IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 13, 2020, accepted October 26, 2020, date of publication November 3, 2020, date of current version November 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035413

Evolving Suspiciousness Metrics From Hybrid Data Set for Boosting a Spectrum Based Fault Localization

ADEKUNLE AKINJOBI AJIBODE, TING SHU, AND ZUOHUA DING, (Associate Member, IEEE)
School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China

Corresponding author: Ting Shu (shuting@zstu.edu.cn)

This work was supported in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LY17F020033, in part by the National Natural Science Foundation of China under Grant 61101111 and Grant 61572441, and in part by the Fundamental Research Funds of Zhejiang Sci-Tech University under Grant 2019Q039.

ABSTRACT Spectrum Based Fault Localization (SBFL) uses different metrics called risk evaluation formula to guide and pinpoint faults in debugging process. The accuracy of a specific SBFL method may be limited by the used formulae and program spectra. However, it has been demonstrated recently that Genetic Programming could be used to automatically design formulae directly from the program spectra. Therefore, this article presents Genetic Programming approach for proposing risk evaluation formula with the inclusion of radicals to evolve suspiciousness metric directly from the program spectra. 92 faults from Unix utilities of SIR repository and 357 real faults from Defect4J repository were used. The approach combines these data sets, used 25% of the total faults (113) to evolve the formulae and the remaining 75% (336) to validate the effectiveness of the metrics generated by our approach. The proposed approach then uses Genetic Programming to run 30 evolution to produce different 30 metrics. The GP-generated metrics consistently out-performed all the classic formulae in both single and multiple faults, especially OP2 on average of 2.25% in single faults and 3.42% in multiple faults. The experiment results conclude that the combination of Hybrid data set and radical is a good technique to evolve effective formulae for spectra-based fault localization.

INDEX TERMS Debugging, fault localization, genetic programming, SBFL.

I. INTRODUCTION

Software systems grow larger and more complex daily, and they are beleaguered with an increasingly large number of faults which later leads to unwanted errors. While automated test data fault location may reduce the cost of test creation and eventually lead to easier debugging in the System Under Test (SUT). The task of debugging itself is still largely left to software developers, taking up to 75% of total software cost for some projects [1].

Software fault localization, which mainly focuses on pinpointing the location of faults, is considered to be the most expensive and time-consuming activities in debugging [2]. In addition, for a large-scale program in particular, manual location of fault codes becomes a problem due to the limitation of testers experience and knowledge. Even in an educational setting, based on interactions with students, it has been reported that finding faults in program code is an extremely difficult task [3], [4]. But fault localization is very important in the debugging process because the faster the source of an error can be found, the faster the cause of the error can be addressed; and this speed reduces the amount of downtime a system may incur, thereby improving availability [4], [5]. So, approaches that can help to automate the fault location process, thereby making software more reliable and maintainable, are currently in great demand.

Therefore, to extenuate this challenge, a number of approaches [6]–[8] [9], [10] [11], [12] have been proposed in both Academia and industry to automate fault localization, while improving the efficiency and effectiveness of software systems. The intuition of fault localization technique is to create a ranking of most probable faulty entities (e.g., program statements, methods or predicates) such that these components may then be examined by developers in order of their suspiciousness (likelihood of containing bugs) until a fault is found. A good technique should rank a faulty component

The associate editor coordinating the review of this manuscript and approving it for publication was Resul Das.

towards the top (if not at the very top) of its ranking such that a fault is discovered early while accessing the ranking. In fact, if the faulty component is at the very top of the ranking (i.e., it is the first component that is examined), then the programmer has been led to the location of a fault purely automatically, without having to intervene thus far [4]. Among the proposed software fault localization techniques is the spectra-based software fault localization (SBFL) which takes the coverage pass and fail information of individual test cases and assigns to each program element in SUT which is called a suspiciousness score. The score of a given program element is expected to be correlated with the likelihood that it contains a fault. The expectation is that the developer will inspect the program elements following the ranking of these suspiciousness scores, thereby reaching the faulty program element faster than following the actual order given by the source code structure. Suspiciousness scores reflect how likely it is for each program entity to be faulty and can be used to sort program entities. Then, developers can follow the suspiciousness rank list (from the beginning to the end in descending order) to manually inspect the source code to diagnose the actual root cause of failures. Recently, SBFL techniques have also been utilized by various automated program repair techniques to localize potential patch locations [13].

The effectiveness and performance of SBFL technique are mostly determined by the risk evaluation formula. The majority of the existing, widely studied formulae are inherited from other fields, such as [14], [15], designed by human intuition, such as [16], [17] and generated by genetic programming, such as [18], but are mainly effective in single fault localization. But as of present, there is no assurance that one formula is optimal for all classes of faults [19].

Formulating a risk evaluation formula that performs invariably against all possible combinations of various program structures, test suites, and potential locations of faults remain a demanding in the field which is now a difficult task for the human to achieve. The only available methodology by human is trial and error: to design automatically and evaluate empirically [18]. Previous studies have made efforts to design a risk evaluation formula that can be proven to be optimal, but only with respect to the case that the fault is known and already contained within a specific program structure [20], [21]. Recently, Genetic Programming (GP) has been successfully applied to automatically design risk evaluation formula [18] which also has proven to be optimal and can be improved, in turn could be universally acceptable for a a spectrum based fault localization. In [18], the empirical results show that, among the 30 GP-evolved formulae, only 8 of these formulae were effective and could outperform some human-designed formulae. However, the formulae were evolved and evaluated on a set of small benchmarks from the Software-artifact Infrastructure Repository (SIR) [22] which contain artificial and seeded faults, and with limited GP operators. The artificial faults are mutants which are automatically created by a tool [23], or mutant-like manually seeded faults created by students [4], or researchers [24]. Artificial faults such as mutants differ from real faults in many respects, including their size, their distribution in code and their difficulty of being detected by tests [25]. It is possible that an evaluation and evolving of metrics on real faults would yield different outcomes than previous study evaluations and evolve on mutants [25].

Evolving suspiciousness metrics from artificial and seeded faults might be the reason that the previous GP generated metrics were unable to outperform one of the humans' designed formulae such as OP2. It is possible that the evolving and evaluation of GP-generated metrics on real faults would yield positive result, thus resolving a cloud of doubt that currently hangs over the field. This apparently shows that there are still ways of improving the process of automatically generating good evaluation formulae that can outperform all human designed formulae and serve as a guide for a spectrum based fault localization. Our motivation elicits from this limitation that the previously generated GP-formulae were unable to outperform some human design formulae, and we believe that evolving suspiciousness metrics from the hybrid data set that contain both artificial faults from small program benchmarks and real faults from large program benchmarks, with the inclusion of more mathematical syntax would produce a better performing metric.

This article re-introduces an evolutionary approach to design a risk evaluation formula for SBFL using Genetic Programming with a python API. This article uses program spectra from four Unix utilities of Software Infrastructure Repository [26] and five Java utilities from Defect4J [27] and the location of their various injected faults and real faults as GP input.

Our study answered four research questions. Firstly, since we combine two data set repositories to evolve our suspiciousness metric, it is our interest to know how the metrics perform in a single data set. We firstly separated single fault programs from multiple fault programs in defect4j repository, we then combined these single fault programs with the other single fault programs collected from SIR repository. We measured the performance of our GP-generated metrics from these combinations and compared the results with the human designed formulae and previously GP-formulae which were reported in [28] as classic formulae. A total number of 255 single faults were used in this comparison. Secondly, the previous research in [18] did not include multiple faults to their study, so, we used Defect4J multiple fault data set to evaluate the effectiveness of our method in multiple faults. We also compared the result to the human designed formulae and previously GP-generated formulae. A total number of 194 multiple faults were used in this comparison. Thirdly, the previous research did not include radicals in their study (except square root) which by virtue have important and convincing improvements in the metrics generated. We measure the effectiveness of these radicals in all data sets used in the experiment by comparing the performance of our metrics that have radicals with the classic formulae in both single and

multiple faults. We therefore use a total number of 449 faults in this comparison. Finally, we use the Wilcoxon signed rank test and Cliff's delta effect size statistical analysis method to test if there is any statistical difference between our generated metrics' performance and other formulae' performance.

Our study has the following main findings:

- Using hybrid data set to evolve suspiciousness metrics have great advantages over the classic formulae. Our study is able to generate 29 metrics that outperform both human design formulae and GP generated formulae that were previously studied.
- The approach produces suspiciousness metrics that can outperform the human design formulae in multiple fault as the previously studied formulae were only effective in single fault localization. This is the greatest advantage of the a spectrum based fault localization research field.
- The experiment results show that radicals are useful mathematical tools to propose well performing metrics for debugging. These have been used in other fields like meteorology to propose formulae for weather forecasting.

To sum up, the paper makes the following contributions:

- This article presents an evolutionary approach to generate a risk evaluation formula from hybrid data sets for SBFL. All existing formulae have been manually designed, often relying only on intuition. The previously GP generated formulae were generated from single fault benchmark, which only outperformed few of the human designed formulae. We are able to generate 29 well performing formulae from hybrid data sets that contain real and artificial faults. These formulae are effective in locating faults in small, medium and large programs.
- It extended the GP operation with complex square roots (radicals) to evolve formulae. The approach we introduced is evaluated with empirical studies, using test spectra data from real world Unix and Java utilities. The result shows that complex roots are good mathematical tools to propose a formula as they have been used in other fields. None of our GP-generated formulae that has complex square root is outperformed by any human design formula.
- The empirical evaluation shows that GP-generated risk evaluation formulae can outperform those formulae designed by human. This implies that if there is need to improve suspiciousness metrics, GP is capable of producing better results.
- The empirical results proved that GP-generated risk evaluation formula from multi-fault data set outperform GP risk evaluation formulae generated from single data sets in both single and multi-fault program spectra. The implication of this result is that we may continue to get better results while including more repository data set.

The rest of this article is organized as follows. In section 2, this article presents some backgrounds on a spectrum-based fault localization. Section 3 explains how we formulate the risk evaluation formulae using Genetic Programming. Section 4 describes the experimental setup and presents the results of our experiment. In section 5, we discuss the results of the empirical evaluation, section 6 presents the related studies and in section 7, we conclude and discuss future work.

II. BACKGROUNDS

A. A SPECTRUM BASED FAULT LOCALIZATION

a spectrum-based fault localization provides fault suspiciousness ratio by analyzing the relationship between a test result (pass or fail) and the visiting information about a statement. We assume that if fail executions happen, a fault exists among statements that were visited during a test in runtime. However, we cannot expect to determine the exact fault location only by the fail test case. Therefore, the pass test cases are also utilized to narrow down the faulty statements. For the fact that fault location aims to reduce the cost of debugging by guiding the process of searching for the location of the fault in the program, then if a program entity is more likely to be faulty, it should be assigned a higher priority in the suspiciousness list. In the same manner, an ideal SBFL technique should always rank the faulty entity with the high suspiciousness score, which can significantly speed up the debugging process for finding the root-cause of test failures. To compute suspiciousness scores of program entities, a SBFL technique first run tests on the target program and record the program a spectrum of each failing or passing tests, i.e., the run-time profiles about which program entities are executed by each test [29]. Based on the program spectra and test outcomes, various statistics can be extracted for suspiciousness computation, e.g., tuple (ef, ep, nf, np), where ef and ep are the numbers of failing and passing tests that execute the program entity e, while nf and np are the numbers of failing and passing tests that do not execute the program entity e. Based on such tuples, various SBFL formulae have been proposed by different researchers. The common assumption of this technique is that a program entity executed by more failing tests and less passing tests is more likely to be faulty. This article considers 14 well known, available SBFL metrics and recently proved GP-classic metric; Tarantula, Statistical Bug Isolation (SBI), Ochiai, Jaccard, Ochiai2, Kulczynski, Op2, Dstar, Wong1, Wong2, and Wong3 [16], [20], [30], [31]. Tarantula, SBI, Ochiai and Jaccard are the most widely-used techniques for the evaluation of fault localization [32], [33]. Ochiai2 is an extended version of Ochiai, which considers the impact of non-executed or passing test cases [14]. Op2 is the optimal SBFL technique for single-fault program [33], whereas Kulczynski and Dstar (star $= 2$) belong to the formula family Dstar which is shown to be more effective than 38 other SBFL techniques [4]. Wong1, Wong2 and Wong3 have been proved to be widely used in many literatures and are considered to be effective for single fault localization [34]. GP02, GP03 and GP19 are the currently proved classic metric for single fault localization [28]. SBFL techniques subsequently use a risk evaluation formula, which

TABLE 1. Motivating example: The faulty statement s5 achieves the first place when ranked according to the tarantula risk evaluation formula.

s_i	mid ( ) { input x, y, z, m;	Test1	Test2	Test3	ep	ef	np	nf	Tarantula	Rank
s1	if (y <z)	*	*	*	2	1	0	0	0.50	4
s2	if (x <y)	*	*	*	2	1	0	0	0.50	4
s3	m=y;	*	*		2	0	0	1	0.00	8
s4	else if (x <z)	*			1	1	0	1	0.33	6
s5	m=z;//fault (m=x)			*	0	1	2	0	1.00	1
s6	else			*	0	1	2	0	1.00	1
s7	if (x >z)		*		1	0	1	1	0.00	8
s8	m =x;	*		*	1	1	1	0	0.67	3
s9	else if (x >y)	*			1	1	0	1	0.33	6
s10	m =y				0	0	2	1	0.00	8
s11	print (“middle number is:“,m);				0	0	2	1	0.00	8
s12	}				0	0	2	1	0.00	8
Result	Pass/Fail Status	P	P	F						

is a formula based on the four counters, to assign risk scores to statements: the scores are designed to correlate the relative risk of each statement containing the fault. Table 1 presents an illustrative example of the Tarantula metric [16], being applied to a SUT with 13 structural elements. Let us assume that the element s5 is the faulty statement, which causes the test case t3 to fail, whereas test case t1 and t2 pass. The second column presents the coverage achieved by these three test cases, respectively. The a spectrum column aggregates the coverage and test results into a set of the aforementioned tuples, which are fed into the Tarantula metric, eventually forming the rank. SBFL technique assumes that the developer is to investigate the SUT following the rank order produced by the technique. In the case of the example above, the developer would find the faulty element first, instead of as the fifth element when inspected following the line number order.

B. THE RISK EVALUATION FORMULAE

The efficacy of SBFL technique is determined by the risk evaluation formulae. Many existing formulae are generated by human, while few are generated by Genetic Programming. Table 2 contains several of the widely studied formula. Interestingly, Jaccard [33] and Ochiai [14] were first studied in Botany and Zoology respectively, but have been subsequently studied in the context of fault localization [20], [21]. The Tarantula was originally developed as a visualization method [16] but also increasingly considered as an SBFL risk evaluation formula independent of visualization [32]. SBI [35] and three different versions of Wong metric [2] have been introduced specifically for fault localization.

Op2 and SBI metrics are recently added to SBFL techniques which show an interesting research focus: these metrics are proven to produce an optimal ranking, as long as the fault is located in a specific program structure (two consecutive If-Then-Else blocks, called ITE2) [20]. Although the proof does not guarantee that Op2 is optimal for all locations of faults (and not just limited to ITE2), the empirical evaluation shows that Op2 is a very strong formula.

TABLE 2. Well known risk evaluation formula.

Name	Definition
Tarantula	$\frac{\frac{ef}{ef+nf}}{\frac{ef}{ef+nf}+\frac{ep}{ep+np}}$
SBI	$1-\frac{ep}{ep+nf}$
Ochiai	$\frac{ef}{\sqrt{(ef+ep)(ef+nf)}}$
Jaccard	$\frac{ef}{ef+ep+nf}$
Ochiai2	$\frac{(ef)(np)}{\sqrt{(ef+ep)(nf+np)(ef+np)(nf+ep)}}$
Kulczynski	$\frac{ef}{nf+ep}$
OP2	$ef-\frac{ep}{ep+np+1}$
Dstar	$\frac{ef^2}{ep+nf}$
GP02	$2(ef+\sqrt{np}+\sqrt{ep})$
GP03	$\sqrt{\lvert ef^2-\sqrt{ep}\rvert}$
GP19	$ef\sqrt{\lvert ep-ef+nf-np\rvert}$
Wong1	ef
Wong2	$ef-ep$
Wong3	$ef-h, h=\begin{cases} ep & \text{if } ep\leq 2 \\ 2+0.1(ep-2) & \text{if } 2<ep\leq 10 \\ 2.8+0.001(ep-10) & \text{if } ep>10 \end{cases}$

C. GENETIC PROGRAMMING FOR RISK EVALUATION FORMULA

The formal approach is difficult for humanity to come up with at least five optimal formulae at the same time. The performance proof of optimality by [20] has presented a complete approach in designing a risk evaluation formula, which will require significant human efforts to provide optimality proof for a wider range of program structures. GP use data driven iteration technique which enables it to examine and

organize the provided data with the goal of better finding the suitable populations to merge for better performance. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. Here, the process of natural selection starts with the selection of the fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. Barring the formal proof of optimality, the most intuitive process of designing a risk evaluation formula would be an iterative modification of a candidate formula, against a wide range of spectra data sets, until its performance reaches an acceptable level. Not only the amount of data will burden the human designer, but its process also will. In fact, this is how GP operates, i.e., a data-driven, systematic trial-and-error. Although GP cannot replace human completely in designing evaluation formula, but it is actually a powerful tool that the human software developer may use to explore the design space and to identify building blocks for producing better formulae. It helps to gain more insight about a specific domain under construction.

D. COMPLEX SQUARE ROOTS (RADICALS)

Complex roots are usually called radicals in mathematics. A radical is a mathematical symbol used to represent the root of a number. In mathematics, an nth root of a number x, where n is usually assumed to be a positive integer, is a number r which, when raised to the power n yields x:

$$r^n = x \tag{1}$$

where n is the degree of the root. A root of degree 2 is called a square root and a root of degree 3, a cube root. Roots of higher degree are referred by using ordinal numbers, as in fourth root, twentieth root, etc. [36]. Roots of real numbers are usually written using the radical symbol or radix with $\sqrt{x}$ denoting the positive square root of x if x is positive and $\sqrt[n]{x}$ denoting the real nth root, if n is odd, and the positive square root if n is even and x is nonnegative. In the other cases, the symbol is not commonly used as being ambiguous. In the expression $\sqrt[n]{x}$, the integer n is called the index, $\sqrt{}$ is the radical sign or radix, and x is called the radicand [36].

Radical expressions are utilized in financial industries to calculate formulae for depreciation, home inflation and interest [37]. Electrical engineers also use radical expressions for measurements and calculations. Biologists compare animal surface areas with radical exponents for size comparisons in scientific research [37]. Meteorologists have used radicals to determine and predict how long a thunderstorm is going to last, or its duration. The formula used is $t = \frac{\sqrt{d}^3}{216}, t = \text{time}, d = \text{diameter in miles}$. The meteorologists have also successfully used radicals to measure the speed of updrafts in order to predict the type of weather that may be generated by them. They have applied it to develop a formula to explain the stability of the atmosphere. [38]. The application of these radicals in these fields motivates us to apply the same to our GP operators to see if they will have any effect on the formulae that will be generated.

This article employed additional 3 radical signs to the initial studied GP operators in [18], so that the GP-generated formulae would include radicals into the evaluation formulae to be generated.

TABLE 3. GP operators.

Operator Node	Definition
gp_add(a, b)	a+b
gp_mul(a, b)	ab
gp_div(a, b)	1 if b = 0; $\frac{a}{b}$ If otherwise
gp_sqrt(a)	$\sqrt{a}$
gp_cube(a)	$\sqrt[3]{a}$
gp_forth(a)	$\sqrt[4]{a}$
gp_fifht(a)	$\sqrt[5]{a}$

III. THE APPROACH

We propose using hybrid data set to evolve suspiciousness metrics for a a spectrum based fault localization. In this section, we first introduced the framework of using two different data set repositories to evolve suspiciousness metrics using genetic programming by discussing the approach we used to carry out the experiment in the study. Secondly, we discussed the GP representation in our study. Thirdly, we discussed how we selected our fitness function for the GP formulae. Lastly, we also discussed the algorithm employed in this study.

A. FRAMEWORK

The approach Hybrid data set Genetic Programming with Complex Root (HDGPCR) is a learning-based approach where Genetic programming is invoked in a multiple data set repository to generate suspiciousness metrics from these data sets directly. Firstly, we manually separated our data set into their categories; i.e. single fault and multiple faults. All the program benchmarks in SIR repository contain single faults of different versions; Defect4J comprises both multiple faults and single faults. In defect4j, we manually separated single faults from multiple faults, and then randomly selected some faults from flex, sed, gzip, grep, JFreechart (Chart), Google Closure compiler, Apache Commons Lang, Apache Commons Math and Joda-Time (Time). We ensured our faults selection from defect4j repository span through both single and multiple faults, and lastly merged all these faults before invoking our genetic programming to evolve evaluation metrics directly.

Figure 1 illustrates the overall structure of generating HDGPCR. This structure consists of two major phases: learning and ranking. As most approaches in a a spectrum-based fault localization, the GP inputs here, are the faulty programs with test cases, the index of the faults in each faulty program, the GP operators and the classic formulae. In the learning

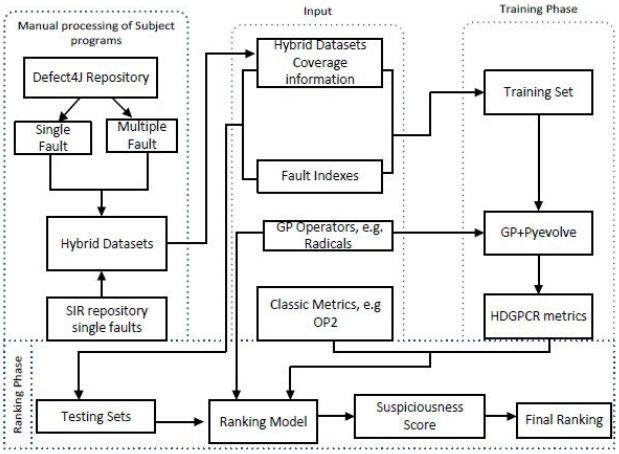


FIGURE 1. Conceptual framework of HDGPCR. This framework takes faulty programs with test cases, set of known faults, GP operators and well-known metrics as input; the output is the final ranking of program entities.

phase, all the faulty programs are those whose program spectra are collected by executing the test cases (including at least one failing test case) and the indexes of the known faults are transferred to the GP and transform into the fault dictionary. The faults' dictionary were then partitioned into two as training set and evaluation set. Genetic Programming learns from these training data sets and compute the fitness of a candidate GP tree as the average normalized ranking of the known faulty statements in the training a a spectrum data sets. GP, runs 30 evolution to produce a range of different formulae. In the ranking phase, the formulae are collected and used to compute the suspiciousness score of the evaluation data set along with other classic formulae. The suspiciousness scores are then collected for ranking. These ranks are formed as the final fault localization result.

B. THE GP REPRESENTATION

In this article, simple tree-based representation is used [18], and a set of simple and complex operators on the basis that they can sufficiently represent most of the existing mathematical operation of the existing risk evaluation formula. Table 3 shows the GP operators used in this article. Addition, subtraction and Multiplication do not need any special treatment, because these operators cannot result in numerical exceptions. The gp_div which will cater for division is conditioned that it will return 1 to avoid ZeroDivisionException in case of any fractional denominator computation that will result in zero. Comparably, the radical operators (gp_sqrt, gp_cube, gp_forth and gp_fifth) are protected absolutely for the given input. Averting numerical exception this way is an advantage for computational environments without advanced exception handling mechanism like GPGPU platforms, without losing too much demonstrative power. For terminal symbols, we use the program spectra data ep, ef, np, nf as well as constant 1.

C. FITNESS FUNCTION FOR GP

The focus of the risk evaluation formula is to ensure that the assigned risk value of a faulty statement results in a highly ranked of that statement in order to reduce the search space of the developer before locating the root cause of the fault. This implies that the performance of the risk evaluation formula can be measured by the position it ranks the faulty statement when the formula is used.

In literature, this correlative evaluation is often referring to as the Expense Metric, which is a normalized ranking of the faulty statement. When a formula is given as F, a set of subject programs P and a fault b in p, the Expense metric can be calculated as;

$$E(F, p, b) = \frac{\text{Ranking of 'b' according to 'F'}}{\text{Number of Statements in 'p'}} * 100 \quad (2)$$

Expense is an evaluative metric that is only applicable when the faulty statement is known. For the fact that this approach is evolving formulae from a known fault, Expense is useful in this case to calculate the fitness function for our genetic programming. But in order to avoid over-fitting to the location of a specific fault, the approach calculates Expense metric for a candidate formula using many faults from different data sets and take average as the fitness function.

For a set of n known faults B $= b1.....,bn$ from corresponding n programs P $= p1.....,pn$, fitness value of a candidate risk evaluation formula F is calculated as follows:

$$Fitness(F, B, P) = \frac{1}{n}\sum_{i=1}^{n} E(F, pi, bi) \quad (3)$$

Based on the risk evaluation formula, multiple statements may get assigned the same risk evaluation values and thereby tie in the ranking.

It is not actually clear what will be the appropriate tiebreaker for a candidate formula, but we break ties and assign the most conservative ranking to all tied ranks before them [39].

D. THE ALGORITHM

To generate HDGPCR metrics, the GP+Pyevolve algorithm used in the experiment starts with generating an initial set of individuals with ramped method, where the population has the flexibility to be of any depth (within maximum length). Each of the candidate is then evaluated using a fitness function and only γ-best candidates are selected to evolve. The fitness calculates the fault position ranking in the data set. The evolution process begins by selecting candidates with Tournament Selection operator. Then crossover and mutation operations get applied to those candidates until β-children gets generated. New offsprings to be generated will have the first half of their genes taken from the first parent and the other half from the second parent. Newly generated candidates are evaluated using fitness function and best γ children are selected to proceed to the next iteration, while the unfit candidates are discarded. At each generation the best fit solution is added to the output set as a set of GP tree formulae.

This evolution process runs until it satisfies one or more termination conditions which is 30 in our case. To ensure our generated optimal formulae are not only good in training data set, we evaluated the formulae along with the classic formulae in the validation data set. The algorithm used in this study for generating evaluation metrics for fault localization in its general form is stated in detail in algorithm 1.

Algorithm 1 HDGPCR Algorithm

Input:
GP Operators P: set of faulty programs with test cases. C: Classic formulae. I: Faults Indexes.
Output:
Learning Model
for each faulty program $p \in$ P; **do**
Select training set from the data set
merge index i to p for $i \in$ I and $p \in$ P as a dictionary.
Compute fitness score of each faulty program
Select **k** best individuals
while the maximum number of generations was not reached **do**
Using the Tournament Selection method select parents.
Apply Crossover and Mutation until **u** descendants generated.
Calculate Fitness.
Add the best formula strings to Output Set
Select **k** best descendants and update population with those for the next iteration.
end while
end for
Ranking Model
for each faulty program $p \in$ P; **do**
Select evaluation set from the data set
Extract GP generated suspiciousness metrics
Extract Classic suspiciousness metrics
Compute Suspicousness score
Rank computed suspiciousness score
end for

The model takes as input the GP operators, set of faulty programs with test cases, classic formulae and a faulty entity index. The model learns from the program entities in relation to the faulty index, converts the two to dictionary and calculates a fitness function for each fault dictionary to produce the best string of formulae that fits the data set. Our goal is to come up with the best formulae that fit into our data set which can also be used in a a spectrum based fault localization problems solving.

IV. EXPERIMENT METHODOLOGY

The approach is evaluated by analyzing the performance of classic metrics along with the metrics generated by our approach on a small and large program that contain both artificial and real faults from two different repositories data sets. The evaluation metrics and statistical analysis employed to examine the effectiveness of our ranking metrics are also discussed in this section.

A. EVOLVING THE METRICS

In this article, a total number of 449 faults were used from two different Repositories i.e. SIR and Defect4J, which are flex, grep, gzip, sed, JFreechart (Chart), Google Closure compiler, Apache Commons Lang, Apache Commons Math and Joda-Time (Time) as highlighted in table 4. The programs were written in C and Java respectively, and were all downloaded from [22], [27]. From SIR repository, 92 faults were used, while 357 faults from Defect4J were also used. To evolve the metrics, we ensured 25 percent of the total faults in the SIR repository benchmark are used which is 23 faults in total. Defect4J data set is separated as single and multiple faults. There were 163 single faults in defect4j repository, and 25 percent of these single faults were selected which gives 41. Also, there were 194 multiple faults in this same defect4j data set benchmark, and 25 percent of these faults were also selected randomly which gives 49. These data sets were randomly selected across all the faulty versions. In total, we used 113 faulty programs to generate our metrics.

TABLE 4. Subject program.

SIR	Description	No of Test	Line of Codes	No of Faults
Flex	Lexical analyzer	670	9,933	47
Gzip	Compression utility	214	3883	18
Grep	Text-search utility	809	7,309	11
Sed	Stream text editor	449	5257	16
Total		**1,260**		**92**
Defect4J				No of Faults
JFreechart (Chart)	Chart	2,205	96,000	26
Google Closure compiler	Closure	7,927	90,000	133
Apache Commons Lang	Lang	2,245	22,000	65
Apache Commons Math	Math	3,602	85,000	106
Joda-Time (Time)	Time	4,130	28,000	27
Total		**20,109**		**357**
Sub-Total		**21, 369**		**449**

B. RESEARCH QUESTIONS

The following research questions are answered in this article:

RQ1: *Can GP formulae with Complex Square roots generated from Hybrid data set outperform the classic formulae in single fault programs?*

To answer this question, a combination of data sets is done for both training and evaluation purpose. The previous study that used GP to generate evaluation metrics for a spectrum based fault localization used only a single fault subject programs from SIR repository which contain only seeded and artificial faults. The research was able to generate 8 GP formulae that outperformed the classic formulae except OP2 in the experiment conducted by [18]. In our approach, we combined single fault subject programs from SIR repository and single faults subject programs from Defect4J repository to evaluate the effectiveness of formulae generated by our approach along with the classic formulae. We used 11 human design classic formulae in this comparison. More so, GP02, 03 and 19 were proved to be classic in research [27].

Therefore, GP02, GP03 and GP19 are also combined with the human designed classic formulae for evaluation in single fault programs along with the formulae generated by our approach. In total, we compared 14 classic formulae with our generated formulae. We name our formulae as HDGPCR (Hybrid data set GP with Complex Root). There are 92 single faults from SIR repository and 163 single faults from Defect4J. We used a total number of 255 single faults in this section.

RQ2: *Can GP formulae generated with Complex Square-roots from Hybrid data set outperform the classic formulae in multiple fault subject programs?*

We used a defect4J multiple fault data set to evaluate our GP-generated formulae in this research question. Our formulae are instrumented in multiple fault programs along with the 14 classic formulae used in research question 1. There are 194 multiple-faults in defect4J version used in this article.

RQ3: *What is the effectiveness of Complex Square root on the GP-generated formulae?*

To answer this question, we combined all the faulty programs use in this experiment and instrument our generated formulae along with the classic formulae in these data sets. We then accessed the performance of these formulae on all the 449 faulty programs used in the experiment. We finally separated all our GP generated formulae that have complex square roots (radicals) i.e. $\sqrt[3]{}$, $\sqrt[4]{}$, $\sqrt[5]{}$ from our formulae that do not include these radicals in their expression. The results are separated for comparison purpose.

RQ4. *Is there any statistical difference in the performance of the new generated formulae with complex square root from hybrid data sets and other classic formulae?*

To answer this question, we employed two different statistical tools. The first statistical tool measured the significance in the performance of our metrics and the classic metrics and the second statistical tool measured the effect size of the mean of our metrics and classic metrics to measure the degree of the differences in their performances.

C. SUBJECT PROGRAMS

This article conducts an experiment on SIR [26] and Defect4J [27] benchmarks, which have been commonly used data set in automatic fault localization [28]. SIR repository contains 92 seeded and real faults in C language while Defect4J contain 357 real-world faults from five large open-source projects in Java language. SIR provides a total of 219 (both real and seeded) faults across the five versions of the four subject programs [26]. We excluded 35 of these faults because these faults were unreachable when compiled for the experimental environment, and additional 92 faults because these are not detected by the chosen test suits. In the SIR repository benchmark, *flex* is a lexical analyzer, *grep* is a text-search utility,*gzip* is a compression utility and *sed* is a stream text editor. *JfreeChart* is a framework to create charts, *Google Closure* is a JavaScript compiler for optimization, *apache commons-Lang and common-Math* are two libraries complementing the existing JDK, while *Joda-Time* is a standard time library from Defect4J repository. The subject programs involve diverse applications with different scales from 9,000 to 96,000 lines of codes. The table below shows the list in detail.

D. EVALUATION METRIC

The effectiveness of a software fault localization technique is normally evaluated by the percentage of statements that need to be examined to find a fault in a program under test. This means that a developer will examine all the program executable statements from top to bottom according to the ranking list generated by the fault localization technique. The lower the exam score of each technique, the better the performance of such technique. Therefore, to evaluate the effectiveness of our metrics, the following metrics are used in this study.

1) EXAM SCORE

To assign a score to every faulty version of each subject program of the system under test, [10] use a function which is defined as percentage of the program that need not be examined to find a faulty statement in the program or a faulty node in the corresponding program dependence graph. This metric represents the developer's effort to find a fault using a list of suspicious program entities. The EXAM score is measured as the relative position in which the faulty entity was ranked. It represents the percentage of entities that must be examined to find the fault. The metric is computed as:

$$\frac{\text{Fault Position Ranking}}{\text{Total number of executable statement}} * 100 \qquad (4)$$

Therefore, in this article, we employed EXAM score to compute suspiciousness score for each faulty element. The rank of these scores is then outputs to a file for comparison purposes.

2) WILCOXON SIGNED-RANK TEST

Wilcoxon signed-rank test which is an alternative option to other existing hypothesis tests such as z-test, paired student's t-test, etc., particularly when a normal distribution of a given population cannot be assumed [4]. Wilcoxon signed rank test can be utilized to give a comparison with a solid statistical basis between different techniques in terms of effectiveness and superiority. After computing the exam score which measures the percentage of lines of code a developer will have to examine before locating the first fault, an evaluation is conducted on the two-sided with "less" alternative hypothesis that each of our metric require the examination of an equal or greater number of statements to be examined before locating the first fault than the classic metrics. Therefore, the null hypothesis, in this case, specify that our metrics will require examining greater statements than the classic formulae before locating all faults. The null hypothesis is stated as follows:

H_0: The number of statements examined by our metrics to locate all faults in both single and multiple faults programs $\geq$ the number of statements examined by classic metrics.

Finally, if H_0 is rejected, the alternative hypothesis is accepted. The alternative hypothesis implies that our metrics will examine fewer statements than classic metrics to locate all faults in both single and multiple faults. This implies that our new metrics are more effective than the classic metrics. The alpha is set to 0.05 i.e. 5% in the experiment.

3) CLIFF'S DELTA EFFECT SIZE TEST

Cliff's Delta effect size is a measurement that can be used to estimate the effect size for an experiment comparing two samples with ordinal data. Suppose that we have two samples Q = (1,..., m) and R = (1, ...,n). We define the delta function as follows:

$$\delta(i,j) = \begin{cases} +1, & q_{i=1\ldots m} > r_{i=1\ldots n} \\ -1, & q_{i=1\ldots m} < r_{i=1\ldots n} \\ 0, & q_{i=1\ldots m} = r_{i=1\ldots n} \end{cases} \tag{5}$$

We therefore define Cliff's delta effect size as:

$$\delta = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \delta(i,j) \tag{6}$$

From the above equation, cliff's delta values ranges from -1 to 1, i.e. $-1 \leq \delta \leq 1$. It is therefore stated that any value near ± 1 indicate the absence of overlap between the two samples, while values near zero indicate a lot of overlap between the two samples. When a significant p-value is obtained, the associated effective size measure is expected to be near $+1.0$ or -1.0 because the difference between the groups is important. The extreme Delta $= \pm 1$ occurs when the intersection between both groups is the empty set. When a non-significant p-value is found, the compared groups tend to overlap and the Cliff's Delta statistic approaches near to 0.0. For a spectrum based fault localization, this statistical tool can be used to test the mean differences of the exam scores computed by each SBFL metric. We used this statistical tool to compare the mean values of the exam scores computed by all the studied formulae in this article. Also in a spectrum based fault localization, the lower the exam score of a particular metric, the better the performance of such metric. Therefore, in this scenario, the smaller the mean value of each metric in 30 runs the better the metric over other metrics. When comparing our metric with the classic metrics, our verdicts are placed as follows according to [40]:

1) When the value of $\delta = 0$, there is no superiority among the two compared metrics. This means that both have the same performance.
2) When the value of $\delta = 0.147$, there is a small difference in the performance of the two formulae compared
3) When the value of $\delta = 0.33$, there is a medium difference in the two metrics' performance.
4) When the value of $\delta = 0.474$, there is a large difference in the performance of the two metrics.

While comparing two metrics, if the output results in negative, that means the average mean value of the first group data is smaller than the average mean value of the second group data. Therefore, in a spectrum based fault localization when the mean value of metric A is smaller than the mean value of metric B, then metric A is considered to outperform metric B. The verdicts above are used to consider the degree of their superiority over one another.

E. IMPLEMENTATION AND CONFIGURATION

Pyevolve API version 0.6.1rc1 [39] is used to implement the Genetic Programming. The algorithm is executed using the Pycharm community distribution in 64-bits operating system of Intel CPU, with the configuration of python interpreter version 2.7.17. Ideally, pyevolve is not compatible with the latest python version 3.7.8. The file was first processed in order to collect the various fault in the spectra data set as the data set contain Zero-base-index. The population size is set to 40, ramping method is used with maximum depth of 4. 100 generations is used with 1.0 cross over rate and mutation rate of 0.08. The Genetic programming is also configured with rank selection operators.

To generate the formulae, 25 percent of each faulty program benchmarks are used to make 113 in total. Therefore, we split the entire data set into training 25% and 75% for validation. We instrumented our genetic program to run 30 evolution which return 30 different formulae. We then used this result to answer research question 1 to 4.

F. RESULTS AND FINDINGS

In this section, we present the results and Findings. The results of our cross-comparison with other classic formulae in the validation single and multiple faults data sets are also detailed and discussed. The evaluation is based on the average number of statements examined in terms of exam score in percentage, the Wilcoxon Signed-rank test, and Cliff's delta effect size statistics result. The Genetic Algorithm is trained with 25% (113 in total) of each of the faults' data set. To measure the effectiveness of our metrics, we used the remaining 75% (336 in total) faults data set to evaluate the generated suspiciousness formulae. The experiment is configured to run 30 evolution automatically to cater for stochastic nature and the formulae were exported for further evaluation. The mean expenses of the 75% faults are taken and compared with other classic formulae.

Table 5, 6 and 7 show the comparison of EXAM scores of Hybrid Data Genetic Programming Algorithm with Complex Root (HDGPCR), GP-classic formulae and other human designed formulae in single and multiple faults.

1) RQ1: COMPARISON OF HDGPCR METRICS WITH CLASSIC FORMULAE IN SINGLE FAULT

To access the performance of HDGPCR against the classic formulae, we used the evaluation set of single faults in our study to compare the EXAM score of HDGPCR metrics and classic metrics (Table 5). We should note that the single fault evaluation set comprises single faults in the following subject programs; flex, sed, grep, gzip and defect4j single faults. We

TABLE 5. Comparison of EXAM score of HDGPCR, and classic formulae in single fault data set.

ID	HDGPCR	Dstar	GP02	GP03	GP19	Jaccard	Kulczynski	Ochiai	Ochiai2	Op2	SBI	Tarantula	Wong1	Wong2	Wong3
HDGPCR01	33.12	44.93	38.66	50.88	59.07	44.75	44.88	59.92	61.61	34.66	59.93	58.60	40.17	48.86	40.10
HDGPCR02	30.52	45.19	38.04	52.03	59.05	45.02	45.15	59.42	61.05	34.73	59.44	58.78	40.31	48.23	40.05
HDGPCR03	29.94	43.41	38.99	50.98	58.00	43.24	43.37	59.11	60.93	32.90	59.14	58.46	40.3	49.01	41.27
HDGPCR04	32.11	42.63	37.56	48.89	58.96	42.48	42.61	58.69	60.30	33.10	58.73	57.39	40.46	48.67	41.39
HDGPCR05	29.39	42.15	37.68	50.73	58.15	41.98	42.11	57.35	59.04	32.70	57.38	56.04	40.4	48.68	42.59
HDGPCR06	30.92	41.88	36.73	50.75	58.58	41.70	41.83	58.52	59.82	31.98	58.54	57.20	40.57	48.26	40.94
HDGPCR07	32.82	42.64	39.80	51.97	58.32	42.48	42.60	57.64	59.22	32.50	57.67	57.00	40.45	48.75	40.46
HDGPCR08	29.54	41.88	36.81	51.86	58.00	41.72	41.85	57.01	58.85	31.68	57.04	57.04	40.19	48.82	40.18
HDGPCR09	30.52	45.29	38.94	51.84	56.88	45.14	45.27	58.75	60.29	34.13	58.79	58.79	40.08	48.95	40.05
HDGPCR10	33.23	44.13	38.24	50.24	59.49	44.01	44.15	60.56	62.30	33.29	60.58	59.24	40.59	47.95	41.26
HDGPCR11	31.52	43.26	38.55	52.51	57.92	43.10	43.23	58.01	59.64	32.40	58.05	56.71	40.57	48.65	40.59
HDGPCR12	29.91	42.92	37.01	48.50	59.27	42.76	42.89	60.34	62.02	32.96	60.37	59.71	40.37	49.02	40.86
HDGPCR13	25.37	40.13	36.02	49.37	59.37	39.97	40.10	58.27	59.96	30.78	58.30	57.62	40.42	48.52	41.72
HDGPCR14	32.93	43.31	36.76	49.79	58.31	43.13	43.26	59.85	61.27	33.38	59.87	58.53	40.64	47.92	42.06
HDGPCR15	32.09	44.02	37.79	49.47	60.59	43.92	44.06	59.73	61.34	33.71	59.76	58.42	40.7	48.26	42.12
HDGPCR16	28.51	41.41	36.63	50.40	56.64	41.24	41.37	58.43	59.70	31.48	58.44	57.11	40.65	47.71	42.47
HDGPCR17	32.26	45.27	40.18	50.61	59.40	45.18	45.30	58.74	60.29	34.79	58.78	58.12	40.12	48.69	41.49
HDGPCR18	28.31	42.80	36.76	49.10	59.88	42.64	42.77	60.49	61.97	32.83	60.53	59.19	40.49	48.06	42.25
HDGPCR19	32.24	45.37	38.21	50.84	58.25	45.25	45.38	59.56	61.24	34.34	59.58	58.90	40.47	48.37	41.95
HDGPCR20	27.42	41.77	37.49	50.42	58.55	41.61	41.74	57.71	59.46	31.63	57.74	57.07	40.32	48.74	41.11
HDGPCR21	30.65	44.63	38.63	48.84	59.43	44.46	44.60	60.55	61.92	34.44	60.58	59.24	40.47	48.69	42.59
HDGPCR22	29.50	42.69	38.61	49.95	60.43	42.60	42.73	59.20	60.47	32.23	59.24	58.58	40.61	47.87	41.61
HDGPCR23	27.48	42.36	37.19	51.07	58.61	42.25	42.38	59.13	60.98	32.15	59.16	57.83	40.28	48.68	41.35
HDGPCR24	30.72	42.60	36.87	51.56	58.64	42.48	42.62	59.07	60.80	32.24	59.09	57.75	40.47	48.01	42.15
HDGPCR25	32.19	41.34	37.77	50.63	58.04	41.23	41.36	58.23	59.56	31.32	58.25	56.91	40.59	48.26	41.04
HDGPCR26	30.50	42.80	37.30	51.12	59.93	42.65	42.77	58.31	59.73	32.18	58.34	57.68	40.45	47.99	42.19
HDGPCR27	30.31	43.44	38.10	51.46	58.05	43.25	43.38	59.67	61.41	33.16	59.68	58.34	40.21	48.56	41.90
HDGPCR28	30.04	41.30	37.57	50.28	58.98	41.19	41.33	58.86	60.47	31.84	58.89	57.55	40.29	48.75	40.24
HDGPCR29	30.03	41.56	37.44	51.33	58.57	41.38	41.52	57.24	58.93	32.03	57.27	55.94	40.4	48.3	40.69
HDGPCR30	33.05	41.52	38.36	49.48	59.88	41.34	41.47	57.80	59.43	32.97	57.82	56.48	40.22	48.66	41.36

will therefore refer all other metrics in this study apart from HDGPCR as classic formulae/metrics.

In our experiments, we presume that EXAM score only returns the percentage of statements to be examined before locating the first fault in the program under test. Our subject programs contain small and large program benchmarks. The result in table 5 shows that our method yields a positive result by outperforming both the previously generated GP-metrics that have been proved to be classic and human design metrics. This implies that there is a great advantage in using hybrid data sets to propose a good metric for a spectrum based fault localization. The best performing metric for hybrid single fault in this article is HDGPCR13 which will guide the developer to rank the faulty element in the program under test among the first 25% of the total elements, while the worst performance is Ochiai2 where the developer will have to examine more than 60% of the total elements in the subject program before locating the first fault. Furthermore, our method will save programmers nothing less than 63% on average search space to locate bugs in their single fault program. We can observe that only HDGPCR25 is outperformed by OP2, but this HDGPCR25 outperformed all other classic formulae. A good turnaround can be seen in HDGPCR20, HDGPCR23 and HDGPCR13 which will save programmer at least 73% and 75% search spaces before locating the bug in a single bug program. Therefore, the average search space that the formulae generated by our method and classic formulae saved developers in hybrid data set single fault programs used in this article are; HDGPCR(30.57), Dstar(42.95), GP02(37.82), GP03(50.56) GP19(58.77), Jaccard(42.81), Kulczynski(42.94), Ochiai(58.87), Ochiai2(60.47), OP2(32.82), SBI(58.90), Tarantula(57.87), Wong1(40.41), Wong2(48.46), Wong3(41.33). Jaccard, Dstar and Kulczynski have almost the same performance and GP19, Ochiai and SBI also have almost the same performance. This implies that some formulae belong to the same family according to their performances. The best three among them are; HDGPCR, OP2 and GP02.

2) RQ2: COMPARISON OF OUR METRIC WITH CLASSIC FORMULAE IN MULTIPLE FAULT

This research question assessed the performance of our metrics in multiple fault program benchmark along with the classic formulae (Table 6). This experiment used 194 multiple fault data sets from Defect4J. The Exam scores of the metrics are collected and compared with each other in order to evaluate the performance of our method. The advantage of using hybrid data set to evolve risk evaluation formulae has been shown in the table 6. We have assessed the performances in single fault and our method performed outstandingly. Seen the overwhelming performance in multiple faults is now pointing a green light to the a spectrum based fault localization community that it is getting to its climax already. In multiple fault data set, it saves at least 63% search space and maximum of 67% search space. This is really a good

TABLE 6. Comparison of EXAM score of HDGPCR, classic formulae in multiple fault data set.

ID	HDGPCR	Dstar	GP02	GP03	GP19	Jaccard	Kulczynski	Ochiai	Ochiai2	Op2	SBI	Tarantula	Wong1	Wong2	Wong3
HDGPCR01	37.18	50.85	40.09	51.98	61.47	50.91	50.96	65.42	66.67	39.07	65.69	65.69	46.67	51.09	50.19
HDGPCR02	33.62	50.22	40.29	51.80	61.38	50.29	50.34	65.13	66.41	38.71	65.42	65.42	45.88	51.58	50.58
HDGPCR03	36.85	50.50	40.15	51.61	61.64	50.58	50.63	65.38	66.66	38.92	65.67	65.67	46.87	51.12	50.62
HDGPCR04	35.16	50.25	40.08	51.60	61.63	50.33	50.37	65.22	66.49	38.77	65.51	65.51	46.53	51.29	51.61
HDGPCR05	35.14	50.36	40.00	51.69	61.51	50.43	50.47	65.37	66.63	38.76	65.65	65.65	46.51	51.38	50.02
HDGPCR06	35.08	50.23	40.08	51.71	61.46	50.29	50.34	65.20	66.45	38.71	65.48	65.48	46.12	51.83	52.49
HDGPCR07	34.43	50.73	40.15	51.86	61.34	50.80	50.85	65.51	66.77	39.05	65.79	65.79	46.63	51.43	51.20
HDGPCR08	37.06	50.48	40.24	51.82	61.34	50.56	50.61	65.39	66.66	38.87	65.68	65.68	46.68	51.09	52.26
HDGPCR09	33.90	50.13	39.95	51.54	61.57	50.20	50.25	65.30	66.54	38.65	65.58	65.58	46.66	50.67	49.53
HDGPCR10	37.72	50.24	40.19	51.57	61.51	50.33	50.37	65.15	66.42	38.69	65.45	65.45	45.76	52.00	51.32
HDGPCR11	35.75	50.22	39.93	51.50	61.67	50.30	50.34	65.25	66.51	38.73	65.53	65.53	46.37	51.32	48.90
HDGPCR12	35.13	50.33	40.31	51.92	61.33	50.39	50.44	65.05	66.30	38.81	65.33	65.33	46.73	50.71	48.86
HDGPCR13	33.82	50.48	40.05	51.81	61.32	50.54	50.58	65.31	66.53	38.87	65.58	65.58	46.26	51.32	52.23
HDGPCR14	37.36	50.21	40.04	51.41	61.73	50.30	50.35	65.28	66.57	38.73	65.58	65.58	45.8	51.91	50.30
HDGPCR15	34.48	50.75	40.17	51.81	61.34	50.81	50.86	65.36	66.59	39.08	65.64	65.64	46.34	50.97	50.95
HDGPCR16	37.03	50.12	40.12	51.63	61.52	50.19	50.23	65.33	66.60	38.70	65.61	65.61	46.07	51.84	50.40
HDGPCR17	34.91	50.39	40.39	51.61	61.46	50.47	50.51	65.20	66.47	38.81	65.49	65.49	45.58	52.07	52.19
HDGPCR18	34.13	50.24	39.77	51.57	61.66	50.29	50.34	65.48	66.71	38.81	65.74	65.74	46.57	51.18	53.41
HDGPCR19	37.45	50.74	39.94	51.84	61.55	50.79	50.84	65.38	66.60	39.06	65.65	65.65	45.84	51.87	50.22
HDGPCR20	33.59	49.97	40.11	51.45	61.32	50.06	50.10	65.17	66.44	38.54	65.48	65.48	46.14	51.56	53.64
HDGPCR21	34.34	50.23	40.02	51.67	61.47	50.30	50.35	65.14	66.39	38.67	65.42	65.42	46.42	50.83	49.81
HDGPCR22	35.93	50.11	40.29	51.48	61.45	50.20	50.25	65.07	66.36	38.67	65.38	65.38	46.37	51.43	49.42
HDGPCR23	33.53	50.13	40.08	51.50	61.47	50.20	50.24	65.15	66.37	38.68	65.43	65.43	45.7	52.13	53.72
HDGPCR24	37.11	50.24	40.22	51.55	61.49	50.32	50.37	65.28	66.53	38.67	65.57	65.57	46.47	51.12	53.11
HDGPCR25	36.18	50.01	40.06	51.56	61.58	50.08	50.12	64.99	66.24	38.63	65.27	65.27	45.83	51.9	51.81
HDGPCR26	35.79	50.61	40.18	52.06	61.39	50.65	50.70	65.23	66.45	38.97	65.48	65.48	46.05	51.86	50.92
HDGPCR27	33.92	49.99	40.04	51.62	61.51	50.06	50.10	65.03	66.28	38.58	65.31	65.31	45.77	51.71	50.31
HDGPCR28	35.35	50.01	40.06	51.32	61.52	50.09	50.14	65.19	66.45	38.63	65.48	65.48	46.35	51.34	51.52
HDGPCR29	34.42	50.61	40.00	51.64	61.38	50.67	50.71	65.64	66.88	38.96	65.91	65.91	46.58	50.97	52.11
HDGPCR30	34.39	49.96	40.20	51.59	61.42	50.03	50.08	65.11	66.39	38.54	65.40	65.40	45.99	51.39	48.59

TABLE 7. Comparison of HDGPCR with complex square root (radicals) and classic formula in all both single and multiple fault.

ID	HDGPCR	Dstar	GP02	GP03	GP19	Jaccard	Kulczynski	Ochiai	Ochiai2	Op2	SBI	Tarantula	Wong1	Wong2	Wong3
HDGPCR02	34.88	51.66	41.95	52.19	61.18	51.74	51.81	65.62	66.97	39.38	65.85	65.56	52.37	52.61	50.73
HDGPCR03	35.04	48.51	38.73	51.81	60.55	48.57	48.61	63.61	64.81	37.30	63.86	63.58	52.23	52.65	50.61
HDGPCR05	34.44	49.14	39.45	50.08	61.76	49.15	49.21	64.38	65.51	38.04	64.54	64.25	52.23	52.62	50.64
HDGPCR07	33.11	48.52	39.52	51.64	60.26	48.53	48.58	63.98	65.38	37.35	64.17	64.02	52.23	52.52	50.67
HDGPCR08	36.53	48.75	40.78	52.05	61.12	48.82	48.88	63.18	64.56	37.58	63.55	63.55	52.23	52.59	50.66
HDGPCR11	35.26	49.14	39.19	52.31	60.20	49.13	49.19	63.10	64.41	37.95	63.22	63.08	52.61	52.60	50.66
HDGPCR14	37.16	49.16	39.78	50.92	61.40	49.23	49.28	64.07	65.45	38.08	64.29	64.15	52.61	52.45	50.82
HDGPCR17	34.39	49.79	39.41	50.87	61.07	49.84	49.91	63.86	65.20	38.61	64.19	64.19	52.11	52.56	50.62
HDGPCR20	33.61	49.29	39.01	50.66	61.45	49.32	49.38	63.87	65.19	37.88	64.15	63.87	52.49	52.72	50.48
HDGPCR21	34.90	50.39	40.23	50.97	62.13	50.42	50.46	65.42	66.76	38.97	65.61	65.47	2.49	52.62	50.65
HDGPCR22	34.88	49.08	39.37	51.11	61.23	49.07	49.14	65.06	66.42	37.93	65.23	65.08	52.22	52.53	50.66
HDGPCR23	34.35	50.88	40.40	50.99	61.49	50.89	50.94	65.43	66.78	39.05	65.64	65.50	52.23	52.60	50.59
HDGPCR24	35.81	49.24	39.02	51.66	60.21	49.34	49.38	64.24	65.39	37.82	64.56	64.27	52.35	52.48	50.71
HDGPCR26	36.67	50.49	40.15	50.46	61.60	50.68	50.71	64.51	65.76	38.39	64.86	64.57	52.35	52.50	50.84
HDGPCR28	35.46	50.87	40.17	51.76	60.67	50.84	50.91	64.83	66.05	39.41	64.93	64.64	52.37	52.60	50.53

result compared to other formulae that has been studied in the previous studies. The performance of our method in multiple faults is twice of that Tarantula, GP19, Ochiai and SBI respectively. As it was mentioned in research question one, the lower the EXAM score, the better the metric for a spectrum based fault localization. The best metric for multiple fault localization according to the EXAM score result of table 6 is HDGPCR23 which has 33.53. The average performance of the formulae generated by our method compares with other classic formulae are; HDGPCR(35.36), DStar(50.31), GP02(40.11), GP03(51.66), GP19(61.48), Jaccard(50.38), Kulczynski(50.43), Ochiai(65.26), Ochiai2(66.51), Op2(38.78), SBI(65.54), Tarantula(65.54), Wong1(46.25), Wong2(51.43), Wong3(51.07). The table also shows that there are some formulae that have almost the same performance in multiple faults, like (DStar vs Kulczynski vs Jaccard), (Ochiai vs SBI vs Tarantula) and (Wong2 vs GP03 vs Wong3). Finally, the best three among the studied metrics are; HDGPCR, OP2 and GP02.

3) RQ3: EFFECTIVENESS OF COMPLEX ROOT (RADICALS)

This question addresses the importance of complex square roots otherwise known as radicals in the pool of the metrics generated by our method. Here, we combined both single and multiple evaluation faulty programs set in our study and then access the performance of each metric in the data set. We separated the EXAM scores of all our metrics that have

complex roots and then compared the result with the classic formulae.

Table 7 shows the performance of our metrics and the classic metrics in the combination of single and multiple faults. After we have generated the EXAM score of each metric compared with our metric, we separated the EXAM scores of those metrics with complex roots (radicals) from those formulae that do not have complex roots (radicals). The result shows that none of the classic formulae are able to outperform GP formulae that have complex roots generated by our method. This result implies that radicals are great tools in proposing formulae in the field of software testing for a spectrum based fault localization because none of the classic formulae outperformed any of the formulae with radicals.

However, the interpretation of the results in this article should be treated with caution. There is no assurance that the subject programs studied in this article will serve as a representative of all possible faults in real life and it is not clear whether they are legitimate samples of the entire group of faults. In the same way, it reduces the expense of designing risk evaluation formula through the use of GP. It also reduces the time to be spent in formulating formulae.

4) RQ4: STATISTICAL COMPARISON OF THE METRICS GENERATED BY OUR METHOD AND CLASSIC FORMULAE

There are many statistical tools that can be used to compare the performance of two or more methods in a given data set. The performance of the formula or method is measured using the test of unseen data sets. In addition, they are trained using η-fold cross-validation, which randomly partitions the data set into different folds. It is clear from the findings that our metrics perform much better than other classic formulae for these specific data sets. The question that might come up is: ''whether the results statistically provide convincing evidence that our method outperforms more than other classic formulae?''. In fact, we need to estimate whether the differences between the performances of the formulae are true and reliable, or they are just due to statistical chance. In addition to the mean values in our result, we are also interested in measuring the superiority, and relationship between our Metrics mean values and classic formulae mean values, because it represents the strength of a formula. We used two statistical tools in this comparison. The first is the Wilcoxon signed rank test which measures the superiority and significant difference in the performance of one formula over another. We used the paired version of the Wilcoxon Signed Rank test to compare the EXAM score values of the GP metrics generated by our method and classic metrics. The paired Wilcoxon Signed Rank test is a non-parametric test statistical hypothesis test that makes use of the sign and the magnitude of the rank of the differences between pairs of measurements, Exam(A) and EXAM(B) that do not follow a normal distribution [41]. At the given significant level, there is two-tailed p-value which can be used to obtain the conclusion. The smaller the p-value, the significance, the formula is; which means that more reliable and consistent performance can be expected. The second measure is the Cliff's delta effect size, which measures the size effect of the EXAM score generated by each metric in the study data set. For any given metrics A and B, if $\delta = 0$, that means the two metrics' performance are not different, if $\delta = 0.147$, that means there is small difference in the performance of the two metrics, if $\delta = 0.33$, there is a medium difference in the performance of the two metrics and if $\delta = 0.474$, there is a large difference in the performance of the two metrics.

Table 8 contains the interpretations of the hypothesis in the context of the current experiment. In this experiment, for a given metric A and B, the list of measures for Exam (A) would be the list of the EXAM score values for all the faulty programs produced by metric A, while the list of measurements for Exam (B) would be the list of Exam score values for all faulty programs B. For the two tailed p-value, if $p \geq \alpha$, the null hypothesis H_0 that says ''The number of statements examined by our metrics to locate all faults in both single and multiple faults programs $\geq$ the number of statements examined by classic metrics'' which means Exam (A) and Exam (B) are not significantly different is accepted; otherwise, the alternative hypothesis H_1 which says ''our metrics examined fewer statements than classic metrics before locating all faults in both single and multiple faults'' which means Exam(A) and EXAM(B) are significantly different is accepted.

In this manner, for all the faulty programs used in this experiment, we conducted Wilcoxon Signed Rank Test for each metric of the metrics generated by our method versus each classic metric. The metrics generated by our method are 30, while classic formulae are 14. We therefore have 420 pairs in this study.

Table 8 also contains Cliff's delta statistics that measure the effect sizes. If when calculated between Formula A and B, the value of $\delta = 0\ or 1.4\ or 1.33\ or 1.47$, then we say there is no difference in the performance of the two metrics or the difference is ''small'' or ''medium'' or ''large.'' The advantage of Cliff's delta statistic is that if EXAM score of metric A is smaller than the EXAM score of metric B, then the p-value of Cliff's delta will be negative, which signifies that metric A can perform better than metric B as EXAM score is very useful to judge the performance of metric in a spectrum based fault localization. The symbol '<' is used to denote if the p value is less than 0.05, '>' if p value is greater than 0.05 and '=' if p value is equal to 0.05. Therefore, table 8 shows that all metrics generated by our method performed better than all the classic formulae as the p values of each metric with individual HDGPCR is less than 0.05 against all the classic formulae, i.e. p<0.05. In summary of this table 8 Cliff's delta effect size; HDGPCR01-30 (except HDGPCR10 and HDGPCR11 which have medium difference) vs all classic metrics have 'large' different performance which means all HDGPCR metrics are better than the classic metric.

HDGPCR22 vs Ochiai have medium difference, which means that there are intermediate superiority between these two metrics.

TABLE 8. Performance comparison of Wilconxon (p = 0.05) and Cliff's delta of HDGPCR and classic formulae on both single and multiple faults.

HDGPCR	Dstar		GP02		GP03		GP19		Jaccard		Kulczynski		Ochiai		Ochiai2		Op2		SBI		Tarantula		Wong1		Wong2		Wong3	
ID	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ	p	δ
HDGPCR01	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR02	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR03	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR04	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR05	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR06	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR07	<	-1.00	<	-0.92	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.82	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR08	<	-0.59	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR09	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR10	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.28	<	-1.00	<	-1.00	<	-0.45	<	-1.00	<	-1.00	<	-1.00
HDGPCR11	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR12	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR13	<	-1.00	<	-1.00	<	-1.00	<	-0.97	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR14	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR15	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR16	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR17	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR18	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR19	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR20	<	-1.00	<	-1.00	<	-0.99	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.97	<	-1.00
HDGPCR21	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR22	<	-1.00	<	-0.99	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.32	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR23	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.63	<	-1.00	<	-1.00	<	-0.63	<	-1.00	<	-1.00
HDGPCR24	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR25	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.41	<	-1.00	<	-1.00	<	-0.90	<	-1.00	<	-1.00	<	-1.00
HDGPCR26	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.46	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR27	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR28	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-0.99
HDGPCR29	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00
HDGPCR30	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00	<	-1.00

HDGPCR10 vs Ochiai2 have medium difference, which means that there are also an average superiority between these two metrics.

HDGPCR11 vs OP2 vs Wong1 have very small differences, which means the three metrics have the capability of performing equally in some cases as there are very small differences in their performances.

V. THREATS TO VALIDITY

The main threat to internal validity may be caused by the data integrity of training and testing data. We used Genetic programming and pyevolve which is a python library to evolve metrics. We also used Scipy which is a python open source for our statistical analysis. Our fault separation from Defect4J repository was done through manual method which could be improved by automation method to avoid any data loss or mistakes in the future research.

Another threat to validity may be the generalizability of the data sets and experimental result. Despite the use of faults from open source projects, provided by Software-artifact Infrastructure Repository (SIR) and Defects4J, our conclusions might be limited as we combined two different programming languages (C and Java) as the subjects programs which could have effects on the individual metrics generated, as well as some unavoidable extemporaneous factors. Defects4J's data set across 6 programs with 395 sets of faults, but our study utilized only 5 programs with 357 sets of faults, written by different developers and targeting different application fields. Nonetheless, future research should verify whether the metrics generated in this article is generalized and applicable to other program structures and test suites.

The basic threat to construct validity is the utilization of three metrics namely, the EXAM score, the Wilcoxon Signed-rank test, and Cliff's delta effect size test. The EXAM score may not be the best metric for comparing the effectiveness of SBFL metrics by humans: in one study, expert programmers diagnosed faults more quickly with Fault Localization tools, but better EXAM scores did not always result in significantly faster debugging [25]. Our study revolves around the comparison of our generated metrics and classic metrics rather than their absolute performances. Other metrics may be better correlated with programmer performance, such as defective statements in the top-N, average wasted efforts and HIT. It is unknown which metrics are best for other uses of fault localization, such as automated program repair. Two of the three measures employed in this study have been used in previous studies like [11].

VI. DISCUSSION

The previous study has demonstrated that GP can be used to evolve the classic formula(s) which can be used in a spectrum based fault localization [18]. The previous research was able to generate 30 formulae, and among all the formulae generated, only few of them could perform as the classic formulae. Though the GP formulae generated in the research was able to outperform few of the formulae used in the experiment, but was not able to outperform OP2 which has been proved to be among the most effective formulae (if not the best) for single fault localization in a spectrum based fault localization. This is our motivations for this research to a further device a new method of instrumenting genetic programming to generate a classic formula(s) for a spectrum based fault localization if not the best.

The result of the experiment in this study shows that combining two repositories data set to evolve SBFL metrics is a good initiative in SBFL research field. It further suggests that complex square roots (radicals) are also good mathematical operands that can be used as Genetic programming operands when evolving metrics directly from the data set. In this article, four research questions are experimentally answered. Table 5 shows how effective HDGPCR in a single data set. The data set comprises 92 faults from SIR repository and 163 single faults from Defect4J repository. In [41], it is reported that GP02, 03 and 19 are classic formulae for single fault localization, therefore we compared the performance of HDGPCR against GP-classic and human design formula. Table 5 clearly shows that HDGPCR25 and HDGPCR30 are not able to outperform one of the human design formulae i.e. OP2 in single fault. This follows the suit in [41] which also conducted a cross comparison of both GP and human design formulae. In that study, OP2 outperformed all human and GP formulae used in the study. Also, in [18], OP2 outperformed all human design formulae and all the 30 GP generated formulae in the study. In our case, OP2 is able to outperform only two out of 30 GP generated formulae. This implies that we have successfully generated 28 GP formulae for SBFL that can compete and outperform OP2 in single fault programs. The best performance of our method's metrics can be seen in HDGPCR13 which will save the developer at least 75% search space before locating the faults in the program under test. These answers the research question 1 that the GP generated formulae by our method outperform the classic formulae in single fault programs.

Table 6 shows the performance of HDGPCR against GP-classic and human design classic formulae in multiple fault data set. We are convinced that GP is a good ideal tool to design a good formula for a spectrum based fault localization. Defect4J multiple faults are used to evaluate the performance of the formulae in multiple fault. It is observable that none of the human designed formulae as well as GP-classic formulae outperformed any of HDGPCR in multiple fault. We could also observe that HDGPCR13 which was outperformed by OP2 in single fault later outperformed OP2 in multiple fault. This implies that we have successfully proposed 30 SBFL formulae for multiple fault localization.

Table 7 compares the performance of HDGPCR metrics that have complex square root (radicals) in their trees against the classic formulae in all the data set in this study. To access the effectiveness of complex square root (radicals) in the metrics generated by our method, we combined both single and multiple faults in our experiment and then run an experiment to compare these metrics with the classic metrics. There are 449 faults in this experiment. We then extracted the EXAM scores of those metrics with radicals. Out of 30 GP metrics generated by our method, 15 of these metrics have complex square root (radicals). Table 7 then shows the comparison of these metrics with the classic metrics. The result in table 7 shows that none of the classic formulae are able to outperform any of the GP generated metrics that has a complex root (radicals). The practical implication is that the utilization of radicals in proposing metrics for a spectrum based fault localization is a welcome idea.

In the same manner, we answer our research question that says GP formulae generated with complex square roots from Hybrid data set performed better than the classic formulae in both single and multiple faults.

Finally, table 8 shows the statistical significance of our formulae over the previously studied formulae. It is observed that our formulae statistically outperformed previously studied classic formulae. We are able to come up with 29 new SBFL formulae that can outperform the classic formulae in both single and multiple faults.

VII. RELATED WORK

Debugging is the process of pinpointing and locating the exact statement that is not allowing a program to produce the expected output. Among the techniques employ to carry out this debugging process is a spectrum based Fault Localization.

a spectrum based Fault localization is also referred to as statistical fault location which its aim is to identify the statement that are suspected to contain the root cause of software failure by examining the passing and failing test execution. Many approaches have been used in a spectrum based fault localization and part of it is using Genetic programming to propound a good metric that can be used to rank the faulty statement. The very first research to use Genetic Programming to evolve formulae for a spectrum based fault localization was carried out in [18]. The paper evolved 30 formulae to compete with the human design formulae. The formulae were theoretically studied in [41], and 3 of the formulae, i.e. GP02, GP03 and GP19 were reported to be classic for a spectrum based fault localization.

Tarantula [16] was the first SBFL risk evaluation formula that originally started its life as a visualization tool. Many other formulae, followed by applying different statistical analysis to compute the ranking of the faulty statements. Tarantula calculates the frequency in which a program entity is executed in all failing test cases, divided by the frequency in which this program entity is executed in all failing and passing test cases. Tarantula has been used in several studies, like [16] which use Tarantula to calculate the suspiciousness score of statements for their parallel debugging technique.

Many studies have proposed different ranking metrics that are similar to Tarantula which used other coverage granularities. In [42], the paper reported a Tarantula-like metric which was evaluated on seven Java open source programs. The paper hypothesizes that information flows present a good model for such interactions and presents a new fault localization technique based on information flow coverage. Using a test suite, the technique ranks the statements in a program in terms of their likelihood of being faulty by comparing the information flows induced by the failing runs with the ones induced by the passing runs. The ranking of the statements associated with a given flow is primarily determined by contrasting

TABLE 9. GP-evolved risk evaluation formulae.

ID	Refined Formula	ID	Refined Formula
HDGPCR01	$\sqrt{(\sqrt{nf}-\frac{ep}{nf})-(\sqrt{np}+(nf)(ef))}$	HDGPCR16	$ep+2np*ef-nf$
HDGPCR02	$(\sqrt[5]{\sqrt[4]{np}}+3ef+np)-(\frac{ef}{ep}-2ef)(ef+np+\frac{np}{ep})$	HDGPCR17	$\sqrt[5]{ef+np-\frac{ef}{ep}+ep}$
HDGPCR03	$\sqrt[3]{\sqrt[3]{\frac{ep}{nf}}+((ef)(np)+ef)}$	HDGPCR18	$ef+\frac{np-ef}{2ep}+(ef*np)+ef$
HDGPCR04	$(\frac{np}{(ep)(ef)}-np)-((ef)(np)+np)$	HDGPCR19	$\sqrt{ep}+(np)(ef)$
HDGPCR05	$\sqrt[3]{(\frac{(np+ef)(nf-ef)+(np)(nf)}{ef-ep}}$	HDGPCR20	$\sqrt[5]{\sqrt{(ep)(\frac{np}{ep})}}-\sqrt[3]{nf}-\frac{np}{ef}+(nf-ef-\sqrt{(ep)(np)})$
HDGPCR06	$(\frac{np^3}{ep}+\sqrt{(np)^2(ef)}$	HDGPCR21	$\frac{(np)(ep+ef)}{ef}-\frac{\sqrt[3]{nf}}{ep+ef}+ef$
HDGPCR07	$\frac{\sqrt[5]{\frac{np}{\sqrt{ef}}}}{\sqrt[3]{\sqrt[4]{ep}+\sqrt{ep}}}$	HDGPCR22	$\frac{\sqrt[3]{\frac{nf}{np}}}{nf-ep^2}-\frac{ef}{ep}-np-(ef-nf)(\frac{np}{nf})$
HDGPCR08	$\sqrt[5]{\frac{2ef+np}{ef+(ep)(nf)}}$	HDGPCR23	$\frac{\sqrt{ep}+\frac{np}{ef}}{(ep+ef+nf)}-(nf)(\frac{ef}{ep})+(\frac{\sqrt[3]{nf}}{np})(nf)$
HDGPCR09	$-\sqrt{np}+(ef+(np)(ef)+\frac{np^2}{ep})$	HDGPCR24	$\sqrt[3]{\sqrt[5]{(np)(\sqrt{ef})}}+\sqrt[5]{\frac{ep}{np}}$
HDGPCR10	$\frac{np^2}{(nf)(ep)+(ef-nf)}$	HDGPCR25	$(\sqrt[3]{ep}-np)(\frac{\sqrt[4]{np}-np}{\sqrt{ef}})$
HDGPCR11	$\frac{\sqrt[4]{\frac{ep}{ef}}}{(ep-ef)-\frac{ep}{ef}}+ef$	HDGPCR26	$\sqrt{np}+\sqrt[4]{\frac{(np)(ep)}{ep-ef}}$
HDGPCR12	$\sqrt{\frac{(ep)(2ef)}{np}+(ef)(np)}$	HDGPCR27	$(ef+np)(ef)+\sqrt{ep}+\frac{np}{ep}+ef$
HDGPCR13	$\frac{(\frac{np}{ep}+ef^2)(\frac{ep}{np}+(ep)(nf))}{\frac{ep+1}{\sqrt[4]{np^2}}}$	HDGPCR28	$\sqrt[5]{\sqrt[4]{np}}-(\sqrt{nf}-\sqrt{ef})(\sqrt{np}+2ef)$
HDGPCR14	$\sqrt[4]{ef+\frac{ef}{nf}+np}$	HDGPCR29	$(ef^2-np)(ef)+\sqrt{ep}+\frac{np}{ep}+ef$
HDGPCR15	$\frac{\sqrt{ef+ep}}{((ep)(ef)+\frac{ep}{nf})+(nf+np)+(ep)(nf)}$	HDGPCR30	$\sqrt{(ep)(np)+\frac{np}{ep}+\frac{ef-nf}{np}}$

the percentage of failing runs to the percentage of passing runs that induced it. Generally, a higher percentage of failing runs implies a higher rank. Furthermore, [43] use a ranking metric similar to Tarantula for basic blocks and [44] propose a ranking metric for predicates that is also similar to Tarantula.

In addition to Tarantula, other techniques based on similarity formulae have been proposed in the last years. A paper used Ochiai and Jaccard similarity coefficients to propose fault localization ranking metrics [6]. Ochiai was originally used in molecular biology, and Jaccard was used in [43] to indicate faulty components in distributed applications. [6] show that both Ochiai and Jaccard outperform Tarantula on fault localization effectiveness. From the outcome of this article, several works have used Ochiai [45], [46].

In [20], the paper proposes two new ranking metrics optimized for single-fault programs, called O and Op. Assuming the existence of a single fault, only statements that are executed in all failing test cases are fault candidates. Kulczynski2 (from Artificial Intelligence) and McCon (from studies of plankton communities) are ranking metrics that have presented better results for programs with two simultaneous faults [47]. DStar technique was also presented in [48], which is a modified version of the Kulczynski ranking metric. The intuition behind DStar is that the execution trace of each statement through test cases can be viewed as an execution pattern. Hence, the similarity between statements more frequently executed by failing test cases can pinpoint the faulty ones.

Some techniques propose different strategies in conjunction with ranking metrics like [20] which presented an approach that assigns different weights to statements in failing test cases according to the number of statements in an execution. The lower, the number of commands, the greater, the chance that one of them will be faulty. [6], [20], [21], [48], [49], all which have been human design risk formulae. For the purpose of improving the Genetic Programming to evolve a risk evaluation formula and to employ hybrid data set, this article present the second GP-based approach to design risk evaluation formulae usable to locate single and multiple fault in a spectrum based fault localization.

Other methods that have been employed in fault localization includes slicing [8], test similarities [29], [50], delta debugging [51], [52], and causal inference [53]. This article only concerns with spectra-based approach. The positive outcome of this research suggests that GP can be employed to evolve a wider range of spectral data for fault localization.

VIII. CONCLUSION AND FUTURE WORK

SBFL has received a significant amount of attention over the past decade. The technique aids debugging by ranking program statements according to the likelihood of being faulty. SBFL depends on the risk evaluation formula to convert program a spectrum data into risk scores. The main focus of this article is to design new risk evaluation formulae that can outperform the existing ones. The recent literature on proposing metric for fault location through genetic programming have empirically shown that GP-formulae can outperform or like human design formula. The research generated 30 formulae, and 3 have been theoretically proved to be classic. This article also presents an empirical study of genetic programming to evolve a formula from hybrid data sets with the inclusion of complex square roots which are otherwise known as radicals.

Radicals are great mathematical operands that have successfully been employed in many fields to generate a functioning and useful formula for different purpose. Hybrid data sets have not been employed in any research to evolve formula, and by this virtue, to the best of our knowledge, this is the first of its kind. Many of human design formulae and the GP-generated classic formulae were studied from single fault and small program benchmarks. We empirically showed that GP-generated formula with complex square root from hybrid data sets can outperform the classic formulae even in both single and multi-fault subject programs. The proof that hybrid data sets is a good technique to evolve classic formulae for a spectrum based fault localization has implications and actionable conclusion of researchers. It shows that the human competitiveness of formulae formulation that will work in both single and multiple fault has been proven to be accomplished as the formulae generated from hybrid data sets can help the developer to rank the faulty statement(s) high to reduce the search space in debugging. It also shows that the human competitiveness of GP has been proven to have accomplished what has been the aim of SBFL researchers for over a decade. This induces that GP will remain an effective methodology to develop competitive technique for a spectrum based fault localization.

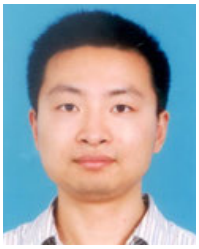
Our future work will focus on accessing the performance of our metrics in other granularity level aside statement level. We are also interested to know if there would be a different performance in GP formula generated in different granularity using the same data set repositories.

REFERENCES

[1] S. Planning, "The economic impacts of inadequate infrastructure for software testing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Planning Rep. 02-03, 2002.
[2] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.
[3] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon, and C. Zander, "Debugging from the student perspective," *IEEE Trans. Educ.*, vol. 53, no. 3, pp. 390–396, Aug. 2010.
[4] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Trans. Rel.*, vol. 63, no. 1, pp. 290–308, Mar. 2014.
[5] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward, "Efficient fault detection and diagnosis in complex software systems with information-theoretic monitoring," *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 4, pp. 510–522, Jul. 2011.
[6] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1780–1792, Nov. 2009.
[7] T. Janssen, R. Abreu, and A. J. C. V. Gemund, "Zoltar: A toolset for automatic fault localization," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.*, Nov. 2009, pp. 662–664.
[8] H. Agrawal, J. R. Horgan, S. London, and W. E. Wong, "Fault localization using execution slices and dataflow tests," in *Proc. 6th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 1995, pp. 143–151.
[9] H. Cleve and A. Zeller, "Locating causes of program failures," in *Proc. 27th Int. Conf. Softw. Eng. (ICSE)*, May 2005, pp. 342–351.
[10] M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," in *Proc. 18th IEEE Int. Conf. Automated Softw. Eng.*, Oct. 2003, pp. 30–39.
[11] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proc. 20th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2005, pp. 273–282.
[12] C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff, "Statistical debugging: A hypothesis testing-based approach," *IEEE Trans. Softw. Eng.*, vol. 32, no. 10, pp. 831–848, Oct. 2006.
[13] F. DeMarco, J. Xuan, D. Le Berre, and M. Monperrus, "Automatic repair of buggy if conditions and missing preconditions with SMT," in *Proc. 6th Int. Workshop Constraints Softw. Test., Verification, Anal. (CSTVA)*, 2014, pp. 30–39.
[14] A. Ochiai, "Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions," *Bull. Jpn. Soc. Sci. Fisheries*, vol. 22, pp. 526–530, Jul. 1957.
[15] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight bug localization with AMPLE," in *Proc. 6th 6th Int. Symp. Automated Anal.-Driven Debugging (AADEBUG)*, 2005, pp. 99–104.
[16] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proc. 24th Int. Conf. Softw. Eng. (ICSE)*, 2002, pp. 467–477.
[17] W. E. Wong, Y. Qi, L. Zhao, and K.-Y. Cai, "Effective fault localization using code coverage," in *Proc. 31st Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2007, pp. 449–456.
[18] S. Yoo, "Evolving human competitive spectra-based fault localisation techniques," in *Proc. Int. Symp. Search Based Softw. Eng.* Berlin, Germany: Springer, 2012, pp. 244–258.
[19] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang, "An empirical study of fault localization families and their combinations," *IEEE Trans. Softw. Eng.*, early access, Jan. 10, 2019, doi: 10.1109/TSE.2019.2892102.
[20] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–32, Aug. 2011.
[21] R. Abreu, P. Zoeteweij, and A. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *Proc. 12th Pacific Rim Int. Symp. Dependable Comput. (PRDC)*, 2006, pp. 39–46.
[22] Software-artifact Infrastructure Repository. (2006). *Software-artifact Infrastructure Repository*. [Online]. Available: https://sir.csc.ncsu.edu/portal/index.php
[23] B. LeTien-Duy and L. ThungFerdian, "Should I follow this fault localization tool's output?" *Empirical Softw. Eng.*, vol. 20, pp. 1237–1274, Nov. 2015.
[24] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria," in *Proc. 16th Int. Conf. Softw. Eng.*, 1994, pp. 191–200.
[25] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 609–620.
[26] G. Rothermel, S. Elbaum, A. Kinneer, and H. Do. (2006). *Software-Artifact Infrastructure Repository*. [Online]. Available: http://sir.unl.edu/portal
[27] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for java programs," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, 2014, pp. 437–440.
[28] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, pp. 1–40, Oct. 2013.
[29] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, "Directed test generation for effective fault localization," in *Proc. 19th Int. Symp. Softw. Test. Anal. (ISSTA)*, 2010, pp. 49–60.
[30] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the accuracy of spectrum-based fault localization," in *Proc. Testing: Academic Ind. Conf. Pract. Res. Techn., MUTATION (TAICPART-MUTATION)*, Sep. 2007, pp. 89–98.
[31] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," *ACM SIGPLAN Notices*, vol. 40, no. 6, pp. 15–26, Jun. 2005.
[32] J. Xuan and M. Monperrus, "Test case purification for improving fault localization," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, 2014, pp. 52–63.
[33] J. Xuan and M. Monperrus, "Learning to combine multiple ranking metrics for fault localization," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Sep. 2014, pp. 191–200.
[34] T. Wu, Y. Dong, M. F. Lau, S. Ng, T. Y. Chen, and M. Jiang, "Performance analysis of maximal risk evaluation formulas for spectrum-based fault localization," *Appl. Sci.*, vol. 10, no. 1, p. 398, Jan. 2020.
[35] S. Park, R. W. Vuduc, and M. Jean Harrold, "Falcon: Fault localization in concurrent programs," in *Proc. ACM/IEEE 32nd Int. Conf. Softw. Eng.*, vol. 1, May 2010, pp. 245–254.

[36] Wikipedia. *nth Root*. [Online]. Available: https://en.wikipedia.org/wiki/Nth_root
[37] Sciencing. (Jun. 25, 2018). *How Radical Expression Works*. [Online]. Available: http://sciencing.com/how-are-radical-expressions-rational-exponents-used-in-real-life-12751906.html
[38] L. Garrett-Hatfield. *How Meteorologists Use Simplifying Radicals*. SeatlePi. [Online]. Available: https://education.seattlepi.com/meteorologists-use-simplifying-radicals-7048.html
[39] C. S. Perone, "Pyevolve: A Python open-source framework for genetic algorithms," *ACM SIGEVOlution*, vol. 4, no. 1, pp. 12–20, 2009.
[40] M. R. Hess and J. D. Kromrey, "Robust confidence intervals for effect sizes: A comparative study of Cohen'sd and Cliff's delta under non-normality and heterogeneous variances," in *Proc. Annu. Meeting Amer. Educ. Res. Assoc.*, 2004, pp. 1–30.
[41] S. Yoo, X. Xie, F.-C. Kuo, T. Y. Chen, and M. Harman, "Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 1, pp. 1–30, 2017.
[42] W. Masri, "Fault localization based on information flow coverage," *Softw. Test., Verification Rel.*, vol. 20, no. 2, pp. 121–147, May 2009.
[43] X. Wang, Q. Gu, X. Zhang, X. Chen, and D. Chen, "Fault localization based on multi-level similarity of execution traces," in *Proc. 16th Asia–Pacific Softw. Eng. Conf.*, Dec. 2009, pp. 399–405.
[44] Y.-M. Chung, C.-Y. Huang, and Y.-C. Huang, "A study of modified testing-based fault localization method," in *Proc. 14th IEEE Pacific Rim Int. Symp. Dependable Comput.*, Dec. 2008, pp. 168–175.
[45] N. DiGiuseppe and J. A. Jones, "Fault density, fault types, and spectra-based fault localization," *Empirical Softw. Eng.*, vol. 20, no. 4, pp. 928–967, Aug. 2015.
[46] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold, "Lightweight fault-localization using multiple coverage types," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, May 2009, pp. 56–66.
[47] L. Naish, H. J. Lee, and K. Ramamohanarao, "Spectral debugging with weights and incremental ranking," in *Proc. 16th Asia–Pacific Softw. Eng. Conf.*, Dec. 2009, pp. 168–175.
[48] W. E. Wong, V. Debroy, Y. Li, and R. Gao, "Software fault localization using DStar (D*)," in *Proc. IEEE 6th Int. Conf. Softw. Secur. Rel.*, Jun. 2012, pp. 21–30.
[49] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull. Soc. Vaudoise Sci. Nat.*, vol. 37, pp. 547–579, 1901.
[50] D. Hao, L. Zhang, Y. Pan, H. Mei, and J. Sun, "On similarity-awareness in testing-based fault localization," *Automated Softw. Eng.*, vol. 15, no. 2, pp. 207–249, Jun. 2008.
[51] A. Zeller, "Automated debugging: Are we close," *Computer*, vol. 11, pp. 26–31, 2001.
[52] A. Zeller, *Why Programs Fail: A Guide to Systematic Debugging*. Amsterdam, The Netherlands: Elsevier, 2009.
[53] G. K. Baah, A. Podgurski, and M. J. Harrold, "Causal inference for statistical fault localization," in *Proc. 19th Int. Symp. Softw. Test. Anal. (ISSTA)*, 2010, pp. 73–84.

ADEKUNLE AKINJOBI AJIBODE was born in Owode Egbado, Ogun, Nigeria. He received the degree in computer science (Ed.) from the Tai Solarin University of Education, Ijagun, in 2014. He is currently pursuing the master's degree in computer science and technology with Zhejiang Sci-Tech University, Hangzhou. He has taught computer science and the related courses at both postsecondary and secondary level of education from 2016 to 2019. His research interests include software testing and business and economy aspect of software development process.



TING SHU was born in July 1979. He received the Ph.D. degree in computer science from Zhejiang University, in 2010. He is currently an Associate Professor with the School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou, China. His current research interests include software testing and network protocol testing.

ZUOHUA DING (Associate Member, IEEE) received the M.S. degree in computer science and the Ph.D. degree in mathematics from the University of South Florida, Tampa, FL, USA, in 1996 and 1998, respectively. He is currently a Professor and the Director of the Laboratory of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, China. He has authored or coauthored over 70 articles. His current research interests include system modeling, software reliability prediction, intelligent software systems, and service robots.