiciousness ranking as it aggregates uncertainty on calculating a suspicious score of the faulty program elements. Then the higher the number of ties that involve faulty statements, the harder it is to precisely estimate at what ranking position during the examination the faulty statement will be encountered *et al.*. Thus reducing the number of critical ties improves precision when locating faults.

To contribute to the analysis of FL techniques, we define a new evaluation metric called *Absolute Critical Tie* (*actie@n*) that uses the number of critical ties in the top-n elements of suspiciousness rankings. One may interpret the metric as the potential effort wasted with *critical ties*, *i.e.* the number of non-faulty program elements tied with faulty ones in the top-n ranked elements. Higher values indicate more dependencies on tie-break strategies and lower precision when locating software faults.

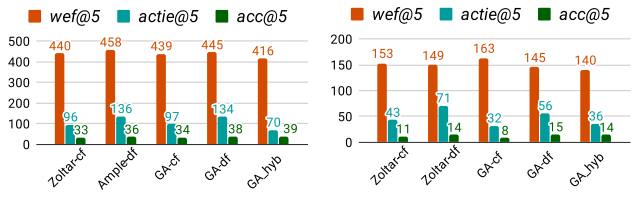
For discussion, let us consider a suspiciousness ranking with two program elements critically tied, occupying the third position (in the worst case). Inspecting the first three elements of the ranking, two of them have the same suspiciousness score and one of them is the faulty element, so, applying *actie@3* in this ranking, the result is 2. In the same example, the *actie@2* is 1, because inspecting two program elements of the ranking, one of them shares the suspiciousness score with the faulty element.

actie does not express the effectiveness of FL methods. It is an auxiliary metric and is should be used together with the other metrics. For example, if we analyze *actie* and *wef* together, it is possible to figure out how much of the wasted effort was used inspecting elements that are critical ties.

Figure 5 shows a summary of results to *acc* *wef* and *actie*. the best heuristics in *accuracy* were considered to represent all other heuristics in these comparisons, they are Zoltar-cf and Ample-df in *Siemens Suite*. In *jsoup* we choose Zoltar-cf and Zoltar-df.

Considering *Siemens Suite* (Figure 5a), the heuristic and GA based only in control-flow information present very similar *actie* results 96 and 97 respectively. We note the same behavior in methods exclusively based on data-flow information, 136 for heuristic, and 134 for GA. However, GA_hyb generated a value of 70 for the critical ties, which is considerably lower than the other methods.

In *jsoup* (Figure 5b), *actie* for data-flow methods is higher than control-flow methods. GA-df demonstrates some improvement to Ample-df, reducing from 71 critical ties to 56. GA_hyb also demonstrates a reduced value of critical ties for *jsoup*.



(a) *Siemens Suite.* **(b) *jsoup.***

Figure 5: Results for Absolute critical tie (*actie@5*) in relation to Wasted effort (*wef@5*) and Accuracy (*acc@5*).

As expected for data-flow based methods, the number of critical ties is higher than the control-flow based methods. GA_hyb results show that it can reduce the need for tie-breaks strategies drastically, but it maintains the satisfactory effectiveness results provided by data-flow information, 14 faults located in *jsoup* and 39 in *Siemens Suite*. This improvement is also reflected in *wef* values, which are the smallest for both benchmarks.

5.2 Statistical Tests Analysis

To deal with the GA stochasticity and improves confidence in our findings, we performed two statistical tests on the results from 30 runs: Wilcoxon pairwise comparison and Vargha & Delaney $\hat{A}_{12}$ tests, as recommended in [5]. The former is to reveal whether there is a significant statistical difference between the results produced by different algorithms. The later provides how much superior the results from an algorithm are against results from another one.

Table 3 shows the results of the statistical tests by comparing the values achieved by all the methods we have used - GA_hyb, GA-cf, GA-df, H-cf (Heuristic with control-flow,), and H-df (heuristic with data-flow). We reported results from the metrics *acc*, *wef*, and *actie*. H-cf and H-df were chosen to represent all heuristics, as earlier we considered the heuristic that demonstrates better results in each metric except for *actie*, which does not express quality alone.

In *jsoup*, for *acc* and *actie* analysis, we selected Zoltar-cf and Zoltar-df, and Zoltar-cf and Ample-df for *wef*. In *Siemens Suite*, for *acc* analysis, we selected Zoltar-cf and Ample-df, Zoltar-cf and Zoltar-df for *wef* and *actie*.

We did not perform a comparison between H-cf and H-df as they are deterministic strategies. The values reported in the Table are only for Varha & Delaney $\hat{A}_{12}$, in which 1 is the method at the line, and 2 is the method at the column. bold-face value indicates that there was no statistically significant difference, considering a confidence level of 99%.

Table 3: Results from the Vargha & Delaney $\hat{A}_{12}$ test by considering all the FL methods and all metrics

		Siemens Suite			jsoup		
		GA_hyb	GA-cf	GA-df	GA_hyb	GA-cf	GA-df
GA-cf	acc@5	0.00			0.00		
	wef@5	1.00			1.00		
	actie@5	1.00			0.14		
GA-df	acc@5	0.15	1.00		0.82	1.00	
	wef@5	1.00	0.99		0.88	0.00	
	actie@5	1.00	1.00		1.00	1.00	
H-cf	acc@5	0.00	0.13	0.00	0.00	0.98	0.00
	wef@5	1.00	0.80	0.00	1.00	0.00	1.00
	actie@5	1.00	0.18	0.00	1.00	1.00	0.00
H-df	acc@5	0.00	1.00	0.00	**0.53**	1.00	0.03
	wef@5	1.00	1.00	1.00	1.00	0.00	0.83
	actie@5	1.00	1.00	1.00	1.00	1.00	1.00

In the case of the Wilcoxon test, its output is the *p-value*, which gives the percentage chance of two samples being statistically equal, instead of different. In our tests, in all comparisons but one (boldface), there was a statistical difference between the results produced by every FL method taking into account all metrics. This finding means that, even when the graphics showed close results, there was a difference between the methods; thus, we were able to make comparisons and get robust conclusions.

Regarding the $\hat{A}_{12}$ test, its output value informs how frequently superior are the values from method 1 compared to the values from method 2. Thus, the closer to 1.0 the output of the test, the higher the chance of the value from method 1 outperforms value from method 2. On the other hand, the closer to 0.0 the output of the test, the higher the chance of the value from method 1 is inferior to the value from method 2. For example, looking at GA-cf versus GA-hyb, in the metric acc@5, the statistical test estimated 0.0, i.,e, GA-hyb is always better than GA-cf, as the higher the values in acc, the better the FL method. Actually, our proposal GA-hyb outperforms all the other methods in acc@5 for the benchmark siemens; in the case of jsoup, GA-hyb was statistically superior to GA-cf and H-cf, yet it was inferior to GA-df, and it presented no significant statistical difference when compared to H-df.

Analyzing the $\hat{A}_{12}$ results for the metric *wef@5*, we see the GA-hyb outperforms all the methods, as low values in wef indicate a better FL method. In siemens, GA-hyb achieved lower *wef@5* than its adversaries in 100% of the 30 runs (i.,e. $\hat{A}_{12} = 1$). In the jsoup, it was 100% better than all the others but one, where it reached lower *wef@5* in 88% when compared to GA-df. In such a comparison, GA-hyb was outperformed by GA-df in *acc@5*. However, it was worst in 82% runs while better 88% times in *wef@5*. Finally, except against GA-cf, GA_hyb had lower critical ties (*actie*) than all the other methods.

Therefore, considering the discussion presented in this section we can answer the research questions.

RQ1: Is the proposed method competitive for locating software faults? concerning the accuracy, the dua-to-node score performs better than control-flow heuristics in *Siemens Suite* and *jsoup*, but analyzing Wasted Effort, the same does not happen with the *Siemens Suite* results. Also, the number of critical ties is considerably higher than control-flow heuristics in both benchmarks. So the answer to RQ1 is"**Yes, the proposed method is competitive, performing even better than traditional methods in the accuracy metric applied in top-5 of suspiciousness ranking**".

RQ2: Does the evolutionary combination of control-flow and data-flow coverage spectra improve fault localization ability? The evolutionary approach GA_hyb performed as accurately as the node-to-dua heuristics to locate faults looking the top-5 elements of the suspiciousness ranking, and demonstrate improvement in the *wef* and *actie* results. So, the answer to RQ2 is "**Yes, although the hybrid approach did not show the best results when analyzing accuracy, it may present contributions reducing the number of critical ties and consequently the number of inspected elements, with a wispy loss of accuracy**".

6 THREATS TO VALIDITY

Some aspects of this study offer potential threats to validity. *Conclusion Validity Threats* - the results using GA approaches suffer from the stochasticity presented in such models. To mitigate this issue, we conducted 30 executions of each GA setup; furthermore, we applied statistical tests to improve assurance about the conclusions presented. *Internal Validity Threats* - To deal with internal threats we executed the three GA-based methods by using the same parameters, as mutation and crossover rate and the number of generations. The manual instrumentation process utilized to obtain control-flow

and data-flow coverage spectra for *Siemens Suite*, was conducted with continual reviews, aiming to ensure no difference in the test case execution when compared with the original programs; additionally, we made the instrumented programs accessible publicly[2]. The coverage spectra of *jsoup* are also available publicly[3] by the JAGUAR tool authors. *Construct Validity Threats* - We utilized a novel metric (*actie*) to investigate FL methods. However, it was used only to support the results of other metrics, which are well established in the literature. *External Validity Threats* - whether more generalizable results, assessment on a large scale for faults and programs, including other programming languages (PL), are needed. We have included programs with different sizes and PLs; the subject programs also present seeded and real faults.

7 FINAL REMARKS

This paper reports an investigation about the use of data-flow based suspiciousness scores in FL methods, and also their behavior in evolutionary combinations of heuristics. The approach is conducted either using data-flow only or together with the traditional control-flow based suspiciousness score.

Absolute metrics support the evaluation of effectiveness of the methods and indicate that, when comparing the methods with well-know heuristics as baselines, the data-flow based approaches demonstrate superior results to their control-flow version.

In order to further investigate the impact of the inclination to generate ties found in data-flow suspiciousness scores, we introduce an absolute metric to measure critical ties and analyze its effects along with other metrics.

The occurrence of ties is natural, but their intensity (size) depends on factors such as method, data and test results; Thus, in future work, we will investigate attributes of the test that impact the occurrence of ties of the FL methods. Besides that, we intend to analyze how the information concerning ties may also be used to guide the search in evolutionary FL approaches.

REFERENCES

[1] R. Abreu, P. Zoeteweij, and A. J. c. Van Gemund. 2006. An Evaluation of Similarity Coefficients for Software Fault Localization. In *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. 39–46. https://doi.org/10.1109/PRDC.2006.18

[2] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. 2009. Spectrum-Based Multiple Fault Localization. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering (ASE '09)*. IEEE Computer Society, Washington, DC, USA, 88–99. https://doi.org/10.1109/ASE.2009.25

[3] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C. van Gemund. 2009. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software* 82, 11 (2009), 1780 – 1792. https://doi.org/10.1016/j.jss.2009.06.035 SI: TAIC PART 2007 and MUTATION 2007.

[4] Paul Ammann and Jeff Offutt. 2017. *Introduction to software testing* (edition 2 ed.). Cambridge University Press, Cambridge, United Kingdom ; New York, NY, USA.

[5] A. Arcuri and L. Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. ACM, New York, NY, USA, 1–10.

[6] Tien-Duy B. Le, David Lo, Claire Le Goues, and Lars Grunske. 2016. A Learning-to-rank Based Fault Localization Approach Using Likely Invariants. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. ACM, New York, NY, USA, 177–188. https://doi.org/10.1145/2931037.2931049

[7] D. M. De-Freitas, P. S. Leitao-Junior, C. G. Camilo-Junior, and R. Harrison. 2018. Mutation-Based Evolutionary Fault Localisation. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. 1–8. https://doi.org/10.1109/CEC.2018.8477719

[8] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: A Python Framework for Evolutionary Algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '12)*. ACM, New York, NY, USA, 85–92. https://doi.org/10.1145/2330784.2330799

[9] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering* 10, 4 (2005), 405–435.

[10] Gordon Fraser and José Miguel Rojas. 2019. *Software Testing*. Springer International Publishing, Cham, 123–192. https://doi.org/10.1007/978-3-030-00262-6_4

[11] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based Software Engineering: Trends, Techniques and Applications. *Comput. Surveys* 45, 1, Article 11 (Dec. 2012), 61 pages. https://doi.org/10.1145/2379776.2379787

[12] Matthew S. Hecht. 1977. *Flow Analysis of Computer Programs*. Elsevier Science Inc., New York, NY, USA.

[13] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. 1994. Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria. In *Proceedings of 16th International Conference on Software Engineering*. 191–200. https://doi.org/10.1109/ICSE.1994.296778

[14] James A. Jones, Mary Jean Harrold, and John T. Stasko. 2009. Visualization for Fault Localization. In *in Proceedings of ICSE 2001 Workshop on Software Visualization*. 71–75.

[15] L. Naish, H. J. Lee, and K. Ramamohanarao. 2009. Spectral Debugging with Weights and Incremental Ranking. In *2009 16th Asia-Pacific Software Engineering Conference*. 168–175. https://doi.org/10.1109/APSEC.2009.32

[16] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2011. A Model for Spectra-based Software Diagnosis. *ACM Trans. Softw. Eng. Methodol.* 20, 3, Article 11 (Aug. 2011), 32 pages. https://doi.org/10.1145/2000791.2000795

[17] Chris Parnin and Alessandro Orso. 2011. Are Automated Debugging Techniques Actually Helping Programmers?. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA '11)*. ACM, New York, NY, USA, 199–209. https://doi.org/10.1145/2001420.2001445

[18] R Core Team. 2013. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/

[19] Colin R. Reeves. 2010. *Genetic Algorithms*. Springer US, Boston, MA, 109–139. https://doi.org/10.1007/978-1-4419-1665-5_5

[20] H. L. Ribeiro, H. A. de Souza, R. P. A. de Araujo, M. L. Chaim, and F. Kon. 2018. Jaguar: A Spectrum-Based Fault Localization Tool for Real-World Software. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 404–409. https://doi.org/10.1109/ICST.2018.00048

[21] Henrique L. Ribeiro, Higor A. de Souza, Roberto P. A. de Araujo, Marcos L. Chaim, and Fabio Kon. 2019. Evaluating data-flow coverage in spectrum-based fault localization. (2019).

[22] R. Santelices, J. A. Jones, Yanbing Yu, and M. J. Harrold. 2009. Lightweight fault-localization using multiple coverage types. In *2009 IEEE 31st International Conference on Software Engineering*. 56–66. https://doi.org/10.1109/ICSE.2009.5070508

[23] Shaowei Wang, David Lo, Lingxiao Jiang, Lucia, and Hoong Chuin Lau. 2011. Search-based fault localization. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, Lawrence, KS, USA, 556–559. https://doi.org/10.1109/ASE.2011.6100124

[24] Jeongju Sohn and Shin Yoo. 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)*. ACM, New York, NY, USA, 273–283. https://doi.org/10.1145/3092703.3092717

[25] W.E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740. https://doi.org/10.1109/TSE.2016.2521368

[26] W. E. Wong, V. Debroy, R. Gao, and Y. Li. 2014. The DStar Method for Effective Software Fault Localization. *IEEE Transactions on Reliability* 63, 1 (March 2014), 290–308. https://doi.org/10.1109/TR.2013.2285319

[27] W. E. Wong, Y. Qi, L. Zhao, and K. Cai. 2007. Effective Fault Localization using Code Coverage. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, Vol. 1. 449–456. https://doi.org/10.1109/COMPSAC.2007.109

[28] XIAOFENG XU, VIDROHA DEBROY, W. ERIC WONG, and DONGHUI GUO. 2011. TIES WITHIN FAULT LOCALIZATION RANKINGS: EXPOSING AND ADDRESSING THE PROBLEM. *International Journal of Software Engineering and Knowledge Engineering* 21, 06 (2011), 803–827. https://doi.org/10.1142/S0218194011005505

[29] Shin Yoo. 2012. Evolving Human Competitive Spectra-Based Fault Localisation Techniques. In *Search Based Software Engineering*, Gordon Fraser and Jerffeson Teixeira de Souza (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 244–258.

[2] Available after review

[3] https://github.com/saeg/experiments