Human Competitiveness of Genetic Programming in Spectrum-Based Fault Localisation: Theoretical and Empirical Analysis

SHIN YOO, Korea Advanced Institute of Science and Technology, Republic of Korea
XIAOYUAN XIE, Wuhan University, China
FEI-CHING KUO and TSONG YUEH CHEN, Swinburne University of Technology, Austrailia
MARK HARMAN, University College London, UK

We report on the application of Genetic Programming to Software Fault Localisation, a problem in the area of Search-Based Software Engineering (SBSE). We give both empirical and theoretical evidence for the human competitiveness of the evolved fault localisation formulæ under the single fault scenario, compared to those generated by human ingenuity and reported in many papers, published over more than a decade. Though there have been previous human competitive results claimed for SBSE problems, this is the first time that evolved solutions have been formally proved to be human competitive. We further prove that no future human investigation could outperform the evolved solutions. We complement these proofs with an empirical analysis of both human and evolved solutions, which indicates that the evolved solutions are not only theoretically human competitive, but also convey similar practical benefits to human-evolved counterparts.

4

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

Additional Key Words and Phrases: Spectrum-based fault localisation, search-based software engineering, genetic programming

ACM Reference Format:
Shin Yoo, Xiaoyuan Xie, Fei-Ching Kuo, Tsong Yueh Chen, and Mark Harman. 2017. Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis. ACM Trans. Softw. Eng. Methodol. 26, 1, Article 4 (June 2017), 30 pages.
DOI: http://dx.doi.org/10.1145/3078840

1. INTRODUCTION

This article presents a theoretically optimal, human competitive, and practical approach to Spectrum-Based Fault Localisation (SBFL) [24, 28] using Genetic Programming (GP) [31, 43]. Our work is situated within a growing trend in software engineering, Search-Based Software Engineering (SBSE) [4, 16, 23, 35], which uses computational search techniques (with a particular emphasis on evolutionary computation [22]). It provides the first provably and theoretically optimal results in the field of SBSE.

This work is supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (Grant No. 2016R1C1B1011), as well as the National Natural Science Foundation of China (Grant No. 61572375).
Authors' addresses: S. Yoo (co-corresponding author), Korea Advanced Institute of Science and Technology, School of Computing, 291 Daehak Ro, Yuseong Gu, Daejeon 34141, Republic of Korea; email: shin.yoo@kaist.ac.kr; X. Xie (co-corresponding author), Wuhan University, Computer School, Wuchang, Wuhan, Hubei, 430072, China; email: xxiw@whu.edu.cn; F.-C. Kuo and T. Y. Chen, Swinburn University of Technology, Hawthorn, VIC 3122, Austrailia; emails: {dkuo, tychen}@swin.edu.au; M. Harman, Facebook London, 10 Brock Street London and University College London, Gower Street, London WC1E 6BT, United Kingdom; email: mark.harman@ucl.ac.uk.

ACM 1049-331X/2017/06-ART4 $15.00
DOI: http://dx.doi.org/10.1145/3078840

SBFL is important, because it offers automated assistance to the debugging process, which is currently labour-intensive, expensive, and time-consuming. SBFL has been advocated as a technique for helping humans find faults faster [20, 42] and also as a supporting technology for automated program repair [32, 50], which automatically fixes certain classes of fault (also using techniques such as GP).

The SBFL *suspiciousness formula* defines the "suspiciousness" of each statement in terms of observations from software testing, thereby forming the "key ingredient" of SBFL. The suspiciousness formula is also known as a *risk formula*, in the sense that it seeks to capture the "risk" that the statement causes the bug. A good risk formula will tend to elevate the reported suspiciousness of truly faulty statements and depress that of innocent statements. However, it is far from obvious how to define such a good risk formula. There has been a great deal of previous work on SBFL, much of which focuses on designing and empirically evaluating different formulæ [1, 9, 30, 51].

We report on a GP solution that searches for formulæ, which we have implemented, showing that it finds known maximal formulæ (previously found by humans) and also novel maximal formulæ (not previously found by humans). We report on a set of experiments on real software systems to evaluate the formulæ found by humans and by GP. Our empirical evaluation indicates that one class of formulæ (found by GP and also by humans) performs best overall. Finally, we prove that, under the single fault scenario, there does not exist a superior formula to the current known maximal formulæ found by humans and/or by GP. Therefore, GP-evolved formulæ are not only human competitive, but no further human analysis could yield superior alternatives. While human competitiveness of SBSE has been empirically shown before [7, 8, 15, 40], we believe this is the first claim backed by a formal mathematical proof.

SBFL is an area of software engineering that has been well studied by humans over many years, and for which human ingenuity has produced publishable advances that have subsequently turned out to include both sub-optimal as well as optimal results. It is an important area that has motivated (and continues to motivate) many leading researchers to attack the problem of finding suitable formulæ with attractive theoretical and practical properties. We believe that this makes it exciting and encouraging that GP has been able to find results that are provably human competitive, theoretically unbeatable, and also practically valuable.

2. BACKGROUND

In this section, we present the SBFL problem (Section 2.1) and summarise the previously published theoretical underpinning framework that we use to construct our proofs (Section 2.2).

2.1. Problem Statement

SBFL refers to a group of techniques that use program spectrum to find the location of the fault in the given program that causes certain tests to fail. Program spectrum can be best described as a summary of a set of program executions [24]. For the SBFL techniques, the most widely used type of program spectrum is the combination of code coverage and the test results, on which this article focuses too. Suppose the System Under Test (SUT) has n statements, and the test suite contains m test cases: the program spectrum for SBFL can be described as a matrix of n rows and 4 columns. Each row corresponds to individual statement of the SUT and contains the tuple (e_f, e_p, n_f, n_p). Members e_f and e_p represent the number of times the corresponding program statement has been executed by tests, with fail and pass as a result, respectively. Similarly, n_f and n_p represent the number of times the corresponding program statement has *not*

Table I. Motivating Example: The Faulty Statement s_7 Achieves the First Place when Ranked According to the Tarantula Risk Evaluation Formula in Equation 1

Structural Elements	Test t_1	Test t_2	Test t_3	Spectrum e_f	n_f	e_p	n_p	Tarantula	Rank
s_1	•	•	•	2	0	1	0	0.00	2
s_2	•	•	•	2	0	1	0	0.00	3
s_3	•			0	2	1	0	0.00	7
s_4	•			0	2	1	0	0.00	8
s_5	•			0	2	1	0	0.00	9
s_6	•		•	1	1	1	0	0.33	5
s_7 (faulty)		•	•	2	0	0	1	1.00	1
s_8	•	•		1	1	1	0	0.33	6
s_9	•	•	•	2	0	1	0	0.50	4
Result	P	F	F						

been executed by tests, with fail and pass as a result, respectively[1]:

$$\text{Tarantula} = \frac{\frac{e_f}{e_f+n_f}}{\frac{e_p}{e_p+n_p} + \frac{e_f}{e_f+n_f}}. \tag{1}$$

SBFL techniques subsequently use a risk evaluation formula, which is a formula based on the four counters, to assign risk scores to statements: the scores are designed to correlate to the relative risk of each statement containing the fault. Table I presents an illustrative example of the Tarantula[2] metric [28], shown in Equation (1), being applied to a SUT with 9 structural elements. Let us assume that the element s_7 is the faulty one, which causes test case t_2 and t_3 to fail, whereas test case t_1 passes. The second column presents the coverage achieved by these three test cases, respectively. The 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. For example, Tarantula assigns the risk score $\frac{\frac{2}{2+0}}{\frac{2}{2+0}+\frac{1}{1+0}} = 0.5$ to s_1, and the score $\frac{\frac{2}{2+0}}{\frac{2}{2+0}+\frac{0}{0+1}} = 1.0$ to s_7. 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 seventh element when inspected following the line number order. Note that the tie breaker is the line number, which is completely independent from the SBFL technique used (please refer to Definition 2.2 in Section 2.2 for more details about tie breakers).

More formally, a SBFL risk evaluation formula is a function from program spectrum to suspiciousness score, such as Tarantula in Equation (1), defined as follow:

Definition 2.1. A risk evaluation formula R is a member of set $\mathcal{F} = \{R | R : I \times I \times I \times I \rightarrow Real\}$ (where I denotes the set of non-negative integers and *Real* denotes the set of real numbers), which maps $A_i = < e^i_f, e^i_p, n^i_f, n^i_p >$ of each statement s_i to its risk value.

Effectiveness of an SBFL significantly depends on the design of the risk evaluation formulæ, of which the literature provides a rich pool. Most of these formulæ have been designed manually. Jaccard [25] and Ochiai [39] were first studied in Botany and Zoology respectively but have been subsequently studied in the context of fault

[1] The sum of e_f, e_p, n_f, and n_p should be m.

[2] Note that this example as well as the choice of the Tarantula metric is purely illustrative.

localisation [3, 37]. Tarantula [28–30, 41], AMPLE [11], Wong formulæ [51], and Naish formulæ [37] have all been designed by software engineers.

In addition to designing new formulæ, much effort was spent on evaluating the existing formulæ. Most of the evaluation was performed empirically, by applying these formulæ to localise a set of known faults in a controlled environment [2, 3, 28]. Beyond comparing different formulæ, others investigated their relationship with external factors, such as test suites and program structures. Yu et al. studied the impact of test suite reduction on the accuracy of fault localisation [57]. Heo et al. considered the homogeneity of test suite in terms of coverage, highlighting that test cases with similar coverage patterns provide little additional information to localisation [21]. Xu et al. sought to reduce the noise, that is, structural element that are happened to be executed simultaneously with the faulty statement [54]. Artzi et al. studied ways to augment the test suite to help the localisation [6], while DiGiuseppe et al. considered the impact of having multiple faults on the accuracy of formulæ [12].

The aforementioned evaluation of SBFL formulæ has been largely dependent on the Expense metric. Expense metric assumes that the human developer inspects the ranked statements in the descending order of their risk scores. The metric measures the portion of the program that the test engineer has to inspect before the fault is localised:

$$\text{Expense} = \frac{\text{Ranking of the faulty statement}}{\text{Number of statements in the program}}. \tag{2}$$

The metric itself assumes a specific mode of usage of the results, that is, linear and manual inspection of the ranked statements. Parnin and Orso questioned whether this approach is really helpful to test engineers [42], highlighting the need to focus on absolute ranks rather than relative measure such as the Expense metric. Gouveia et al., on the other hand, reported that automated fault localisation technique has improved developers' capability to efficiently debug faults [20]. It should be noted that SBFL techniques are increasingly being used by other, automated algorithms such as automated program repair [14, 50] and failure reproduction [26, 27]. Qi et al. evaluated the performance of different SBFL techniques in the context of automated program repair and reported that relative performance based on the Expense metric did not hold when SBFL techniques are applied to program repair [44]. Later, Moon et al. suggested a new evaluation metric, called Locality Information Loss, based on cross entropy between the actual and the predicted location of the fault [36]. Other work investigated how quickly localisation can be achieved. Yoo et al. considered an information theoretic approach towards selecting the test case that will yield the maximum amount of information regarding the locality of the fault [56], whereas Gonzalez-Sanchez et al. studied the impact of test prioritisation on fault localisation [17–19]. Finally, Steimann et al. discussed various threats to validity relevant to empirical SBFL research [48].

2.2. Theoretical Foundations

Recently SBFL formulæ has been analysed, not only empirically, but also theoretically. Abreu et al. [1] proposed a model-based method and proved, for the first time, that the following formula, R, is equivalent to one optimal formula under the single fault scenario:

$$R = \begin{cases} \frac{e_f}{e_f+e_p} & \text{if } e_f = F \\ MIN & \text{if } e_f < F \end{cases}.$$

Later, the Naish1 and Naish2 formulæ were designed with an accompanying proof, which shows they produce optimal ranking, as long as the fault is located in a specific program structure (two consecutive `If-Then-Else` blocks, called `ITE2`) [37].

Subsequently, Naish et al. posited that, ranking-wise, these formulas are optimal in a single-fault scenario [38]: while much of this analysis aligns with this article, we show that there are single-fault counter-examples for which Naish formulæ are not optimal.

Steimann et al. considered the lower and upper bounds of the localisation, in addition to outlining potential threats to validity when studying SBFL formulæ [Steimann et al. 48]. Intuitively, to be useful, an SBFL technique should produce the ranking of the faulty statement with a lower bound that is higher than $\frac{n-1}{2} - 1$; otherwise, a random order inspection of statements will have a better average performance. Steimann et al. speculated that the upper bound of the ranking produced by an SBFL formula will be specific to each combination of a program and a test suite. What this article proves is that, even if it is possible for a formula to reach the upper bound for a single fault, the same formula will not always be as effective for other faults.

Xie et al. presented a comprehensive theoretical framework that can show equivalence and hierarchy between 50 of known formulæ with respect to the Expense metric (i.e., the ranking) [52]. We briefly review the existing theoretical framework of Xie et al. [52] here, to make the article and its theoretical contributions self-contained. The framework has been used to show equivalence or dominance between different formulæ, with respect to the Expense metric, against any combinations of programs, test suites, and faulty statements. The existing theoretical framework makes several assumptions, which are listed as follows:

1. We focus on the single fault localisation, that is, we assume that there exists a single faulty statement in the program.
2. A test oracle exists, that is, for any test case, the testing result of either *fail* or *pass*, can be decided.
3. The fault is executed by the test suite. Being a type of dynamic analysis, SBFL techniques cannot localise faults in statements that are either not covered by the test suite, or even missing from the program (i.e., omission faults).
4. We exclude the non-deterministic faults so the relationships we prove to exist between SBFL formulæ hold regardless of the choice of specific test executions.[3]
5. For each fault that needs to be localised, the test suite contains at least one passing and one failing test case.

Note that these assumptions are shared by this article. For readers who are interested in the justifications, validity and impacts of the above assumptions, please refer to the previous work [52].

SBFL techniques seek to rank program statements in the order of the likelihood of being faulty. In practice, a tie-breaking scheme may be required to determine the order of the statements that share the same risk scores. The consistent tie-breaking scheme is defined as follows:

Definition 2.2 (*Consistent Tie-Breaking Scheme*). Given any two sets of statements S_1 and S_2, which contain elements having the same risk values (see Definition 2.1). A tie-breaking scheme returns the ordered statement lists O_1 and O_2 for S_1 and S_2, respectively. The tie-breaking scheme is said to be consistent, if all elements common to S_1 and S_2 have the same relative order in both O_1 and O_2.

Now let us turn to the relationships between two formulæ. Let R_1 and R_2 be two risk evaluation formulæ in $\mathcal{F}$, and E_1 and E_2 denote the Expenses with respect to the same faulty statement for R_1 and R_2, respectively. We define two types of relations between R_1 and R_2 as follows.

[3]However, this does not mean that SBFL cannot be applied to non-deterministic faults.

Definition 2.3 (*Better*). R_1 is said to be *better* than, or to *dominate*, R_2 (denoted as $R_1 \rightarrow R_2$) if, for any program, faulty statement s_f, test suite, and consistent tie-breaking scheme, E_1 is less than or equal to E_2.

Definition 2.4 (*Equivalent*). R_1 and R_2 are said to be *equivalent* (denoted as $R_1 \leftrightarrow R_2$), if, for any program, faulty statement s_f, test suite and consistent tie-breaking scheme, E_1 is equal to E_2.

It follows from the definition that $R_1 \rightarrow R_2$ means R_2 is not more effective than R_1. As a reminder, if both $R_1 \rightarrow R_2$ and $R_2 \rightarrow R_1$ hold, then it follows that $R_1 \leftrightarrow R_2$; if $R_1 \rightarrow R_2$ holds but $R_2 \rightarrow R_1$ does not hold, $R_1 \rightarrow R_2$ is said to be a strictly "*better*" relation. Here, the notion of the better relation aims to be applicable to any combination of subject programs, test suites, and faults: consequently, it is a conservative concept. In practice, it is entirely possible that a better relation based on statistical significance exists; however, that lies beyond the scope of this work.

In order to compare two risk evaluation formulæ in $\mathcal{F}$ under the above definitions of relations, the previous work [52] have provided a theoretical framework, which divides all statements into three mutually exclusive subsets, as follows.

Definition 2.5. Given a program with n statements $PG =< s_1, s_2, \ldots, s_n >$, a test suite of m test cases $TS = \{t_1, t_2, \ldots, t_m\}$, and a risk evaluation formula R, which assigns a risk value to each program statement. For each statement s_i, a spectrum vector $\sigma(s_i) =< e^i_f, e^i_p, n^i_f, n^i_p >$ can be constructed from TS, and $R(e^i_f, e^i_p, n^i_f, n^i_p)$ is a risk evaluation formula that assigns a risk value to statement s_i. For any faulty statement s_f, it is possible to define the following three sets of statements (note that $1 \le i \le n$):

$$S^R_B = \left\{s_i \in S | R(e^i_f, e^i_p, n^i_f, n^i_p) > R\left(e^f_f, e^f_p, n^f_f, n^f_p\right)\right\},$$
$$S^R_F = \left\{s_i \in S | R(e^i_f, e^i_p, n^i_f, n^i_p) = R\left(e^f_f, e^f_p, n^f_f, n^f_p\right)\right\},$$
$$S^R_A = \left\{s_i \in S | R(e^i_f, e^i_p, n^i_f, n^i_p) < R\left(e^f_f, e^f_p, n^f_f, n^f_p\right)\right\}.$$

That is, statements in S^R_B have higher risk values than s_f, and thus are all ranked above any statements in S^R_F; statements in S^R_F have the same equal risk value as that of s_f and, thus, are all ranked in the middle of the ranking list, together with s_f (tie-breaking scheme is needed to further distinguish them); and statements in S^R_A have lower risk values than s_f and, thus, are all ranked below any statements in S^R_F.

In the current framework, two results have been developed for establishing the relationship between two risk evaluation formulæ. Intuitively, the following theorems convert the problem of deciding dominance between formulæ into a problem of set membership. If, for a given fault, formula A always produces a smaller *better* subset than formula B (i.e., a fewer number of statements whose risk values are higher than that of the faulty statement compared to formula B), then A dominates B [52]. These theorems formalise the concept of set-based dominance and equivalence:

THEOREM 2.6. *Given any two risk evaluation formulæ R_1 and R_2 from $\mathcal{F}$, if, for any program, faulty statement s_f, and test suite, it holds that $S^{R_1}_B \subseteq S^{R_2}_B \wedge S^{R_2}_A \subseteq S^{R_1}_A$, then $R_1 \rightarrow R_2$.*

THEOREM 2.7. *Let R_1 and R_2 be two risk evaluation formulæ from $\mathcal{F}$. If, for any program, faulty statement s_f, and test suite, it holds that $S^{R_1}_B = S^{R_2}_B \wedge S^{R_1}_F = S^{R_2}_F \wedge S^{R_1}_A = S^{R_2}_A$, then $R_1 \leftrightarrow R_2$.*

Table II. Known and Novel Maximal SBFL formulæ

Equivalence Group	Formula	Expression	Found by
ER_1'	Naish1 [37]	$\begin{cases} -1 & \text{if } e_f < F \\ n_p & \text{if } e_f = F \end{cases}$	Human
	Naish2 [37]	$e_f - \frac{e_p}{e_p+n_p+1}$	Human
	GP13 [55]	$e_f(1 + \frac{1}{2e_p+e_f})$	GP
ER_5	Wong1 [51]	e_f	Human
	Russel & Rao [46]	$\frac{e_f}{e_f+n_f+e_p+n_p}$	Human
	Binary [37]	$\begin{cases} 0 & \text{if } e_f < F \\ 1 & \text{if } e_f = F \end{cases}$	Human
GP02 [55]		$2(e_f + \sqrt{n_p}) + \sqrt{e_p}$	GP
GP03 [55]		$\sqrt{\lvert e_f^2 - \sqrt{e_p}\rvert}$	GP
GP19 [55]		$e_f\sqrt{\lvert e_p - e_f + n_f - n_p\rvert}$	GP

Let us briefly sketch the proof for Theorem 2.6. Consider another formula R_3, such that for any program, s_f and test suite, $S_B^{R_3} = S_B^{R_1}$, $S_F^{R_1} \subseteq S_F^{R_3}$ and $S_A^{R_3} \subseteq S_A^{R_1}$; $S_B^{R_3} \subseteq S_B^{R_2}$, $S_F^{R_2} \subseteq S_F^{R_3}$, and $S_A^{R_3} = S_A^{R_2}$. The assumption about a consistent tie-breaking scheme implies that, within the equal subset of a fault, S_F^R, the faulty statement s_f will always get the same relative ranking. Consequently, we always have $E_1 \leq E_3 \leq E_2$. Immediately after Definition 2.3, this theorem is proved. The proof for Theorem 2.7 follows naturally: if two formulæ produce before and after sets of an equal size, s_f is always ranked at the same place by them. For detailed proofs, please refer to the previous work [52].

The definition of limited maximality, that is, maximality with respect to $\mathbb{S}$, is as follows:

Definition 2.8 (*Limited Maximality*). A risk evaluation formula R_1 from a subset of formulæ, $\mathbb{S} \subset \mathcal{F}$, is said to be a *maximal* formula of $\mathbb{S}$ if for any element $R_2 \in \mathbb{S}$, $R_2 \rightarrow R_1$ implies $R_2 \leftrightarrow R_1$.

Table II shows the two groups of maximal formulæ, ER_1 and ER_5, that have been identified by studying 30 available formulæ. It also lists some formulæ evolved by Genetic Programming (ER_1' is ER_1 extended by a new entry from GP), which are introduced in Section 3. The detailed and complete proofs that formulæ within each group share the same set subdivision can be found in the previous work [52].

3. GENETIC PROGRAMMING FOR SBFL FORMULÆ

3.1. Current Status

After over a decade of manual effort to design SBFL formulæ, Genetic Programming (GP) has been applied to the automated design of SBFL formulæ. Yoo evolved a set of 30 different formulæ by formulating the design of SBFL formulæ as expression searching through GP [55]. Yoo represented SBFL formulæ in GP trees using a basic set of GP nodes, with protected division and square root, listed in Table III. GP was configured with ramping initialisation, rank selection, single point crossover with rate of 1.0, and subtree replacement operator with the rate of 0.8.

Fitness of a candidate GP tree was measured by applying the corresponding formula to spectrum data sets from known faults in SIR testing benchmark suite [13]. The raw fitness of a candidate GP tree is the average normalised ranking of the known seeded faulty statements in the training spectrum data sets: trees were evolved to minimise this, measured from 20 randomly selected faults of 92 studied.

Table III. List of GP Operators used by Yoo [55]

Operator Node	Definition
`gp_add(a, b)`	`a + b`
`gp_sub(a, b)`	`a - b`
`gp_mul(a, b)`	`ab`
`gp_div(a, b)`	1 if $b = 0$, $\frac{a}{b}$ otherwise
`gp_sqrt(a)`	$\sqrt{\lvert a \rvert}$

Yoo empirically evaluated the evolved formulæ using a separate set of the remaining 72 known seeded faults, reserved as testing sets. Across 30 independent evolutions, GP rarely repeated itself and produced a range of different formulæ. Evaluated empirically against human designed formulæ including maximal formulæ such as Naish1 and Wong1 in Table II, some of the evolved formulæ performed equally well, or even better than the known maximal formulæ. While this suggests the necessity of repeated applications of GP to obtain a well performing formula (due to the inherent randomness of GP), the results of the empirical evaluation were encouraging: this was the first time GP produced human competitive results for the design of SBFL formulæ.

The human competitiveness of the evolved formulæ has been subsequently proved theoretically: Xie et al. applied the existing theoretical framework to show that GP evolved a formula (GP13 in Table II) that is equivalent to the known maximal formulæ designed manually [53]. Other evolved formulæ formed their own maximal groups, such as GP02, GP03, an GP19 in Table II.

The current status of the application of GP to SBFL can be summarised as follows:

- GP can successfully evolve SBFL risk evaluation formulæ using the spectral data sets of known faults as training data sets [55].
- GP-evolved formulæ have been theoretically proven to be equivalent to some of the best performing formulæ designed by humans [53].

3.2. How This Article Advances the State of the Art

Some of the GP-evolved formulæ either belonged to a known maximal group by being equivalent to other formulæ in the group, or formed their own maximal groups. Intuitively, maximal formulæ do not dominate each other: if a maximal formula A performs better than another maximal formula B from a different maximal group when localising fault f_1, it is always possible to construct another fault f_2 for which B outperforms A. However, it is possible that certain faults and resulting spectral patterns are more prevalent than others in the real world software, favouring certain maximal groups. To study the extent of these practical ramifications of the previous theoretical findings, we first undertook an empirical study. Our study compares the performances of the known maximal groups, including GP-evolved formulæ. The results show that one GP-evolved formula, GP13 (and equivalent formulæ), performs best against the largest number of faults empirically.

Subsequently, we investigate whether it is possible that any human can outperform GP. In the second, theoretical part of the article, we prove that no single formula can dominate all known maximal formulæ, including the GP-evolved one. That is, the *greatest* formula does not exist.

Therefore, the empirical and theoretical studies in this article collectively demonstrate that GP has evolved an SBFL formula that not only performs the best empirically but also is provably the best possible, providing very compelling evidence for the human competitiveness of GP.

Table IV. Subject Programs

Program	Description	LOC	# of Tests
`flex`	Lexical analyser	9,933	670
`grep`	Text-search utility	7,309	809
`gzip`	Compression utility	3,883	214
`sed`	Stream text editor	5,257	449
`space`	Array Definition Language interpreter	5,902	13,585

4. EMPIRICAL STUDY

The existing theoretical analysis [52, 53] shows that maximal formulæ among the known 50 formulæ form a non-dominating relationship, which means it is possible that for some faults one maximal formula will always outperform another, while for other faults it will be the opposite. However, the theoretical analysis considers all possible faults. We conjecture that faults that are actually embedded in common program structures and detected by test cases may exhibit certain spectral patterns that will favour certain maximals. For example, a common pattern observed in many risk evaluation formulæ is that higher e_f and n_p values are associated with higher suspiciousness. While this conforms to the common notion in software testing, it is still possible that certain faults will exhibit a different trend under a specific combination of subject programs and test suites, resulting in the actual faulty statement to show e_f and n_p values lower than those not faulty (this particular pattern is later analysed as a feature called Faulty Border in the visualization of risk evaluation formulæ; please refer to Definition 5.3 in Section 5.2).

Our interest in such a phenomenon is twofold. First, if such a favoured maximal group exists, practitioners should use formulæ from it (RQ1). Second, we want to see whether GP-evolved maximal formulæ are favoured over other, human designed formulæ (RQ2). To investigate this, the empirical study evaluates the known maximal groups[4] in Table II against each other using faults injected to a widely studied testing benchmarks. The aim of the empirical result is to complement the theoretical analysis with a set of benchmark programs; however, it should be still noted that other sets of subject programs may yield different results.

4.1. Experimental Set-up

4.1.1. Subject Programs. In these experiments, we use five subject programs from Software Infrastructure Repository (SIR) [13]. Table IV describes the functionalities and sizes of these programs: the size is measured in Source Lines of Code, excluding whitespaces, using `SLOCCount` [47].[5] Table IV also presents the size test suites employed. We adopted all test cases provided by SIR, including the "universe" test plan and the additional test cases.

4.1.2. Faults and Measurements. For the empirical study, we generated 200 randomly mutated versions of each program without any duplicates. The mutation has been applied using a C mutation tool we implemented in Perl, which contains the following mutation operators: insertion, deletion, and replacement of unary, binary, and short-cut arithmetic operators, replacement of relational operators, replacement of conditional operators, insertion, deletion, and replacement of logical operators, and replacement of

[4]GP02, GP03, and GP19 have been slightly modified based on the insights we gained while working on the theoretical study; see Section 2.2 Proposition 5.11. As such, they are referred to as GP_2^M, GP_3^M, and GP_{19}^M from now on.

[5]All source code files in each program has been combined into a single `.c` file, which is how these subject programs are provided by SIR. Our mutation operators have been applied to this single source file.

short-cut assignment operators. Each mutated version contains a single faulty statement. By executing the adopted test cases on the mutated subject programs, we filter out the mutants that either fail to compile or crash test cases rather than terminate with failure. The remaining numbers of mutants for the five program are: 96 for `flex`, 67 for `grep`, 71 for `gzip`, 113 for `sed`, and 118 for `space`.

The spectral data consist of the structural coverage achieved by individual test cases and their outcomes (i.e., pass or fail). We use statement coverage to rank program statements using SBFL: the coverage has been collected using the GNU profiler `gcov`. When multiple statements are assigned with the same risk evaluation score, we use their original line number as the tie breaker: the statement with the lower line number becomes higher ranked.[6] The test cases were executed on a cluster of 64-bit Intel Clovertown CPUs running CentOS version 5.0.

The formulæ are evaluated using the Expense metric, which is the percentage of code that needs to be examined before the faulty statement is identified [45]. The lower the Expense metric from a formula for a given fault is, the fewer statements the developer has to check, hence the better the performance of the formula is.

4.2. Experimental Result

4.2.1. Descriptive Statistics. Figure 1 presents the descriptive statistics, including the mean, the lower (Q1) and the upper (Q3) quartiles, as well as 1.5 times the Inter Quartile Range (IQR—denoted by the whiskers of the boxplots), computed over all mutants of each subject program. In general, ER'_1 performs the best among the five maximal formulæ and maximal groups. For all five subject programs, ER'_1 tends to produce the lowest Expense, with noticeable difference in some cases. For example, with `grep`, the minimum Expense of ER_5 is about 84 times larger than that of ER'_1. Similarly, with `space`, at the point Q3, the Expense values of GP^M_{19} and ER'_1 are 7.94% and 0.53%, respectively, the former being about 14 times larger than the latter. The trend is repeated in the following case, in which the Expense of the latter is about 7 to 74 times of that of the former (i.e., ER'_1):

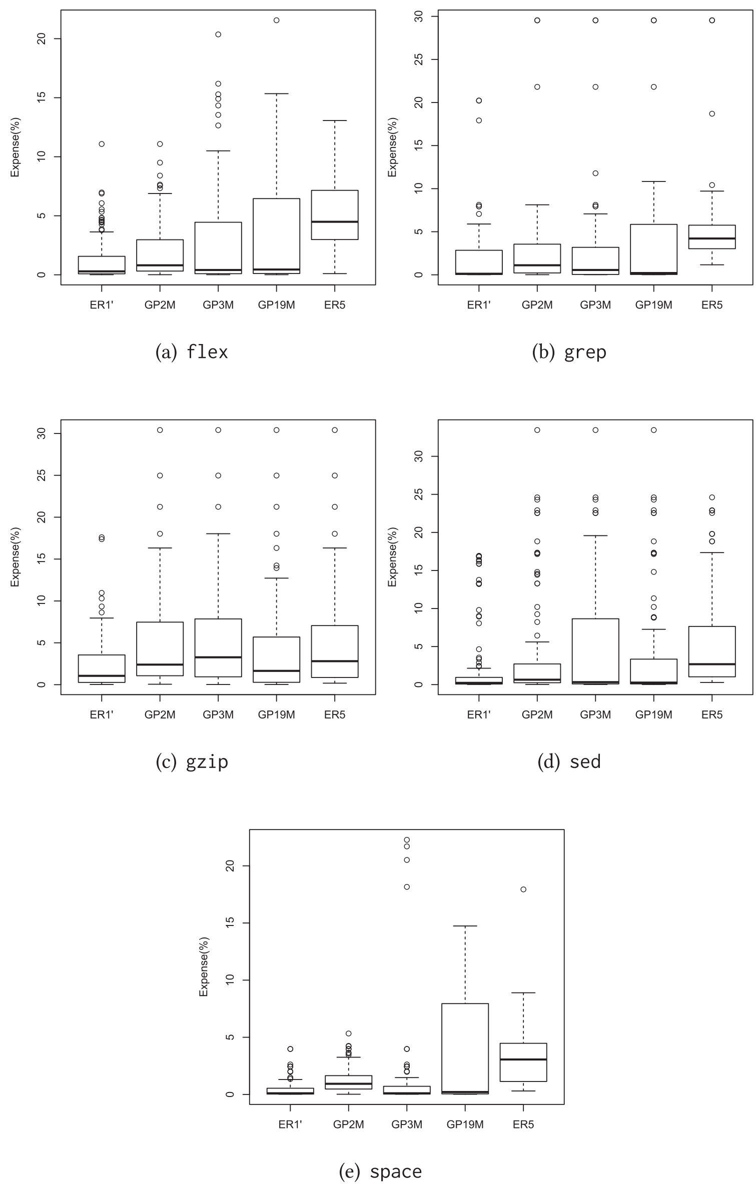
- ER'_1 *vs.* ER_5: at the minimum point in all subject programs, at points Q1 and median in all programs except `gzip`, and at point Q3 in all programs except `grep` and `gzip`.
- ER'_1 *vs.* GP^M_2: at points Q1 and median in `grep` and `space`.
- ER'_1 *vs.* GP^M_3: at point Q3 in program `sed`, and at the maximum point in `space`.
- ER'_1 *vs.* GP^M_{19}: at point Q3 in `space`.

In remaining cases, although ER'_1 does not show significant advantages over the other formulæ, it still produces the lowest Expenses among all the five maximal formulæ and groups when comparing the minimum, Q1, median, Q3, and the maximum. The mean Expense values of GP^M_2, GP^M_3, GP^M_{19}, and ER_5 are from 1.2 to 7.5 times larger than that of ER'_1.

On the other hand, ER_5 shows the worst performance among these five maximal formulæ and groups. Its Expense values are mostly the highest in all programs, except for the maximum in `flex`, `sed`, and `space`, for which GP^M_2, GP^M_3, or GP^M_{19} perform the worst.

Formulæ GP^M_2, GP^M_3, and GP^M_{19} tend to produce very similar Expense values that are higher than those of ER'_1 but lower than those of ER_5 in most cases. However, the results of the comparisons of formulæ other than ER'_1 are not always consistent.

[6]Note that any consistent tie-breaker, that is, one that always breaks ties between two specific statements in the same way, will do here. We choose the line number as a simple tie breaker that satisfies the requirement.



(a) flex (b) grep

(c) gzip (d) sed

(e) space

Fig. 1. Boxplots of Expense metric from the subject programs.

In addition to the mean values, we are also interested in the dispersal of the observed Expense values, because it represents the stability of a formula. The smaller the dispersal is, the narrower the range of the Expense is, which means that more reliable and consistent performance can be expected. We have performed Shapiro-Wilk normality test to check whether the observed Expense values are normally distributed: the results, presented in Table VI, suggest that we can reject the null hypothesis that the sample comes from a population that has a normal distribution. Consequently, we use IQR as the measure of dispersal. Figure 1 shows that ER_1' has the smallest dispersal, which means ER_1' not only performs the best, but also performs the most

Table V. Wilcoxon-Signed-Rank Test p-values after Bonferroni Correction and A_{12} Statistics

A	B	2-tailed	1-tailed, L	1-tailed, U	A_{12}	2-tailed	1-tailed, L	1-tailed, U	A_{12}	2-tailed	1-tailed, L	1-tailed, U	A_{12}
		flex				grep				gzip			
ER'_1	GP^M_2	3.59E-07	1.00E+00	1.79E-07	0.36	4.84E-05	1.00E+00	2.42E-05	0.36	5.16E-07	1.00E+00	2.58E-07	0.34
ER'_1	GP^M_3	2.00E-02	1.00E+00	1.00E-02	0.45	3.66E-02	1.00E+00	1.83E-02	0.46	1.50E-06	1.00E+00	7.51E-07	0.36
ER'_1	GP^M_{19}	7.38E-05	1.00E+00	3.69E-05	0.41	1.91E-01	1.00E+00	9.54E-02	0.44	7.63E-02	1.00E+00	3.82E-02	0.46
ER'_1	ER_5	3.00E-11	1.00E+00	1.50E-11	0.13	1.99E-12	1.00E+00	9.96E-13	0.18	3.79E-04	1.00E+00	1.90E-04	0.35
GP^M_2	GP^M_3	1.00E+00	1.00E+00	1.00E+00	0.57	1.00E+00	1.00E+00	1.00E+00	0.59	1.00E+00	1.00E+00	1.00E+00	0.50
GP^M_2	GP^M_{19}	1.00E+00	1.00E+00	1.00E+00	0.53	1.00E+00	1.00E+00	1.00E+00	0.55	5.04E-02	2.52E-02	1.00E+00	0.60
GP^M_2	ER_5	3.56E-08	1.00E+00	1.78E-08	0.20	4.08E-08	1.00E+00	2.04E-08	0.23	1.00E+00	1.00E+00	1.00E+00	0.51
GP^M_3	GP^M_{19}	1.00E+00	1.00E+00	1.00E+00	0.47	1.00E+00	1.00E+00	1.00E+00	0.47	1.53E-02	7.64E-03	1.00E+00	0.59
GP^M_3	ER_5	7.67E-05	1.00E+00	3.83E-05	0.24	1.00E-07	1.00E+00	5.00E-08	0.20	1.00E+00	1.00E+00	1.00E+00	0.50
GP^M_{19}	ER_5	7.87E-03	1.00E+00	3.94E-03	0.29	7.61E-04	1.00E+00	3.80E-04	0.31	1.00E+00	1.00E+00	8.12E-01	0.40
		sed				space							
ER'_1	GP^M_2	1.33E-05	1.00E+00	6.64E-06	0.34	1.99E-13	1.00E+00	9.94E-14	0.22				
ER'_1	GP^M_3	3.85E-04	1.00E+00	1.92E-04	0.42	1.00E+00	1.00E+00	1.00E+00	0.49				
ER'_1	GP^M_{19}	1.96E-03	1.00E+00	9.82E-04	0.45	1.09E-09	1.00E+00	5.46E-10	0.38				
ER'_1	ER_5	3.54E-14	1.00E+00	1.77E-14	0.18	4.82E-16	1.00E+00	2.41E-16	0.07				
GP^M_2	GP^M_3	1.00E+00	1.00E+00	1.00E+00	0.56	3.70E-07	1.85E-07	1.00E+00	0.76				
GP^M_2	GP^M_{19}	1.00E+00	1.00E+00	1.00E+00	0.59	1.00E+00	1.00E+00	1.00E+00	0.57				
GP^M_2	ER_5	7.83E-08	1.00E+00	3.91E-08	0.29	1.42E-11	1.00E+00	7.10E-12	0.22				
GP^M_3	GP^M_{19}	5.49E-01	2.75E-01	1.00E+00	0.54	2.15E-03	1.00E+00	1.08E-03	0.39				
GP^M_3	ER_5	9.80E-01	1.00E+00	4.89E-01	0.32	7.61E-12	1.00E+00	3.80E-12	0.10				
GP^M_{19}	ER_5	5.24E-07	1.00E+00	2.62E-07	0.26	1.00E+00	1.00E+00	1.00E+00	0.34				

consistently. On the other hand, GP^M_3 and GP^M_{19} have the most unstable performance among the maximal formulæ and groups. While GP^M_2 and ER_5 can deliver relatively stable performance for `flex` and `space`, they behave more similarly to GP^M_3 and GP^M_{19} for the remaining three programs.

4.2.2. Wilcoxon Signed Rank Test and Effect Sizes. Table V presents the results of the paired Wilcoxon Signed Rank Test. We use the paired version of Wilcoxon Signed Rank test to compare the Expense values of the five maximal formulæ and groups. The paired Wilcoxon Signed Rank test is a non-parametric statistical hypothesis test that makes use of the sign and the magnitude of the rank of the differences between pairs of measurements, $E(A)$ and $E(B)$, that do not follow a normal distribution [10]. At the given significant level σ, there are both two-tailed p-value and one-tailed p-value, which can be used to obtain a conclusion.

Table VII contains the interpretations of the hypotheses in the context of the current experiment. In the current context, for a given pair of formulæ A and B, the list of measurements for $E(A)$ would be the list of the Expense values for all mutants produced by A, while the list of measurements for $E(B)$ would be the list of the Expense values for all mutants produced by B. For the two-tailed p-value, if $p \geq \sigma$, the null hypothesis H_0 that $E(A)$ and $E(B)$ are not significantly different is accepted; otherwise, the alternative hypothesis H_1 that $E(A)$ and $E(B)$ are significantly different is accepted. For one-tailed p-value, there are two cases, the lower case and the upper case. In the lower case, if $p \geq \sigma$, H_0 that $E(A)$ does not significantly tend to be greater than the $E(B)$ is accepted; otherwise, H_1 that $E(A)$ significantly tends to be greater than the $E(B)$ is accepted. And in the upper case, if $p \geq \sigma$, H_0 that $E(A)$ does not significantly tend to be less than the $E(B)$ is accepted; otherwise, H_1, that $E(A)$ significantly tends to be less than the $E(B)$, is accepted.

In our experiments, for each subject program, we conduct Wilcoxon Signed Rank Test for the following pairs: ER'_1 *vs.* GP^M_2, ER'_1 *vs.* GP^M_3, ER'_1 *vs.* GP^M_{19}, ER'_1 *vs.* ER_5, GP^M_2 *vs.* GP^M_3, GP^M_2 *vs.* GP^M_{19}, GP^M_2 *vs.* ER_5, GP^M_3 *vs.* GP^M_{19}, GP^M_3 *vs.* ER_5, and GP^M_{19} *vs.* ER_5. In total, this results in 150 (10 pairs of formulæ $\times$ 3 types of hypotheses $\times$ 5 programs) Wilcoxon Signed Rank Sum tests. Given the large number of hypotheses testing, we have applied the standard Bonferroni adjustment [5] to address the problem of the

Table VI. The p–values from Shapiro-Wilk Normality Test on Observed Expense Values

Subject	ER'_1	$GP2^M$	$GP3^M$	$GP19^M$	ER'_5
flex	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**	**0.0406**
grep	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**
gzip	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**
sed	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**
space	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**	**< 1e-4**

Table VII. Interpretation of the Hypotheses in the Context of SBFL

Hypotheses:	H_0	H_1
Acceptance Condition:	p-value ≥ 0.05	p-value < 0.05
2-tailed:	$E(A) \simeq E(B)$: A and B DO have similar performance	$E(A) \not\simeq E(B)$: A and B DO NO T have similar performance
1-tailed (lower):	$E(A) \leq E(B)$: A DOES NOT tend to be worse than B	$E(A) > E(B)$: A DOES tend to be worse than B
1-tailed (upper):	$E(A) \geq E(B)$: A DOES NOT tend to be better than B	$E(A) < E(B)$: A DOES tend to be better than B

higher probability of Type I errors in multiple comparisons. Both the two-tailed and the one-tailed p-values are recorded. We set the α level (after Bonferroni correction) to 0.05.

Table V also contains Vargha-Delaney's A_{12} statistics [49] that measures the effect sizes. If, when calculated between formula A and B, the value of A_{12} is lower than 0.5, then it means that A outperforms B (i.e., A tends to produce lower Expense than B); greater than 0.5, B outperforms A (i.e., B tends to produce lower Expense than A). The farther the value is from 0.5, the greater the effect size is. It presents a similar conclusion to the ones observed in Section 4.2.1. In general, ER'_1 shows the best performance; while the effect sizes vary, it consistently outperforms all the other. Similarly, ER_5 is consistently outperformed by others, with the exception of the case of `gzip` for which ER_5 performs more equally to the GP evolved formulæ. For all other subjects, GP evolved formulæ tend to form the middle group.

These partial orders can be summarised into the following order of maximal formulæ and groups, based on their performance, for each subject program, as follows. $A \geq B$ means "A is better than or similar to B" indicated by the statistical test, while those grouped by parentheses form weak orders with small effect sizes:

- `flex`: $ER'_1 \geq (GP_3^M \geq GP_{19}^M \geq GP_2^M) \geq ER_5$,
- `grep`: $ER'_1 \geq (GP_3^M \geq GP_{19}^M \geq GP_2^M) \geq ER_5$,
- `gzip`: $ER'_1 \geq (GP_{19}^M \geq GP_3^M) \geq (ER_5 \geq GP_2^M)$
- `sed`: $ER'_1 \geq (GP_{19}^M \geq GP_3^M \geq GP_2^M) \geq ER_5$,
- `space`: $(ER'_1 \geq GP_3^M) \geq (GP_{19}^M \geq GP_2^M) \geq ER_5$.

These partial orders confirm the observations in Section 4.2.1 that: (i) in general ER'_1 convincingly outperforms other formulæ, and (ii) ER_5 performs the worst in most cases. Recall that ER'_1 was found by GP, so this finding indicates that GP found the most attractive maximal formula according to our empirical analysis of the practical aspects of the risk formulæ studied.

Other GP-evolved formulæ, GP_3^M and GP_{19}^M perform roughly the same overall, while GP_2^M being the worst among the GP formulæ. GP_3^M performs noticeably better than GP_2^M and GP_{19}^M for `space`, but the trend is not repeated in other programs. However, it is not possible to generalise the comparison between these three formulæ based on only five subject programs.

4.3. Discussion

4.3.1. Insights. The results of the experimental study provide guidance on which maximal formula or group to apply when there is no *a priori* knowledge about the fault. While these formulæ are all maximal as described in Section 2.2, their effectiveness against actual faults can vary significantly, showing the value of empirical study. To summarise the insights from the results of the study:

- ER_1' tends to perform better than the other four maximal formulæ and groups.
- ER_5 tends to perform worse than the other four maximal formulæ and groups.
- GP_2^M, GP_3^M, and GP_{19}^M perform better than ER_5 but worse than ER_1'. Comparisons between these three formulæ show mixed results.

These partial orders collectively answer RQ1. From Table II, we know that the risk values of statements with $e_f = F$ monotonically decrease with e_p for formulæ in ER_1'. However, this is not the case with GP_2^M, GP_3^M, or GP_{19}^M. The fact that ER_1' produces generally stable and good performance, while GP_2^M, GP_3^M, and GP_{19}^M give results with larger variances, confirms the existing intuition on designing SBFL formulæ: in general, higher e_f and lower e_p values are believed to be correlated with higher risk evaluation values and lower Expense. Those formulæ not following this intuition may deliver very good performance, but only for the class of faults whose corresponding e_p values also happen to be high. This observation essentially recaptures the claim of single-fault optimality posited by Naish et al. [38]; the same has also been observed in the trend among formulæ evolved by genetic programming [55]. As for the better performance of GP (GP_2^M, GP_3^M, and GP_{19}^M) over ER_5, it is most likely because ER_5 does not further distinguish statements whose e_f values are equal to F. Consequently, the performance of ER_5 shall largely depend on the adoption of tie-breaking scheme.

To answer RQ2, overall, Genetic Programming has successfully evolved GP_{13} that is equivalent to manual designed formulæ in ER_1, and GP_2^M, GP_3^M, and GP_{19}^M that outperform another manually designed formulæ in ER_5, which shows its capability to evolve competent SBFL formula.

4.3.2. Threats to Validity. There are several sources of threats to validity of the empirical study. Since the empirical study uses program mutation to seed faults, the choice of mutation operators may affect the behaviour of faults. In addition, real not seeded faults may affect the performance of different maximal formulas differently. All subject programs are small to medium `C` programs, and analysis on programs of different sizes, written in different language, may also produce different results. Finally, the method used to generate the test suites provided by SIR may affect the fault detection capability of the resulting test suites, eventually affecting the composition of spectrum datasets. While all these factors limit the degree to which the findings can be generalised, the complementary theoretical analysis as well as existing empirical analysis of individual risk evaluation formulæ may be consulted to mitigate the threats. Comparisons of maximal risk evaluation formula groups using real-world faults remain as future work.

5. MAXIMAL AND GREATEST FORMULÆ

Now, we turn to the question of whether some future work (by human or machine) could potentially outperform the results already obtained using GP in the context of SBFL under the single fault scenario. The existing definition of a maximal formula in Definition 2.8 only concerned a subset of formulæ, $\mathbb{S}$, out of all possible formulæ, $\mathcal{F}$. The subset $\mathbb{S}$ contained 50 formulæ, 30 manually designed ones and 20 GP-evolved ones. The five identified maximal groups are only with respect to these 50 formulæ. Now, let us generalise our analysis by replacing $\mathbb{S}$ with $\mathcal{F}$. This will, in turn, lead to the investigation of the "greatest" formula. We first construct a 3D space that can visualize risk evaluation formulæ with Lemma 5.1, and narrow down the location that corresponds to the faulty program element using Lemmas 5.2, 5.4, and 5.5.

5.1. Preliminaries

5.1.1. Spectral Coordinate. Let us first present some definitions and lemmas. Given a test suite TS, let T denote its size, F denote the number of *failed* test cases, and P denote the number of passed test cases. From the definitions and the earlier assumptions, it follows that $1 \leq F < T$, $1 \leq P < T$, and $P + F = T$, as well as the following lemmas:

LEMMA 5.1. *For any $\sigma(s_i) =< e^i_f, e^i_p, n^i_f, n^i_p >$, it holds that $e^i_f + e^i_p > 0 \wedge e^i_f + n^i_f = F \wedge e^i_p + n^i_p = P \wedge e^i_f \leq F \wedge e^i_p \leq P$.*

LEMMA 5.2. *For any faulty statement s_f with $\sigma(s_f) =< e^f_f, e^f_p, n^f_f, n^f_p >$, if s_f is the only faulty statement in the program, it follows that $e^f_f = F \wedge n^f_f = 0$.*

Intuitively, Lemmas 5.1 and 5.2 allow us to reason about risk evaluation formulæ spatially in three dimensions. Definition 2.1 involves five dimensions: four members of program spectrum and the risk score. Following the visualisation method used by Lee [33], we now reduce the space of SBFL to three dimensions. For a given pair of program and test suite, the values of F and P are constants. Thus, for each statement s_i, it follows that $\sigma(s_i) =< e^i_f, P - n^i_p, F - e^i_f, n^i_p >$ after Lemma 5.1, which can be denoted as $\bar{\sigma}(s_i) =< e^i_f, e^i_p >$. That is, program spectrum contains two independent parameters in a specific context (i.e., a pair of a program and a test suite), and not four.

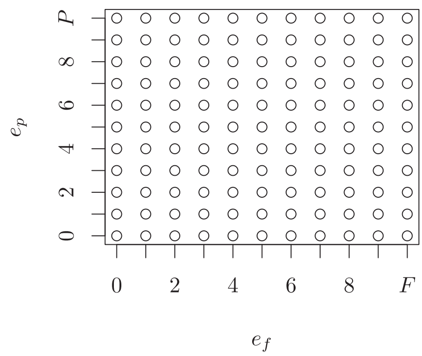
Consequently, it is possible to formulate $\overline{\mathcal{F}} = \{\overline{R} | \overline{R} : I_f \times I_p \rightarrow Real\}$, where I_f denotes the set of integers within $[0, F]$ and I_p denotes the set of integers within $[0, P]$, such that $\overline{R}(e^i_f, n^i_p) = R(e^i_f, e^i_p, n^i_f, n^i_p))$. In the subsequent discussion, when two formulæ from $\mathcal{F}$ are compared, it is assumed that they are being applied to the same program and test suite. Thus, in the context of such comparisons, symbols R and $\overline{R}$ can and will be used interchangeably, as are symbols $\mathcal{F}$ and $\overline{\mathcal{F}}$.

Given any values of P and F, the input domain of any formula $\overline{R}$ is shown as the grid in Figure 2(a), where both e_f and e_p are non-negative integers and $0 \leq e^i_f \leq F$ and $0 \leq e^i_p \leq P$. Given a pair of test suite and program, each point (e_f, e_p) on this grid is associated with a group of statements that have the corresponding e_f and e_p values. Note that the number of statements that associated with each point (e_f, e_p) is independent of the formula but solely decided by the pair of program and test suite.

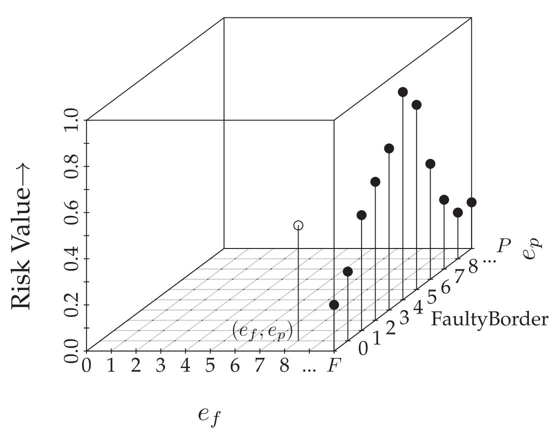
A formula $\overline{R}$ maps each point $\bar{\sigma} = (e_f, e_p)$ to a real number that is the risk value of all statements associated with this point, as shown in Figure 2(b). Any assignment of risk values is independent of the number of statements associated with each point (e_f, e_p) but solely decided by the definition of $\overline{R}$.

5.1.2. Analysis of SBFL Space. Now let us focus on the part of the SBFL space that actually contains the coordinate of the faulty statement. This, in turn, will allow us to reason about both maximal formulæ and the greatest formula more precisely. Lemma 5.2 allows us to limit the region of the input domain $\overline{A}$ in which the faulty statement can be.

Definition 5.3 (*Faulty Border*). Let us call the sequential points $<(F, 0), (F, 1), \ldots, (F, e_p), \ldots, (F, P)>$ $(0 \leq e_p \leq P)$ the *Faulty Border*, which is denoted as E. Figure 2(b) illustrates a potential E.



(a) Spectral Coordinate $\bar{\sigma}$ for SBFL formulæ: because $e_f^i + e_p^i > 0$ and $e_f^i + n_f^i = F$, the program spectrum of any statement can be simply reduced to $< e_f^i, e_p^i >$, given fixed values of P and F. The program spectrum can be interpreted as a 2D plane with discrete coordinates depicted above.



(b) Mapping from $\bar{\sigma}$ to risk values by formula $\overline{R}$[1]: SBFL formulæ assign risk scores to program statements, which can be mapped to the spectral coordinates in Figure 2(a). Consequently, we obtain a 3D space, in which formulæ map spectral coordinates to risk scores. The points whose e_f values are equal to F form the faulty border: the faulty statement is guaranteed to be mapped on the border.

Fig. 2. Visualising the SBFL Space.

Immediately from the above definition, for any given formula R, it follows that the risk values of all points on E are solely decided by their values of e_p. In addition, immediately after Lemma 5.2, the faulty statement s_f is associated with the point (F, e_p^f) of E, where $0 \leq e_p^f \leq P$, as stated in the following lemma.

LEMMA 5.4 (LOCATION OF FAULTY STATEMENT s_f). *The faulty statement s_f must be associated with a point (F, e_p^f) on E. And e_p^f can be any value between* $[0, P]$.

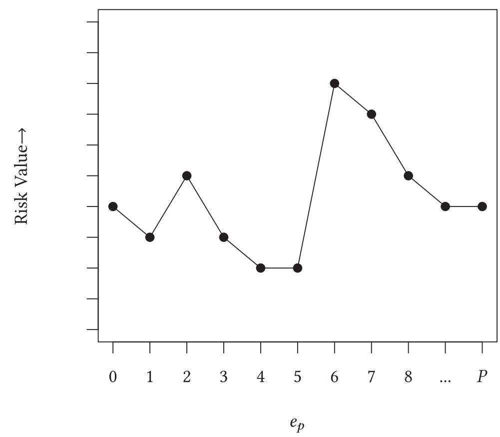


Fig. 3. Example of a faulty border of $\overline{R}$.

Lemma 5.4 reflects multiple confounding factors in software testing. One of the confounding factors is the "Coincidental Correctness Test" (CCT) [34]. Ideally, the faulty statement s_f will produce $e_p^f = 0$, as executing s_f should result in a failure. CCTs are the tests that execute s_f but still pass. The number of CCT is equal to e_p^f, that is, the value of e_p for s_f. There can be an arbitrary number of CCTs in a given test suite, and so is the value of e_p^f. Another factor is that the spectrum data abstracts the data input/output; both a passing and a failing test execution can exhibit the same spectrum as a result.

As a reminder, points (F, e_p^i) other than the one associated with s_f on E are associated with correct statements, where $n_f^i = F \ \wedge \ 0 \leq e_p^i \leq P \ \wedge \ e_p^i \neq e_p^f$. Depending on the adopted formula, the risk values of such points can be either greater than, equal to, or smaller than that of point (F, e_p^f), that is, the point associated with s_f.

Lemma 5.5. *For a given program and a test suite, the point of E, with which s_f is associated, may also be associated with other correct statements s_i having $(F, e_p^i) = (F, e_p^f)$. These statements share the same risk values as that of s_f, regardless of the selection of the formula.*

Note that a correct statement that is executed if and only if s_f is executed will share the same risk evaluation value as s_f. Lemma 5.5 reflects another common phenomenon in software testing. That is, correct statements s_i may still have $e_f^i = F$, and their e_p^i could be either greater than, equal to or smaller than e_p^f of the faulty statement s_f and so are their risk values. An example of the faulty border can be found in Figure 3.

5.2. Maximality in $\mathcal{F}$

First, let us present the definition of the maximality with respect to $\mathcal{F}$, that is, the set of all possible formulæ. In contrast, existing work defined them only with respect to the known formulæ.

Definition 5.6 (*Maximality*). A risk evaluation formula R is said to be a maximal formula in $\mathcal{F}$ if, for any formula $R' \in \mathcal{F}$ such that $R' \neq R \wedge R' \rightarrow R$, it also holds that $R' \leftrightarrow R$.

Definition 5.6 means that if R is a maximal formula in $\mathcal{F}$, there will be no other formula R' that can be strictly better than R. An important implication of Definition 5.6 is that maximal formulæ from different maximal groups, such as one from $ER1'$ and another from $ER5$ do not form any definite relation. A formula from $ER1'$ will, in some cases, rank certain faults higher than $ER5$; however, there exist other faults that are ranked higher by $ER5$ than by $ER1'$. The remainder of Section 5.2 proves that formula groups that have been proved to have limited maximality [52] are also generally maximal as in Definition 5.6.

Intuitively, Definition 5.6 states that, if a maximal formula R is dominated by R' (i.e., $R' \rightarrow R$), then it is only because R and R' are equivalent to each other (i.e., $R \leftrightarrow R'$). However, since a formula from one maximal group, for example, ER'_1, does not dominate another formula from another maximal group, for example, ER_5, formulæ from different maximal groups do not have to be equivalent to each other.

Definition 5.7 (*Ranking*). Given a formula R, we use $o_p^{i,j} =< n_p^i, n_p^j, op >$ to denote the relation between the risk scores of two distinct points (F, n_p^i) and (F, n_p^j) on E. Given that $n_p^i < n_p^j$, op can be either "$>$" (i.e., $R(F, n_p^i) > R(F, n_p^j)$), "$<$" (i.e., $R(F, n_p^i) < R(F, n_p^j)$), or "$=$" (i.e., $R(F, n_p^i) = R(F, n_p^j)$).

Let P_R denote the set of $o_p^{i,j}$ based on Definition 5.7. P_R effectively captures the ranking between the statements that belong to E, by collecting the relations between risk scores of each pair of distinct points on the faulty border E. Let U_R denote the set of points outside E that have risk scores higher than or equal to those of some points (F, e_p^i) on E, for formula R. More formally,

$$U_R = \{\bar{\sigma} \in I_f \times I_p - E | \exists \bar{\sigma}' \in E \text{ such that } R(\bar{\sigma}) \geq R(\bar{\sigma}')\}.$$

With all the above preliminary, let us now turn to the analysis of the maximality for all formulæ in $\mathcal{F}$. Lemma 5.8 states that a maximal formula R cannot have non-empty U_R. Intuitively, if U_R is not empty, the point in U_R can be used to construct a program with which a non-faulty statement (that corresponds to the point in U_R) ranks higher than the actual faulty statement. Furthermore, we can subsequently create another formula R' that is guaranteed to dominate R: R' simply needs to suppress the score of points in U_R to dominate R.[7]

LEMMA 5.8. *For any formula $R \in \mathcal{F}$, if $U_R \neq \emptyset$, then R is not a maximal element of $\mathcal{F}$.*

PROOF. The proof shows that, if $U_R \neq \emptyset$, then there exists $R' \in \mathcal{F}$ such that $R' \rightarrow R$ but $R \nrightarrow R'$. First, let us construct $R' \in \mathcal{F}$ such that $R' \rightarrow R$. Assume that U_R is non-empty. Let R' be defined as follow:

$$R' = \begin{cases} R, & \text{if } e_f = F, \\ R - (C_1 - C_2 + 1), & \text{otherwise,} \end{cases}$$

where C_1 is the highest risk value of R for all points outside E, while C_2 is the lowest risk value of R for all points on E. By the definition of R', any point outside E has risk value lower than those of the points in E, which means all statements associated with points outside E have risk values lower than that of s_f.

Let $U_{R'}$ denote the sets of points outside E, which have risk values higher than or equal to those of some points (F, e_p^i) on E, for formula R'. By definition, R' assigns identical risk values to points on E as R, while ensuring that $U_{R'} = \emptyset$.

[7]This can be interpreted as a generalisation of previous work about optimality under the single fault scenario [1]. Please refer to the end of Section 5.3 for a detailed discussion.

Case: statements associated with E. These statements will be assigned to the same set division by both R and R', for any pair of program and test suite.

Case: statements associated with points outside E. For formula R', since these points (including those in U_R) always have risk values lower than that of s_f on E, the corresponding statements belong to $S_A^{R'}$. However, for formula R, since $U_R \neq \emptyset$, some statements corresponding to points outside E belong to either S_B^R, S_F^R, and S_A^R.

Summarizing the above two cases, we have $S_B^{R'} \subseteq S_B^R$ and $S_A^R \subseteq S_A^{R'}$. Following Theorem 2.6, $R' \rightarrow R$.

Let us now turn to show that $R \nrightarrow R'$, by illustrating that it is possible for R' to produce a smaller Expense value than R. Since $U_R \neq \emptyset$, there exists L, a set of points on E whose risk values evaluated by R are not higher than any point in U_R. To show that R' can produce a smaller Expense value than R, it is sufficient to show that $\bar{\sigma}(s_f) \in L$ while $U_R \neq \emptyset$. However, both L and U_R are specific to the choice of R. In order not to lose generality, therefore, let us show that it is possible to construct a program and a test suite such that $\bar{\sigma}(s_f)$ can be placed anywhere on E, and another statement $\bar{\sigma}(s_i)$ can be placed anywhere in $I_f \times I_p - E$, independently from each other.[8] Figure 6 illustrates such a program: the feasibility of the construction of the test suite is described in Example 1 in Appendix.

With such a program and a test suite, any statement associated with points outside U_R always have the same relative ranking to s_f in R and R'. For all statements associated with U_R, formula R' will rank them below s_f. However, with R:

- Statements that are associated with U_R and have risk values higher than that of s_f, are always ranked before s_f by R.
- Statements that are associated with U_R and have risk values equal to that of s_f, will be tied together with s_f by R. However, it is possible to have a consistent tie-breaking scheme that ranks parts or even all of these statements before s_f.

It is always possible to have statements associated with U_R ranked before s_f. Consequently, the Expense of R' is smaller than that of R. Therefore, $R \rightarrow R'$ does hold.

In conclusion, if R assigns point (e_f^j, e_p^j) outside E with risk value higher than, or equal to, that of at least one point (F, n_p^i) on E, there always exists another formula R' for which $R' \rightarrow R$ holds but $R \rightarrow R'$ does not hold. Therefore, following Definition 5.6, R cannot be a maximal formula. $\square$

Now let us show that equivalence between two formulæ depends *only* on the shape of the faulty border E, as long as coordinates outside E are all assigned lower scores than those on it (i.e., U_R is empty). Given two distinct risk evaluation formulæ, R_1 and R_2, let P_{R_1} and P_{R_2} denote the set of $o_p^{i,j}$ for all pairs of distinct points (F, e_p^i) and (F, e_p^j) on E (where $e_p^i < e_p^j$), for R_1 and R_2, respectively. Let U_{R_1} and U_{R_2} denote the sets of points outside E that have risk values higher than, or equal to, those of some points (F, e_p^i) on E, for formula R_1 and R_2, respectively.

LEMMA 5.9. *If $U_{R_1} = U_{R_2} = \emptyset$ and $P_{R_1} = P_{R_2}$, then it follows that $R_1 \leftrightarrow R_2$.*

PROOF. Consider the following two cases.

Case: statements associated with E. Since $P_{R_1} = P_{R_2}$, then for each pair of these statements, the relation between their risk values is always the same in R_1 and R_2. As a consequence, these statements have the same relative order with respect to s_f (which

[8]Given a specific R such that $U_R \neq \emptyset$, this allows us to place $\bar{\sigma}(s_f) \in L$ and $\bar{\sigma}(s_i) \in U_R$.

is associated with one point on E) between R_1 and R_2, and hence belong to the same set-division for R_1 and R_2 with any pair of program and test suite.

Case: statements associated with points outside E. Since both U_{R_1} and U_{R_2} are empty, these statements always have risk values lower than that of the faulty statement s_f (which is associated with one point on E), therefore these statements belong to both $S_A^{R_1}$ and $S_A^{R_2}$.

In summary, we have $S_B^{R_1} = S_B^{R_2}$, $S_F^{R_1} = S_F^{R_2}$ and $S_A^{R_1} = S_A^{R_2}$. Following Theorem 2.7, $R_1 \leftrightarrow R_2$. □

It also follows that, if two formulæ with empty U_R have differently shaped faulty borders, they form a non-dominating pair.

LEMMA 5.10. *If $U_{R_1} = U_{R_2} = \emptyset$ but $P_{R_1} \neq P_{R_2}$, then we have $R_1 \nrightarrow R_2$ and $R_2 \nrightarrow R_1$.*

PROOF. Since $P_{R_1} \neq P_{R_2}$, there must exist at least one pair of points on E, $((F, e_p^i), (F, e_p^j))$ (where $e_p^i < e_p^j$), such that $< e_p^i, e_p^j, op_1 > \in P_{R_1} \wedge < e_p^i, e_p^j, op_2 > \in P_{R_2} \wedge op_1 \neq op_2$. It is sufficient to consider the following two cases, because other cases can be transformed to these two cases by swapping R_1 and R_2:

Case: $R_1(F, e_p^i) < R_1(F, e_p^j)$ and $R_2(F, e_p^i) > R_2(F, e_p^j)$. With the program[9] shown in Figure 6, it is possible to construct a test suite, such that e_f^4, e_f^5, e_f^9, and e_f^{10} are smaller than F. (As a reminder, it always holds that $e_f^2 = e_f^7 = 0$.) While for s_1, s_3, s_6, s_8 (s_f), and s_{11}, whose e_f values are all equal to F, we have $e_p^f = e_p^i < e_p^1 = e_p^3 = e_p^6 = e_p^{11} = e_p^j$[10]. Then, for R_1, we have s_1, s_3, s_6, and s_{11} ranked before s_f and other statements ranked after s_f. However, for R_2, we have s_f ranked at the top of the whole list. Therefore, the Expense of R_2 is lower than that of R_1.

On the other hand, it is also possible to construct another test suite, such that e_f^4 and e_f^5 are both smaller than F, but e_f^9 is equal to F. (Correspondingly, $e_f^{10} = 0$.) For s_1, s_3, s_6, s_8 (s_f), s_9, and s_{11}, whose e_f values are all equal to F, we have $e_p^9 = e_p^i < e_p^1 = e_p^3 = e_p^6 = e_p^f = e_p^{11} = e_p^j$[11]. Then, for R_1, s_1, s_3 s_6, s_f, and s_{11} are tied together at the top of the whole list, before s_9. However, for R_2, s_9 is ranked at the top, immediately followed by s_1, s_3 s_6, s_f, and s_{11} that are tied together. Therefore, with a consistent tie-breaking scheme, the Expense of R_1 is lower than that of R_2.

In summary, for the case that $R_1(F, e_p^i) < R_1(F, e_p^j)$, while $R_2(F, e_p^i) > R_2(F, e_p^j)$, it is always possible to find examples to demonstrate $R_1 \nrightarrow R_2$ and $R_2 \nrightarrow R_1$.

Case: $R_1(F, e_p^i) < R_1(F, e_p^j)$ and $R_2(F, e_p^i) = R_2(F, e_p^j)$. With the program shown in Figure 6, it is possible to construct a test suite, such that $e_f^4 = F$ (correspondingly, $e_f^5 = 0$), while e_f^9 and e_f^{10} are smaller than F. Then, for s_1, s_3, s_4, s_6, s_8 (s_f), and s_{11}, whose e_f values are all equal to F, it follows that $e_p^4 = e_p^i < e_p^1 = e_p^3 = e_p^6 = e_p^f = e_p^{11} = e_p^j$[12]. Then, for R_1, s_1, s_3, s_6, s_f, and s_{11} are tied together at the top of the ranking, before s_4. However, for R_2, s_1, s_3, s_4, s_6, s_f, and s_{11} are tied together at the top of the entire ranking. Since the number of tied statements are different, the Expense now depends on the tie-breaking scheme, without any guarantee of clear dominance of one formula. For example, if the original order of the statements is used as the tie-breaker, R_1 yields

[9]This is a purposefully constructed program to show that it is possible to generate spectrum data required by the proof. It has been previously used by Xie et al. [52].
[10]For the feasibility of this test suite, please refer to **Test Suite A** in Example 5 of the Appendix.
[11]For the feasibility of this test suite, please refer to **Test Suite B** in Example 5 of the Appendix.
[12]For the feasibility of this test suite, please refer to **Test Suite C** in Example 5 of the Appendix.

```
        void foo(double x, double y,
                 double z) {
s1 :      if(z <= 0){
s2 :        // s2
          } else {
s3 :        if (z <= 12) {
s4 :          // s4
            } else {
s5 :          // s5
            }
s6 :        if (z <= 3) {
s7 :          // s7
            } else {
s8 :          if (2 * x − y < 0) { //faulty,
                   should be: if (x − y < 0)
s9 :            // s9
              } else {
s10 :           // s10
              }
            }
          }
s11 :     return; // s11
        }
```

Fig. 4. Source Code

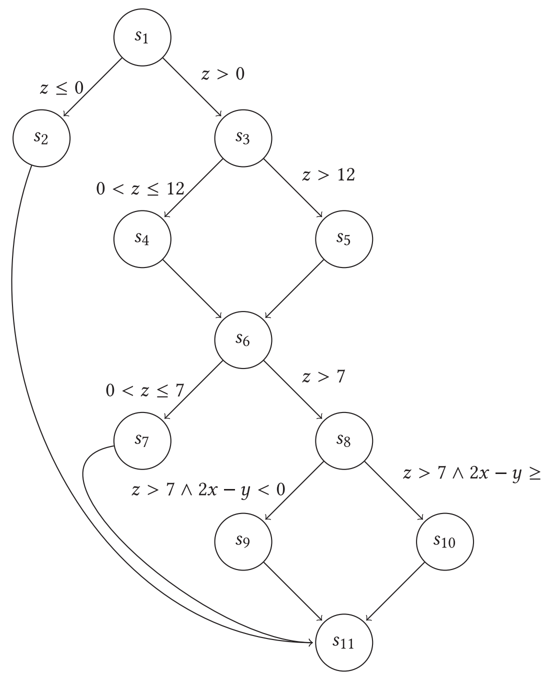


Fig. 5. Control Flow

Fig. 6. Sample program: the faulty statement s_f is s_8.

a lower Expense value than R_2; if the reverse of the original order is adopted, the opposite would follow.

On the other hand, it is also possible to construct another test suite, such that e_f^4, e_f^5, e_f^9, and e_f^{10} are smaller than F. Then, for s_1, s_3, s_6, s_8 (s_f), and s_{11} whose e_f values are all equal to F, we have $e_p^f = e_p^i < e_p^1 = e_p^3 = e_p^6 = e_p^{11} = e_p^j$. (For the feasibility of this test suite, please refer to **Test Suite A** in Example 5 of the Appendix.) Then, for R_1, we have s_1, s_3, s_6, and s_{11} tied together at the top of the whole list before s_f, and other statements ranked after s_f. However, for R_2, we have s_1, s_3, s_6, s_f, and s_{11} tied together at the top of the whole list. Since the number of tied statements are different, the Expense now depends on the tie-breaking scheme, without any guarantee of clear dominance of one formula. For example, if the original order of the statements is used as the tie-breaker, R_2 yields a lower Expense value than R_1; if the reverse of the original order is adopted, the opposite would follow.

In summary, for the case that $R_1(F, e_p^i) < R_1(F, e_p^j)$ while $R_2(F, e_p^i) = R_2(F, e_p^j)$, it is possible to demonstrate that $R_1 \nrightarrow R_2$ and $R_2 \nrightarrow R_1$.

In conclusion, for any two formulæ whose U_{R_1} and U_{R_2} are both $\emptyset$, but $P_{R_1} \neq P_{R_2}$, it follows that $R_1 \nrightarrow R_2$ and $R_2 \nrightarrow R_1$. $\square$

With above preliminaries, let us now turn to the complete analysis of the maximal and greatest formulæ of $\mathcal{F}$. First, we can define the maximality of a formula using the relationship between dominance, U_R, and the faulty border E (Lemmas 5.9 and 5.10).

PROPOSITION 5.11. *A formula R is a maximal element of $\mathcal{F}$ **if and only if** U_R is empty.*

Proof. First, it holds that if R is a maximal element of $\mathcal{F}$, then $U_R = \emptyset$. This follows immediately after Lemma 5.8.

Second, let us turn to proving that if $U_R = \emptyset$, then R is a maximal element of $\mathcal{F}$. Assume that $U_R = \emptyset$. Then, for any distinct formula R', let $P_{R'}$ denote the set of $o_p^{i,j}$ for all pairs of distinct points (F, n_p^i) and (F, n_p^j) on E (where $n_p^i < n_p^j$) and $U_{R'}$ denote the sets of points outside E that have risk values higher than or equal to those of some points (F, n_p^i) on E, for formula R'. There are following cases.

Case: $U_{R'} \neq \emptyset$. As illustrated in the proof of Lemma 5.8, it is always possible to construct another formula R'', such that $U_{R''} = \emptyset$, $R'' \rightarrow R'$ and $R' \nrightarrow R''$. If $P_{R''} = P_R$, after Lemma 5.9, then $R'' \leftrightarrow R$, and, consequently, $R' \nrightarrow R$. Otherwise, if $P_{R''} \neq P_R$, after Lemma 5.10, $R \nrightarrow R''$ and $R'' \nrightarrow R$. As a consequence, $R' \nrightarrow R$.

Case: $U_{R'} = \emptyset$. Similar to the above analysis, if $P_{R'} = P_R$, after Lemma 5.9, $R' \leftrightarrow R$. Otherwise, if $P_{R'} \neq P_R$, after Lemma 5.10, $R \nrightarrow R'$ and $R' \nrightarrow R$.

In summary, if $U_R = \emptyset$, then for any formula R', we have either $R' \leftrightarrow R$ or $R' \nrightarrow R$. After Definition 5.6, R is a maximal element of $\mathcal{F}$. □

With Proposition 5.11, which is a necessary and sufficient condition for a maximal formula of $\mathcal{F}$, there is a simple method to convert any given non-maximal formula into a maximal formula, which is effectively described in the proof of Lemma 5.8. Given any formula R that has a non-empty U_R (i.e., R is non-maximal according to Proposition 5.11), we can always convert R into a maximal formula R', where R' assigns identical risk values to points on E as R, but assigns, to all points in U_R, a constant C whose value is smaller than the risk value of any point on E.[13]

Now, let us re-visit the five maximal formulæ of the 50 investigated formulæ. After Proposition 5.11, it is possible to show that two of them (ER_1' and ER_5) are maximal but not greatest formulæ, while three of them (GP02, GP03, and GP19) are not are not maximal elements of $\mathcal{F}$, as follows:

Corollary 5.12. *ER_1' and ER_5 are maximal elements of $\mathcal{F}$.*

Proof. For any formula R in ER_1' or ER_5, $U_R = \emptyset$. After Proposition 5.11, formulæ in ER_1' and ER_5 are maximal formulæ of $\mathcal{F}$. □

Corollary 5.13. *GP02, GP03, and GP19 are not maximal elements of $\mathcal{F}$.*

Proof. Consider the program in Figure 6. The following three test suites can be constructed:

Case: for GP02. Construct a test suite that satisfies the following: $F = 2$, $P = 9$, s_8 (s_f) satisfies $e_f^f = 2 = F \wedge e_p^f = 5 < P$, and s_9 satisfies $e_f^9 = 1 < F \wedge e_p^9 = 4 < e_p^f$. Then, following the definition of GP02 (which is $2(e_f + \sqrt{n_p}) + \sqrt{e_p}$), the following risk evaluation values are obtained: $GP02(s_8) = 13$, which is smaller than $GP02(s_9) = 14$. Since s_f is on E, this shows that there can exist points outside E with risk values higher than that of the point on E. Following Proposition 5.11, GP02 is not a maximal element of $\mathcal{F}$.[14]

Case: for GP03. Construct a test suite that satisfies the following: $F = 2$, $P = 30$, s_f satisfies $e_f^f = 2 = F \wedge e_p^f = 25 < P$, and s_9 satisfies $e_f^9 = 1 < F \wedge e_p^9 = 25 = e_p^f$. Then, following the definition of GP03 (which is $\sqrt{|e_f^2 - \sqrt{e_p}|}$), the following risk evaluation values are obtained: $GP03(s_f) = 1$, which is smaller than $GP03(s_9) = 2$. Since s_f is on E, this shows that there can exist points outside E with risk values higher than that

[13] This is how GP02, GP03, and GP19 have been treated for the empirical study in Section 4.
[14] The feasibility of this scenario is analysed in Example 2 of the Appendix.

of the point on E. Following Proposition 5.11, GP03 is not the maximal to all formulæ in $\mathcal{F}$.[15]

Case: for GP19. Construct a test suite that satisfies the following: $F = 5$, $P = 50$, s_f satisfies $e_f^f = 5 = F \wedge e_p^f = 1 < P$, and s_4 satisfies $e_f^4 = 4 < F \wedge e_p^f < e_p^4 = 49 < P$. Then, following the definition of GP19 (which is $e_f\sqrt{|e_p - e_f + n_f - n_p|}$), the following risk evaluation values are obtained: $GP19(s_f) = 5\sqrt{5}$, which is smaller than $GP19(s_4) = 12\sqrt{5}$. Since s_f is on E, this shows that there can exist points outside E with risk values higher than that of the point on E. Following Proposition 5.11, GP19 is not the maximal to all formulæ in $\mathcal{F}$.[16] $\square$

With Proposition 5.11, it becomes possible to identify formulæ that are maximal elements of $\mathcal{F}$. Furthermore, within these maximal formulæ, we are interested in whether there exists the greatest formulæ and have the following conclusion.

5.3. Greatest Formulæ in $\mathcal{F}$: The Non-Existence Proof

A greatest formula in $\mathcal{F}$ is the formula that is better than any other formulæ in $\mathcal{F}$. It is formally defined as follows:

Definition 5.14 (*Greatest Formula*). A risk evaluation formula R is said to be a *greatest* formula in $\mathcal{F}$ if, for any formula $R' \in \mathcal{F} \wedge R' \neq R$, it holds that $R \rightarrow R'$.

Let us now turn to the greatest formula, or, in fact, proving the lack of thereof. We have shown the relationship between dominance, maximality, and the faulty border. Intuitively, the non-existent proof for the greatest formula shows that no single shape of faulty border can lead to a formula that dominates all other formulas. In the following proof, we use the existing maximal groups (Corollary 5.12) to construct a *reductio ad absurdum* proof; if we assume a greatest formula, it is always possible to construct another formula that dominates it, which contradicts the assumption.

PROPOSITION 5.15. *There is no formula that is greatest against the set of all formulæ, $\mathcal{F}$.*

PROOF. Assume that there exists a greatest formula R_g. Let P_{R_g} denote the set of $o_p^{i,j}$ for all pairs of distinct points (F, e_p^i) and (F, e_p^j) on E (where $e_p^i < e_p^j$) and U_{R_g} denote the sets of points outside E that have risk values higher than or equal to those of some points (F, e_p^i) on E, for formula R_g. After Proposition 5.11, $U_{R_g} = \emptyset$ and R_g is a maximal element of $\mathcal{F}$.

Consider the two maximal groups of formulæ ER_1' and ER_5, which have been proved to be non-equivalent to each other [52]. Let $U_{ER_1'}$ and U_{ER_5} denote the sets of points outside the faulty border that have risk values higher than or equal to those of some points (F, e_p^i) on the faulty border, for formulæ in ER_1' and ER_5, respectively. Let $P_{ER_1'}$ and P_{ER_5} denote the set of $o_p^{i,j}$ for all pairs of distinct points (F, e_p^i) and (F, e_p^j) on the faulty border (where $e_p^i < e_p^j$), for formulæ in ER_1' and ER_5, respectively. According to Corollary 5.12, it follows that $U_{ER_1'} = U_{ER_5} = \emptyset$ and $P_{ER_1'} \neq P_{ER_5}$. Thus, there are three possible cases for P_{R_g}, as follows:

[15]The feasibility of this scenario is analysed in Example 3 of the Appendix.
[16]The feasibility of this scenario is analysed in Example 4 of the Appendix.

Case: $P_{R_g} = P_{ER'_1}$. Then it follows that, for ER_5, $U_{ER_5} = U_{R_g} = \emptyset \wedge P_{ER_5} \neq P_{R_g}$.

Case: $P_{R_g} = P_{ER_5}$. Then it follows that, for ER'_1, $U_{ER'_1} = U_{R_g} = \emptyset \wedge P_{ER'_1} \neq P_{R_g}$.

Case: $P_{R_g} \neq P_{ER'_1}$ and $P_{R_g} \neq P_{ER_5}$. Then it follows that, both for ER'_1 and ER_5, $U_{ER'_1} = U_{R_g} = \emptyset \wedge P_{ER'_1} \neq P_{R_g} \wedge U_{ER_5} = U_{R_g} = \emptyset$ but $P_{ER_5} \neq P_{R_g}$.

For any of the above cases, it is possible to construct another formula R' such that $U_{R'} = U_{R_g} = \emptyset$ and $P_{R'} \neq P_{R_g}$. After Lemma 5.10, we have $R' \nrightarrow R_g$ and $R_g \nrightarrow R'$. After Definition 5.14, R_g cannot be the greatest formula. □

It should be noted that the above conclusion is not only consistent with but also extends existing work to a more general case. Abreu et al. first pointed out that, under the single fault scenario, to have one formula optimal, only statements with $e_f = F$ should be considered, and also that their ranking is further decided by their e_p values [1]. Later, Naish et al. reported similar insights [38]. These results share the essential idea with our Proposition 5.11. However, instead of investigating a specific set of formulæ, our theoretical observation covers any formula within the scope in Definition 2.1. This means that we can analyse any distribution of risk values on the faulty border, instead of focusing on the ones favouring lower e_p values. With Definition 2.1, it is possible to formally define "maximal" and "greatest" formulæ. As proved in Corollary 5.12, the optimal formulæ in Naish et al. [38] actually form a single equivalent group of maximal formulæ in our context. Additionally, with Lemma 5.9 and Proposition 5.11, it follows that the "optimal formula" discussed by Abreau et al. [1] is the maximal version of q_e [52], which is also equivalent to ER'_1.

6. CONCLUSIONS

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 risk evaluation formulæ to convert program spectrum data into risk scores. The main focus of the research has been the design of new risk evaluation formulæ that would outperform the existing ones.

Recently, GP has been applied to automatically evolve SBFL formulæ and was shown, empirically, to perform as well as the best known formulæ designed by humans. Subsequently, it was shown to be theoretically equivalent.

This article presents both an empirical and a theoretical study that provides evidence for the human competitiveness of GP applied to SBFL under the single fault scenario. The empirical study shows that, among the known maximal formulæ, the GP-evolved one is among those that are most practically useful. Subsequently, the theoretical study proves that there cannot exist any better formula. In summary, GP has evolved a formula that performs practically the best, and no human could ever design a formula that can outperform it.

The proof has implications and actionable conclusions for both the GP community and the software engineering community. For the GP community, it shows the human competitiveness of GP: it has been proven to have accomplished what has been the aim of SBFL researchers for over a decade. For the SBFL research community, it shows that pursuing the greatest formula is no longer a viable research goal. Future work on fault localisation will be encouraged to consider specialisation, that is, designing formulæ that are effective in certain contexts, such as a specific project or a particular type of faults. In the wider context, the proof illustrates the limitations of the spectrum-based approaches, and encourages fault localisation techniques to consider signals other than program spectrum. For both specialisation and post-spectrum fault localisation, GP will remain an effective methodology to develop a competitive technique.

APPENDIX

Analysis of the Example Program

In the proof, the program in Figure 6 has been used as an example. The program accepts three independent real numbers: x, y and z. The statement s_8 contains a fault: the correct predicate should be "`if(x-y<0)`" not "`if(2x-y<0)`" and, therefore, is denoted by s_f. Figure 7 shows the corresponding regions of failure inducing inputs in the space of all possible inputs.

First, consider statements s_9 and s_{10}:

- Any test case $t_i = (x_i, y_i, z_i)$ such that $z_i > 7 \wedge (x_i, y_i) \in Fail_9$ will cover s_9 and fail. Let the number of such test cases be e_f^9.
- Any test case $t_i = (x_i, y_i, z_i)$ such that $z_i > 7 \wedge (x_i, y_i) \in Pass_9$ will cover s_9 and pass. Let the number of such test cases be e_p^9.
- Any test case $t_i = (x_i, y_i, z_i)$ such that $z_i > 7 \wedge (x_i, y_i) \in Fail_{10}$ will cover s_{10} and fail. Let the number of such test cases be e_f^{10}.
- Any test case $t_i = (x_i, y_i, z_i)$ such that $z_i > 7 \wedge (x_i, y_i) \in Pass_{10}$ will cover s_{10} and pass.Let the number of such test cases be e_p^{10}.

Since s_9 and s_{10} are the **`true`** and **`false`** branches of the **`if`** statement in s_f, only one of these two statements are executed by any test case. Consequently, $e_f^9 = n_f^{10}$, $n_f^9 = e_f^{10}$, $e_p^9 = n_p^{10}$, and $n_p^9 = e_p^{10}$. There is *no* constraint in selecting test cases from each of the above four regions, and generating test cases from these regions is independent from each other. Therefore, by adjusting the number of test cases in each region, it is possible to have values of e_f^9, e_p^9, e_f^{10}, and e_p^{10}.

Next, consider statement s_8 (i.e., s_f). It is possible to have any number of passing (i.e., e_p^f) and failing (i.e., $e_f^f = F$) test cases by adjusting the above four sets of test cases, because we have $e_f^9 + e_f^{10} = e_f^f = F$ and $e_p^9 + e_p^{10} = e_p^f$.

Then, let us consider statement s_7. Any test case $t_i =< x_i, y_i, z_i >$ such that $0 < z_i \leq 7$ will cover s_7 and pass: e_p^7 is equal to the number of such test cases, while $e_f^7 = 0$. It is possible to have any number of such test cases. Therefore, by adjusting this number, it is possible to assign any value to e_p^7.

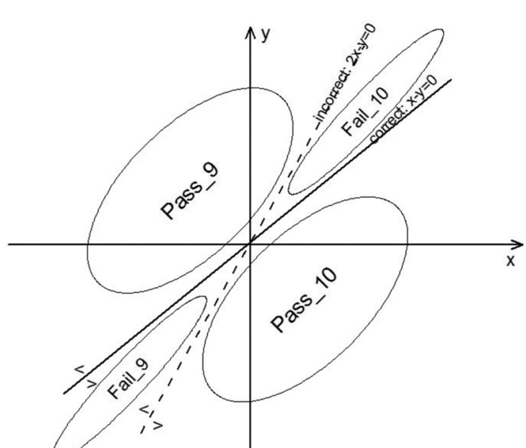


Fig. 7. Sample program.

Now, consider statements s_6, s_{11}, and s_3, whose e_f values are identical to each other, and so are their e_p. It can be seen from Figure 6 that $e_f^6 = e_f^{11} = e_f^3 = e_f^f = F$ and $e_p^6 = e_p^3 = e_p^{11} = (e_p^f + e_p^7)$. As a reminder, according to the above analysis, e_p^f and e_p^7 can be assigned with any values independently.

Next, consider statements s_4 and s_5. As shown in Figure 6, any test case $t_i = <x_i, y_i, z_i>$ such that $0 < z_i \leq 12$ will cover s_4. These test cases can be further categorised as following:

- Any test case $t_i =< x_i, y_i, z_i >$ such that $0 < z_i \leq 7$ will definitely continue to cover s_7 and thus always pass. The number of these test cases is equal to e_p^7.
- Any test case $t_i =< x_i, y_i, z_i >$ such that $7 < z_i \leq 12$ while $< x_i, y_i >\in Pass_9 \cup Pass_{10}$ will also pass. Let us denote the number of these test cases as $\overline{e_p^4}$.
- Any test case $t_i =< x_i, y_i, z_i >$ such that $7 < z_i \leq 12$ while $< x_i, y_i >\in Fail_9 \cup Fail_{10}$ will always fail. The number of these test cases is e_f^4.

It is not difficult to find that e_f^4 is the size of the subset of failing test cases that cover s_f and satisfy $7 < z_i \leq 12$. Consequently, it follows that $e_f^4 \leq F$. On the other hand, $e_p^4 = e_p^7 + \overline{e_p^4}$, where $\overline{e_p^4}$ is the size of the subset of passing test cases that cover and satisfy $7 < z_i \leq 12$. Thus, it also follows that $\overline{e_p^4} \leq e_p^f$. As a reminder, the values of e_p^7 and $\overline{e_p^4}$ can be decided independently. Therefore, e_p^4 can be either smaller than, equal to or greater than e_p^f.

While for s_5, any test case $t_i =< x_i, y_i, z_i >$ such that $z_i > 12$ will cover s_5. These test cases can be further categorised as following:

- Any test case $t_i =< x_i, y_i, z_i >$ such that $z_i > 12$ while $< x_i, y_i >\in Pass_9 \cup Pass10$ will always pass. The number of these test cases is e_p^5.
- Any test case $t_i =< x_i, y_i, z_i >$ such that $z_i > 12$ while $< x_i, y_i >\in Fail_9 \cup Fail10$ will always fail. The number of these test cases is e_f^5.

Note that e_p^5 is the size of the subset of passing test cases that cover s_f and satisfy $z_i > 12$, while e_f^5 is the size of the subset of failing test cases that cover s_f and satisfy $z_i > 12$. Thus, it follows that $e_f^5 \leq e_f^f = F$ and $e_p^5 \leq e_p^f$.

It should be noted that e_f^4 and e_f^5 are not independent. There is a constraint that $e_f^4 + e_f^5 = e_f^f = F$. However, there is no similar constraint on e_p^4 and e_p^5.

Next, let us consider s_2. As shown in Figure 6, any test case $t_i =< x_i, y_i, z_i >$ such that $z_i \leq 0$ will cover s_2 and pass: e_p^2 is equal to the number of such test cases, while e_f^2 is always 0. It is possible to have any number of such test cases. By adjusting this number, it is possible to assign any value to e_p^2.

Finally, let us consider s_1. The structure of the program dictates that $e_f^1 = e_f^f = F$ and $e_p^1 = (e_p^3 + e_p^2) = (e_p^f + e_p^7 + e_p^2) = P$. As analysed above, it is possible to assign, independently, any values to e_p^f, e_p^7 and e_p^2. Consequently, e_p^1 (i.e., the number of total passing test cases P) can be any value that is no less than e_p^f.

Feasibility of Test Suites Used in Proofs

This section presents the analysis of the feasibility of the example test suites used in the proofs.

Example 1. With the program in Figure 6, the proof in Proposition 5.11 requires the construction of a test suite such that e_p^f and e_p^4 are any values, and $e_f^4 < F$. According the above discussion, for s_f, we can assign any value to e_p^f; while for s_4, e_f^4 can be any value within [0, F] and e_p^4 can be either smaller than, equal to or greater than e_p^f. Therefore, it is always possible to construct such a test suite.

Example 2. With the program in Figure 6, the proof for GP2 in Corollary 5.13 requires the construction of a test suite such that $F = 2$, $P = 9$, s_8 (s_f) has $e_f^f = 2 = F$ and $e_p^f = 5 < P$, and s_9 has $e_f^9 = 1 < F$ and $e_p^9 = 4 < e_p^f$. According to the above analysis, we can assign any value to e_p^f and any value to P that is no less than e_p^f. And for s_9, we can have any value of e_f^9 within [0, F] and any value of e_p^9 within [0, e_p^f]. Therefore, it is always possible to construct such a test suite.

Example 3. With the program in Figure 6, the proof for GP3 in Corollary 5.13 requires the construction of a test suite such that $F = 2$, $P = 30$, s_f has $e_f^f = 2 = F$ and $e_p^f = 25 < P$, and s_9 has $e_f^9 = 1 < F$ and $e_p^9 = 25 = e_p^f$. Similar to the analysis in Example 2, it is always possible to construct such a test suite.

Example 4. With the program in Figure 6, the proof for GP19 in Corollary 5.13 requires the construction of a test suite such that $F = 5$, $P = 50$, s_f has $e_f^f = 5 = F$ and $e_p^f = 1 < P$, and s_4 has $e_f^4 = 4 < F$ and $e_p^f < e_p^4 = 49 < P$. According to the above analysis, for s_f, we can assign any values to e_f^f (i.e., F) and e_p^f; while for s_4, e_f^4 can be any value within [0, F] and e_p^4 can be either smaller than, equal to or greater than e_p^f; and P can be any value that is no less than either e_p^4 or e_p^f. Therefore, it is always possible to construct such a test suite.

Example 5. Let us denote the e_p values of any two points on the faulty border E as e_p^L and e_p^H, where $e_p^L < e_p^H$. For the given program in Figure 6, the proof in Lemma 5.10 requires construction of two test suites, which are referred to as **Test Suite A**, **Test Suite B**, and **Test Suite C** in the following discussion.

Test Suite A: We have e_f^4, e_f^5, e_f^9 and e_f^{10} smaller than F, $e_f^1 = e_f^3 = e_f^6 = e_f^f = e_f^{11} = F$, $e_p^f = e_p^L$ and $e_p^1 = e_p^3 = e_p^6 = e_p^{11} = e_p^H$. According to the above analysis, e_f^4, e_f^5, e_f^9, and e_f^{10} can all be less than F simultaneously. And the e_f values for s_1, s_3, s_6, s_f, and s_{11} are always equal to F. Besides, for s_f, it is always possible to have any value of e_p^f; while it is always possible to have any equal value of e_p that is larger than e_p^f for s_1, s_3, s_6, and s_{11}. Therefore, this test suite is feasible.

Test Suite B: We have both e_f^4 and e_f^5 smaller than F, $e_f^1 = e_f^3 = e_f^6 = e_f^f = e_f^9 = e_f^{11} = F$, $e_p^9 = e_p^L$ and $e_p^1 = e_p^3 = e_p^6 = e_p^f = e_p^{11} = e_p^H$. As discussed above, it is always possible to have both e_f^4 and e_f^5 smaller than F. And it is also possible to assign the same value of e_f (i.e., F) to s_1, s_3, s_6, s_f, s_9, and s_{11}. Moreover, s_1, s_3, s_6, s_f, and s_{11} are possible to have the same e_p value that is higher than s_9. As a consequence, this test suite is always feasible.

Test Suite C: We have e_f^9 and e_f^{10} smaller than F, $e_f^1 = e_f^3 = e_f^4 = e_f^6 = e_f^f = e_f^{11} = F$, $e_p^4 = e_p^L$ and $e_p^1 = e_p^3 = e_p^6 = e_p^f = e_p^{11} = e_p^H$. As discussed above, it is always possible to have both e_f^9 and e_f^{10} smaller than F. And it is also possible to assign the same value of e_f (i.e., F) to s_1, s_3, s_4, s_6, s_f, and s_{11}. Moreover, s_1, s_3,

s_6, s_f, and s_{11} are possible to have the same e_p value that is higher than s_4. As a consequence, this test suite is always feasible.

REFERENCES

[1] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. 2009. Spectrum-based multiple fault localization. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE'09)*. 88–99.

[2] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. 2006. An evaluation of similarity coefficients for software fault localization. In *The Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. IEEE, 39–46.

[3] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. 2007. On the accuracy of spectrum-based fault localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques—MUTATION*. IEEE Computer Society, 89–98.

[4] Wasif Afzal, Richard Torkar, and Robert Feldt. 2009. A systematic review of search-based testing for non-functional system properties. *Info. Softw. Technol.* 51, 6 (2009), 957–976.

[5] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. ACM, New York, 1–10.

[6] Shay Artzi, Julian Dolby, Frank Tip, and Marco Pistoia. 2010. Directed test generation for effective fault localization. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA'10)*. ACM, New York, 49–60.

[7] Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke. 2015. Automated software transplantation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA'15)*. ACM, New York, 257–269. DOI:http://dx.doi.org/10.1145/2771783.2771796

[8] Mariano Ceccato, Alessandro Marchetto, Leonardo Mariani, Cu D. Nguyen, and Paolo Tonella. 2015. Do automatically generated test cases make debugging easier? An experimental assessment of debugging effectiveness and efficiency. *ACM Trans. Softw. Eng. Methodol.* 25, 1 (Dec. 2015), 5:1–5:38.

[9] Yanping Chen, Robert L. Probert, and D. Paul Sims. 2002. Specification-based regression test selection with risk analysis. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative research (CASCON'02)*. IBM Press, 1–14.

[10] Gregory W. Corder and Dale I. Foreman. 2009. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley.

[11] Valentin Dallmeier, Christian Lindig, and Andreas Zeller. 2005. Lightweight bug localization with AMPLE. In *Proceedings of the 6th International Symposium on Automated Analysis-driven Debugging (AADEBUG'05)*. ACM, New York, 99–104.

[12] Nicholas DiGiuseppe and James A. Jones. 2011. On the influence of multiple faults on coverage-based fault localization. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA'11)*. ACM, New York, NY, USA, 210–220.

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

[14] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer, and Claire Le Goues. 2009. A genetic programming approach to automated software repair. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)*. ACM, New York, 947–954.

[15] Gordon Fraser, Matt Staats, Phil McMinn, Andrea Arcuri, and Frank Padberg. 2013. Does automated white-box test generation really help software testers? In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'13)*. 291–301.

[16] Fabrício Gomes Freitas and Jerffeson Teixeira Souza. 2011. Ten years of search-based software engineering: A bibliometric analysis. In *Search-Based Software Engineering*, MyraB. Cohen and Mel Ó Cinnéide (Eds.). Lecture Notes in Computer Science, Vol. 6956. Springer, Berlin, 18–32.

[17] A. Gonzalez-Sanchez, R. Abreu, H. G. Gross, and A. J. C. van Gemund. 2011b. Prioritizing tests for fault localization through ambiguity group reduction. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*. 83–92.

[18] Alberto Gonzalez-Sanchez, Rui Abreu, Hans-Gerhard Gross, and Arjan J. C. van Gemund. 2011a. Spectrum-based sequential diagnosis. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI'11)*. AAAI Press, 189–196.

[19] A. Gonzalez-Sanchez, E. Piel, H. G. Gross, and A. J. C. van Gemund. 2010. Prioritizing tests for software fault localization. In *Proceedings of the 2010 10th International Conference on Quality Software*. 42–51.

[20] C. Gouveia, J. Campos, and R. Abreu. 2013. Using HTML5 visualizations in software fault localization. In *Proceedings of the 1st IEEE Working Conference on Software Visualization (VISSOFT'13)*. 1–10.

[21] Dan Hao, Lu Zhang, Ying Pan, Hong Mei, and Jiasu Sun. 2008. On similarity-awareness in testing-based fault localization. *Auto. Softw. Eng.* 15 (June 2008), 207–249. Issue 2.

[22] Mark Harman. 2011. Software engineering meets evolutionary computation. *IEEE Comput.* 44, 10 (Oct. 2011), 31–39.

[23] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *Comput. Surveys* 45, 1, Article 11 (December 2012), 61 pages.

[24] Mary Jean Harrold, Gregg Rothermel, Rui Wu, and Liu Yi. 1998. An empirical investigation of program spectra. In *Proceedings of the ACM SIGPLAN-SIGSOFT workshop on Program Analysis for Software Tools and Engineering (PASTE'98)*. ACM, New York, 83–90.

[25] Paul Jaccard. 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles* 37 (1901), 547–579.

[26] Wei Jin and Alessandro Orso. 2012. BugRedux: Reproducing field failures for in-house debugging. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*. IEEE Press, Piscataway, NJ, 474–484.

[27] Wei Jin and Alessandro Orso. 2013. F3: Fault localization for field failures. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA'13)*. ACM, New York, 213–223.

[28] James A. Jones and Mary Jean Harrold. 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th International Conference on Automated Software Engineering (ASE'05)*. ACM, 273–282.

[29] James A. Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*. ACM, New York, 467–477.

[30] James A. Jones, Mary Jean Harrold, and John T. Stasko. 2001. Visualization for fault localization. In *Proceedings of ICSE Workshop on Software Visualization*. 71–75.

[31] J. R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

[32] Claire Le Goues, Stephanie Forrest, and Westley Weimer. 2013. Current challenges in automatic software repair. *Softw. Qual. J.* 21, 3 (2013), 421–443.

[33] Hua Jie Lee. 2011. *Software Debugging using Program Spectra*. Ph.D. Dissertation. University of Melbourne.

[34] W. Masri and R. A. Assi. 2010. Cleansing test suites from coincidental correctness to enhance fault-localization. In *Proceedings of the 2010 3rd International Conference on Software Testing, Verification and Validation (ICST'10)*. 165–174.

[35] Philip McMinn. 2004. Search-based software test data generation: A survey. *Softw. Test. Verificat. Reliabil.* 14, 2 (June 2004), 105–156.

[36] Seokhyeon Moon, Yunho Kim, Moonzoo Kim, and Shin Yoo. 2014. Ask the mutants: Mutating faulty programs for fault localization. In *Proceedings of the 7th International Conference on Software Testing, Verification and Validation (ICST'14)*. 153–162.

[37] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2011. A model for spectra-based software diagnosis. *ACM Trans. Softw. Eng. Methodol.* 20, 3, Article 11 (August 2011), 32 pages.

[38] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2012. Spectral debugging: How much better can we do? In *Proceedings of the 35th Australasian Computer Science Conference—Volume 122 (ACSC'12)*. Australian Computer Society, Inc., Darlinghurst, Australia, 99–106.

[39] A. Ochiai. 1957. Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions. *Bull. Japan. Soc. Sci. Fish.* 22, 9 (1957), 526–530.

[40] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE'13)*. IEEE, 522–531.

[41] Sangmin Park, Richard W. Vuduc, and Mary Jean Harrold. 2010. Falcon: Fault localization in concurrent programs. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE'10)*. ACM, New York, 245–254.

[42] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA'11)*. ACM, New York, 199–209.

[43] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming*. Published via http://lulu.com and retrieved from http://www.gp-field-guide.org.uk (with contributions by J. R. Koza).

[44] Yuhua Qi, Xiaoguang Mao, Yan Lei, and Chengsong Wang. 2013. Using automated program repair for evaluating the effectiveness of fault localization techniques. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA'13)*. ACM, New York, 191–201.

[45] M. Renieres and S. P. Reiss. 2003. Fault localization with nearest neighbor queries. In *Proceedings of the 18th International Conference on Automated Software Engineering*. 30–39.

[46] P. F. Russel and T. Ramachandra Rao. 1940. On habitat and association of species of anopheline larvae in south-eastern Madras. *J. Malar. Inst. India* 3, 1 (1940), 153–178.

[47] SLOCCount. 2004. Retrieved from http://www.dwheeler.com/sloccount/sloccount.html (2004).

[48] Friedrich Steimann, Marcus Frenkel, and Rui Abreu. 2013. Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA'13)*. ACM, New York, 314–324.

[49] András Vargha and Harold D. Delaney. 2000. A critique and improvement of the "CL" common language effect size statistics of McGraw and Wong. *J. Educat. Behav. Stat.* 25, 2 (2000), pp. 101–132.

[50] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically finding patches using genetic programming. In *Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE'09)*. IEEE.

[51] W. Eric Wong, Yu Qi, Lei Zhao, and Kai-Yuan Cai. 2007. Effective fault localization using code coverage. In *Proceedings of the 31st Annual International Computer Software and Applications Conference—Volume 01 (COMPSAC'07)*. IEEE Computer Society, Washington, DC, 449–456.

[52] Xiaoyuan Xie, Tsong Yueh Chen, Fei-Ching Kuo, and Baowen Xu. 2013a. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Trans. Softw. Eng. Methodol.* 22, 4, Article 31 (October 2013), 40 pages.

[53] Xiaoyuan Xie, Fei-Ching Kuo, Tsong Yueh Chen, Shin Yoo, and Mark Harman. 2013b. Provably optimal and human-competitive results in SBSE for spectrum-based fault localisation. In *Search-Based Software Engineering*, Günther Ruhe and Yuanyuan Zhang (Eds.). Lecture Notes in Computer Science, Vol. 8084. Springer, Berlin, 224–238.

[54] Jian Xu, W. K. Chan, Zhenyu Zhang, T. H. Tse, and Shanping Li. 2011. A dynamic fault localization technique with noise reduction for java programs. In *Proceedings of the 11th International Conference on Quality Software*, Manuel Núñez, Robert M. Hierons, and Mercedes G. Merayo (Eds.). IEEE Computer Society, 11–20.

[55] Shin Yoo. 2012. Evolving human competitive spectra-based fault localisation techniques. In *Search-Based Software Engineering*, Gordon Fraser and Jerffeson Teixeira de Souza (Eds.). Lecture Notes in Computer Science, Vol. 7515. Springer, Berlin, 244–258.

[56] Shin Yoo, Mark Harman, and David Clark. 2013. Fault localization prioritization: Comparing information-theoretic and coverage-based approaches. *ACM Trans. Softw. Eng. Methodol.* 22, 3 (July 2013), 19:1–19:29.

[57] Yanbing Yu, James A. Jones, and Mary Jean Harrold. 2008. An empirical study of the effects of test-suite reduction on fault localization. In *Proceedings of the International Conference on Software Engineering (ICSE'08)*. ACM, 201–210.

Received December 2015; revised January 2017; accepted March 2017