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, Swinburne University of Technology, Australia
TSONG YUEH CHEN, Swinburne University of Technology, Australia
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 formulae 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.

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:
https://doi.org/0000001.0000001

1 INTRODUCTION

This paper 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.

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

Spectrum-Based Fault Localisation (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 paper 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 been executed by tests, with fail and pass as a result respectively. 

Table 1. Motivating Example: the faulty statement $s_7$ achieves the first place when ranked according to the Tarantula risk evaluation formula in Equation 1.

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

\[
\text{Tarantula} = \frac{e_f}{e_f + n_f} = \frac{e_p + n_p}{e_f + n_f} + \frac{e_f}{e_f + n_f}
\] (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 1 presents an illustrative example of the Tarantula 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 3 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 $\frac{\frac{1}{2}}{\frac{1}{2} + \frac{1}{2}} = 0.5$ to $s_1$, and the score $\frac{\frac{1}{2}}{\frac{1}{2} + \frac{1}{2}} = 1.0$ to $s_7$. SBFL technique assumes that the developer is to investigate the SUT following the rank order produced by the technique. In 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

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

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

\[ R = \{ R : I \times I \times I \times I \rightarrow \text{Real} \} \]

(where \( I \) denotes the set of non-negative integers and \( \text{Real} \) denotes the set of real numbers), which maps \( A_i = \langle e_i^f, e_i^p, n_i^f, n_i^p \rangle \) 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 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, i.e. 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:

\[ \text{Expense} = \frac{\text{Ranking of the faulty statement}}{\text{Number of statements in the program}} \] (2)

The metric itself assumes a specific mode of usage of the results, i.e. 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].
Human Competitiveness of Genetic Programming in Spectrum Based Fault Localisation

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:

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

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 paper, 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 [48]. Intuitively, to be useful, an SBFL technique should produce the ranking of the faulty statement with a lower bound that is higher than $\frac{n-1}{2}$; 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 paper 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, in order to make the paper 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, i.e. 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 that 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 paper. For readers who are interested in the justifications, validity and impacts of the above assumptions, please refer to the previous work [52].

\(^3\)However, this does not mean that SBFL cannot be applied to non-deterministic faults.
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.

**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 \( PG =< s_1, s_2, ..., s_n > \), a test suite of \( m \) test cases \( TS = \{t_1, t_2, ..., t_m\} \), and a risk evaluation formula \( R \), which assigns a risk value to each program statement. For each statement \( s_i \), a spectrum vector \( \sigma(s_i) =< e_{f_i}^1, e_{p_i}^1, n_{f_i}^1, n_{p_i}^1 > \) can be constructed from \( TS \), and \( R(e_{f_i}^1, e_{p_i}^1, n_{f_i}^1, n_{p_i}^1) \) 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 \( 1 \leq i \leq n \)):

\[
S_B^R = \{ s_i \in S | R(e_{f_i}^1, e_{p_i}^1, n_{f_i}^1, n_{p_i}^1) > R(e_{f_f}^1, e_{p_f}^1, n_{f_f}^1, n_{p_f}^1) \} \\
S_F^R = \{ s_i \in S | R(e_{f_i}^1, e_{p_i}^1, n_{f_i}^1, n_{p_i}^1) = R(e_{f_f}^1, e_{p_f}^1, n_{f_f}^1, n_{p_f}^1) \} \\
S_A^R = \{ s_i \in S | R(e_{f_i}^1, e_{p_i}^1, n_{f_i}^1, n_{p_i}^1) < R(e_{f_f}^1, e_{p_f}^1, n_{f_f}^1, n_{p_f}^1) \}
\]

That is, statements in \( S_B^R \) have higher risk values than \( s_f \), and thus are all ranked above any statements in \( S_F^R \); statements in \( S_F^R \) 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 \( S_A^R \) have lower risk values than \( s_f \) and, thus, are all ranked below any statements in \( S_F^R \).

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 $S_{R_1}^B \subseteq S_{R_2}^B \land S_{R_1}^F \subseteq S_{R_2}^F$, 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 $S_{R_1}^B = S_{R_2}^B \land S_{R_1}^F = S_{R_2}^F$, then $R_1 \leftrightarrow R_2$.

Let us briefly sketch the proof for Theorem 2.6. Consider another formula $R_3$, such that for any program, faulty statement $s_f$, and test suite, $S_{R_3}^B = S_{R_1}^B \land S_{R_3}^F \subseteq S_{R_2}^F$ and $S_{R_3}^A \subseteq S_{R_1}^A \land S_{R_2}^A = S_{R_3}^A$. The assumption about a consistent tie-breaking scheme implies that, within the equal subset of a fault, $S_{R_1}^F$, 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].

### Table 2. Known and novel Maximal SBFL formulæ

<table>
<thead>
<tr>
<th>Equivalence Group</th>
<th>Formula</th>
<th>Expression</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ER'_1$</td>
<td>Naish1 [37]</td>
<td>$\begin{cases} -1 &amp; \text{if } e_f &lt; F \ n_p &amp; \text{if } e_f \geq F \end{cases}$</td>
<td>Human</td>
</tr>
<tr>
<td></td>
<td>Naish2 [37]</td>
<td>$e_f - \frac{e_p}{e_f + n_p}$</td>
<td>Human</td>
</tr>
<tr>
<td></td>
<td>GP13 [55]</td>
<td>$e_f(1 + \frac{1}{2e_f + e_p})$</td>
<td>GP</td>
</tr>
<tr>
<td>$ER_2$</td>
<td>Wong1 [51]</td>
<td>$e_f$</td>
<td>Human</td>
</tr>
<tr>
<td></td>
<td>Russel &amp; Rao [46]</td>
<td>$\frac{e_f}{e_f + n_f + n_p}$</td>
<td>Human</td>
</tr>
<tr>
<td></td>
<td>Binary [37]</td>
<td>$\begin{cases} 0 &amp; \text{if } e_f &lt; F \ 1 &amp; \text{if } e_f = F \end{cases}$</td>
<td>Human</td>
</tr>
<tr>
<td></td>
<td>GP02 [55]</td>
<td>$2(e_f + \sqrt{n_p}) + \sqrt{e_p}$</td>
<td>GP</td>
</tr>
<tr>
<td></td>
<td>GP03 [55]</td>
<td>$\sqrt{</td>
<td>e_f^2 - e_p^2</td>
</tr>
<tr>
<td></td>
<td>GP19 [55]</td>
<td>$e_f\sqrt{</td>
<td>e_p - e_f + n_f - n_p</td>
</tr>
</tbody>
</table>

The definition of limited maximality, i.e. 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 $R_2 \leftrightarrow R_1$. 

ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 4, Article 39. Publication date: March 2010.
Table 2 shows the two groups of maximal formulæ, ER$_1$ and ER$_3$, 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 3. 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 3. List of GP operators used by Yoo [55]

<table>
<thead>
<tr>
<th>Operator Node (a, b)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp_add(a, b)</td>
<td>a + b</td>
</tr>
<tr>
<td>gp_sub(a, b)</td>
<td>a - b</td>
</tr>
<tr>
<td>gp_mul(a, b)</td>
<td>ab</td>
</tr>
<tr>
<td>gp_div(a, b)</td>
<td>1 if b = 0, ( \frac{a}{b} ) otherwise</td>
</tr>
<tr>
<td>gp_sqrt(a)</td>
<td>( \sqrt{</td>
</tr>
</tbody>
</table>

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 2, 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 2) that is equivalent to the known maximal formulæ designed manually [53]. Other evolved formulæ formed their own maximal groups, such as GP02, GP03, and GP19 in Table 2.

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 Paper 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 paper, 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 paper 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.

4 EMPIRICAL STUDY

The existing theoretical analysis [52, 53] shows that maximal formulæ among the known 50 formulæ form 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 two-fold. 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 in Table 2 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

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 $GP_{M2}^M$, $GP_{M3}^M$, and $GP_{M19}^M$ from now on.
Table 4. Subject programs

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

4.1.1 Subject Programs. In these experiments, we use five subject programs from Software Infrastructure Repository (SIR) [13]. Table 4 describes the functionalities and sizes of these programs: the size is measured in Source Lines of Code, excluding whitespaces, using SLOCCount [47]5. Table 4 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 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 ranked6. 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. Figures 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 -
Fig. 1. Boxplots of Expense metric from the subject programs

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 $GP_{19}M$ and $ER_1'$ are 7.94% and

ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 4, Article 39. Publication date: March 2010.
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. $ER_1'$):

Table 5. Wilcoxon-Signed-Rank Test $p$-values after Bonferroni correction and $A_{12}$ Statistics

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>2-tailed</th>
<th>1-tailed, L</th>
<th>1-tailed, U</th>
<th>$A_{12}$</th>
<th>2-tailed</th>
<th>1-tailed, L</th>
<th>1-tailed, U</th>
<th>$A_{12}$</th>
<th>2-tailed</th>
<th>1-tailed, L</th>
<th>1-tailed, U</th>
<th>$A_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>flex</td>
<td></td>
<td></td>
<td></td>
<td>grep</td>
<td></td>
<td></td>
<td></td>
<td>gzip</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$GP_{13}$</td>
<td>3.59E-07</td>
<td>1.00E+00</td>
<td>1.79E-07</td>
<td>0.36</td>
<td>4.84E-05</td>
<td>1.00E+00</td>
<td>2.42E-05</td>
<td>0.36</td>
<td>5.16E-07</td>
<td>1.00E+00</td>
<td>2.58E-07</td>
<td>0.34</td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$GP_{13}$</td>
<td>2.00E-02</td>
<td>1.00E+00</td>
<td>1.00E-02</td>
<td>0.45</td>
<td>3.66E-02</td>
<td>1.00E+00</td>
<td>1.83E-02</td>
<td>0.45</td>
<td>1.50E-06</td>
<td>1.00E+00</td>
<td>7.51E-07</td>
<td>0.36</td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$GP_{13}$</td>
<td>7.38E-05</td>
<td>1.00E+00</td>
<td>3.69E-05</td>
<td>0.41</td>
<td>1.91E-01</td>
<td>1.00E+00</td>
<td>9.54E-02</td>
<td>0.44</td>
<td>7.63E-02</td>
<td>1.00E+00</td>
<td>3.82E-02</td>
<td>0.46</td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$ER_3$</td>
<td>3.00E-11</td>
<td>1.00E+00</td>
<td>1.50E-11</td>
<td>0.13</td>
<td>1.99E-12</td>
<td>1.00E+00</td>
<td>9.96E-13</td>
<td>0.18</td>
<td>3.79E-04</td>
<td>1.00E+00</td>
<td>1.90E-04</td>
<td>0.35</td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$GP_{13}^M$</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.57</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.59</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.50</td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$GP_{13}^M$</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.53</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.55</td>
<td>5.04E-02</td>
<td>2.52E-02</td>
<td>1.00E+00</td>
<td>0.60</td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$ER_3$</td>
<td>3.56E-08</td>
<td>1.00E+00</td>
<td>1.78E-08</td>
<td>0.20</td>
<td>4.08E-08</td>
<td>1.00E+00</td>
<td>2.04E-08</td>
<td>0.23</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.51</td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$GP_{13}^M$</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.47</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.47</td>
<td>1.53E-02</td>
<td>7.64E-03</td>
<td>1.00E+00</td>
<td>0.59</td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$ER_3$</td>
<td>7.67E-05</td>
<td>1.00E+00</td>
<td>3.83E-05</td>
<td>0.24</td>
<td>1.00E-07</td>
<td>1.00E+00</td>
<td>5.00E-08</td>
<td>0.20</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.50</td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$ER_3$</td>
<td>7.87E-03</td>
<td>1.00E+00</td>
<td>3.94E-03</td>
<td>0.29</td>
<td>7.61E-04</td>
<td>1.00E+00</td>
<td>3.88E-04</td>
<td>0.31</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>8.12E-01</td>
<td>0.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sed</td>
<td></td>
<td></td>
<td></td>
<td>space</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$GP_{13}^M$</td>
<td>1.33E-05</td>
<td>1.00E+00</td>
<td>6.64E-06</td>
<td>0.34</td>
<td>1.99E-13</td>
<td>1.00E+00</td>
<td>9.94E-14</td>
<td>0.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$GP_{13}^M$</td>
<td>3.85E-04</td>
<td>1.00E+00</td>
<td>1.92E-04</td>
<td>0.42</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$GP_{13}^M$</td>
<td>1.96E-03</td>
<td>1.00E+00</td>
<td>9.82E-04</td>
<td>0.45</td>
<td>1.09E-09</td>
<td>1.00E+00</td>
<td>5.46E-10</td>
<td>0.38</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ER_1'$</td>
<td>$ER_3$</td>
<td>3.54E-14</td>
<td>1.00E+00</td>
<td>1.77E-14</td>
<td>0.18</td>
<td>4.82E-16</td>
<td>1.00E+00</td>
<td>2.41E-16</td>
<td>0.07</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$GP_{13}^M$</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.56</td>
<td>3.70E-07</td>
<td>1.85E-07</td>
<td>1.00E+00</td>
<td>0.76</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$ER_3$</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.59</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$GP_{13}^M$</td>
<td>7.83E-08</td>
<td>1.00E+00</td>
<td>3.91E-08</td>
<td>0.29</td>
<td>1.42E-11</td>
<td>1.00E+00</td>
<td>7.10E-12</td>
<td>0.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$ER_3$</td>
<td>5.49E-01</td>
<td>2.75E-01</td>
<td>1.00E+00</td>
<td>0.54</td>
<td>2.15E-03</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.39</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$ER_3$</td>
<td>9.80E-01</td>
<td>1.00E+00</td>
<td>4.89E-01</td>
<td>0.32</td>
<td>7.61E-12</td>
<td>1.00E+00</td>
<td>3.80E-12</td>
<td>0.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$GP_{13}^N$</td>
<td>$ER_3$</td>
<td>5.24E-07</td>
<td>1.00E+00</td>
<td>2.62E-07</td>
<td>0.26</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>1.00E+00</td>
<td>0.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $ER_1'$ vs. $ER_3$: 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. $GP_{13}^M$: at points Q1 and median in grep and space.
- $ER_1'$ vs. $GP_{13}^M$: at point Q3 in program sed, and at the maximum point in space.
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 the null hypothesis that the sample comes from a population which has a normal distribution. Consequently, we use IQR as the measure of dispersal. Figure 1 shows that \( ER' \) values are normally distributed: the results, presented in Table 6, suggest that we can reject the null hypothesis that the Expense values of the five maximal formulæ and groups. 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 6, suggest that we can reject the null hypothesis that the sample comes from a population which has a normal distribution. Consequently, we use IQR as the measure of dispersal. Figure 1 shows that \( ER' \) has the smallest dispersal, which means \( ER' \) not only performs the best, but also performs the most consistently. On the other hand, \( GP' \) and \( GP'' \) tend to produce very similar Expense values that are higher than \( ER' \) but lower than those of \( GP' \). 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 \( flex \) and \( sed \). For the remaining three programs, \( GP' \), \( GP'' \), and \( GP''' \) perform the worst. Formulæ \( GP'M' \) and \( GP''M' \) do not follow the null hypothesis that the sample comes from a population which has a normal distribution. Consequently, we use IQR as the measure of dispersal. Figure 1 shows that \( ER' \) has the smallest dispersal, which means \( ER' \) not only performs the best, but also performs the most consistently. On the other hand, \( GP' \) and \( GP'' \) have the most unstable performance among the maximal formulæ and groups. While \( GP'M' \) and \( ER_5 \) can deliver relatively stable performance for \( flex \) and \( sed \), they behave more similarly to \( GP'M' \) and \( GP''M' \) for the remaining three programs.

### Table 6. The \( p \)-values from Shapiro-Wilk Normality Test on Observed Expense Values

<table>
<thead>
<tr>
<th>Subject</th>
<th>( ER' )</th>
<th>( GP'M' )</th>
<th>( GP''M' )</th>
<th>( GP'''M' )</th>
<th>( ER_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>flex</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( 0.0406 )</td>
</tr>
<tr>
<td>grep</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
</tr>
<tr>
<td>gzip</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
</tr>
<tr>
<td>sed</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
</tr>
<tr>
<td>space</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
<td>( &lt; 1e-4 )</td>
</tr>
</tbody>
</table>

- \( ER' \) vs. \( GP'M' \): at point Q3 in space.

In remaining cases, although \( ER' \) 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 \( GP'M' \), \( GP''M' \), \( GP'''M' \), and \( ER_5 \) are from 1.2 to 7.5 times larger than that of \( ER' \).

4.2.2 Wilcoxon Signed Rank Test and Effect Sizes. Tables 5 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 \(\sigma\), there are both two-tailed \(p\)-value and one-tailed \(p\)-value which can be used to obtain a conclusion.

Table 7 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: \(ER'_1\) v.s. \(GP_{M2}\), \(ER'_1\) v.s. \(GP_{M3}\), \(ER'_1\) v.s. \(GP_{M19}\), \(ER'_1\) v.s. \(ER_5\), \(GP_{M2}\) v.s. \(GP_{M3}\), \(GP_{M2}\) v.s. \(GP_{M19}\), \(GP_{M3}\) v.s. \(ER_5\), \(GP_{M19}\), \(GP_{M3}\) v.s. \(GP_{M19}\), \(GP_{M3}\) v.s. \(ER_5\), and \(GP_{M19}\) v.s. \(ER_5\). 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 higher probability of Type I errors in multiple comparisons. Both the two-tailed and the one-tailed \(p\)-values are recorded. We set the \(a\) level (after Bonferroni correction) to 0.05.

Table 5 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, 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:

- f1exec: \(ER'_1 \geq (GP_{M3} \geq GP_{M19} \geq GP_{M2}) \geq ER_5\)
- grep: \(ER'_1 \geq (GP_{M3} \geq GP_{M19} \geq GP_{M2}) \geq ER_5\)
- gzip: \(ER'_1 \geq (GP_{M19} \geq GP_{M3}) \geq (ER_5 \geq GP_{M2})\)
- sed: \(ER'_1 \geq (GP_{M19} \geq GP_{M3} \geq GP_{M2}) \geq ER_5\)
- space: \((ER'_1 \geq GP_{M3}) \geq (GP_{M19} \geq GP_{M2}) \geq ER_5\)

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, \(GP_{M3}\) and \(GP_{M19}\) perform roughly the same overall, while \(GP_{M2}\) being the worst among the GP formulæ. \(GP_{M3}\) performs noticeably better than \(GP_{M2}\) and \(GP_{M19}\) 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.
- $GP^M_2$, $GP^M_3$, and $GP^M_{19}$ 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 2, 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 $GP^M_2$, $GP^M_3$, or $GP^M_{19}$. The fact that $ER'_1$ produces generally stable and good performance, while $GP^M_2$, $GP^M_3$, and $GP^M_{19}$ 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 ($GP^M_2$, $GP^M_3$, and $GP^M_{19}$) 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 $GP^M_2$, $GP^M_3$, and $GP^M_{19}$ 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 formulæs 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 formulæ in Definition 2.8 only concerned a subset of formulæ, $S$, out of all possible formulæ, $F$. The subset $S$ 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 $S$ with $\mathcal{F}$. This will, in turn, lead to the investigation of the “greatest” formulæ. 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 Lemma 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 $\sigma(s_i) = \langle e^i_f, e^i_p, n^i_f, n^i_p \rangle$, it holds that $e^i_f + e^i_p > 0 \land e^i_f + n^i_p = P \land e^i_f \leq F \land e^i_p \leq P$.

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

Intuitively, Lemma 5.1 and 5.2 allow us to reason about risk evaluation formulæ spatially in three dimensions. Definition 2.1 involves 5 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 $\sigma(s_i) = \langle e^i_f, P - n^i_p, F - e^i_f, n^i_p \rangle$ after Lemma 5.1, which can be denoted as $\bar{\sigma}(s_i) = \langle e^i_f, e^i_p \rangle$. 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 $\bar{\mathcal{F}} = \{ \bar{R}[\mathcal{R} : I_f \times I_p \rightarrow \text{Real}] \}$, where $I_f$ denotes the set of integers within $[0, F]$ and $I_p$ denotes the set of integers within $[0, P]$, such that $\bar{R}(e^i_f, n^i_p) = R(e^i_f, e^i_p, n^i_f, n^i_p))$. 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 $\bar{R}$ and $\bar{R}$ can and will be used interchangeably, as are symbols $\mathcal{F}$ and $\mathcal{F}$.

Given any values of $P$ and $F$, the input domain of any formulæ $\bar{R}$ is shown as the grid in Figure 2(a), where both $e^i_f$ and $e^i_p$ 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 $e^i_f$ and $e^i_p$ values. Note that the number of statements that associated with each point $(e_f, e_p)$ is independent of the formulæ, but solely decided by the pair of program and test suite.

A formulæ $\bar{R}$ 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 $\bar{R}$.

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 formulæ more precisely. Lemma 5.2 allows us to limit the region of the input domain $\bar{A}$ in which the faulty statement can be.

**Definition 5.3 (Faulty Border).** Let us call the sequential points $< (F, 0), (F, 1), ..., (F, e_p), ..., (F, P) >$ $(0 \leq e_p \leq P)$ the Faulty Border, which is denoted as $E$. Figure 2(b) illustrates a potential $E$.
Human Competitiveness of Genetic Programming in Spectrum Based Fault Localisation

(a) Spectral Coordinate \( \bar{\sigma} \) for SBFL formulæ: because \( e_f + e_p > 0 \) and \( e_f + n_f = F \), the program spectrum of any statement can be simply reduced to \( < e_f, e_p > \), 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 \( R \): 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 statements $s_f$).** 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 $[0, P]$.

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 $e_p^f$, i.e. 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 $(F, e_i^p)$ other than the one associated with $s_f$ on $E$ are associated with correct statements, where $n_i^f = F \land 0 \leq e_i^p \leq P \land e_i^p \neq e_p^f$. Depending on the adopted formula, the risk values of such points can be either greater than, equal to, or smaller than that of point $(F, e_p^f)$, i.e. 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 $(F, e_i^f) = (F, e_p^f)$. 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 $e_i^f = F$, and their $e_i^p$ 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.

![fig:3](image)

*Fig. 3. Example of a faulty border of $R$*
5.2 Maximality in $\mathcal{F}$

First, let us present the definition of the maximality with respect to $\mathcal{F}$, i.e. 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 $R' \neq R \land R' \rightarrow R$, it also holds that $R' \iff R$.

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. $R \iff R'$). However, since a formula from one maximal group, e.g. $ER1'$, does not dominate another formula from another maximal group, e.g., $ER5$, formulæ from different maximal groups do not have to be equivalent to each other.

**Definition 5.7. Ranking.** Given a formula $R$, we use $\rho_p^{i,j} = \langle n_p^i, n_p^j, op \rangle$ 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 $n_p^j < n_p^i$, $op$ can be either “$>$” (i.e., $R(F, n_p^j) > R(F, n_p^i)$), “$<$” (i.e., $R(F, n_p^j) < R(F, n_p^i)$), or “$=$” (i.e., $R(F, n_p^j) = R(F, n_p^i)$).

Let $P_R$ denote the set of $\rho_p^{i,j}$ 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,

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

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 $R'$.

**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 $R \not\iff R'$. 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:

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

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.
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 $(F, e^j_F)$ on $E$, for formula $R'$. By definition, $R'$ assigns identical risk values to points on $E$ as $R$, while ensuring that $U_{R'} = \emptyset$.

**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 $S^R_A$. However, for formula $R$, since $U_R \neq \emptyset$, some statements corresponding to points outside $E$ belong to either $S^R_B$, $S^R_E$, and $S^R_F$.

Summarizing the above two cases, we have $S^R_B \subseteq S^R_B$ and $S^R_A \subseteq S^R_A$. Following Theorem 2.6, $R' \rightarrow R$.

Let us now turn to show that $R \not\rightarrow R'$, 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 $I_r \times I_p - E$, 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 which ranks parts or even all of these statements before $s_f$.

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

In conclusion, if $R$ assigns point $(e^j_F, e^j_P)$ outside $E$ with risk value higher than, or equal to, that of at least one point $(F, n^i_F)$ 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 $\phi^i_{p,F}$ for all pairs of distinct points $(F, e^i_F)$ and $(F, e^i_F)$ on $E$ (where $e^i_F < e^i_F$), for $R_1$ and $R_2$, respectively. Let $U_{R_1}$ and $U_{R_2}$ denote the sets of points outside $E$ which have risk values higher than, or equal to, those of some points $(F, e^i_F)$ on $E$, for formula $R_1$ and $R_2$, respectively.

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

---

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

ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 4, Article 39. Publication date: March 2010.
void foo(double x, double y, double z) {
    if (z <= 0) {
        // s2
    } else {
        if (z <= 12) {
            // s4
        } else {
            // s5
        }
    }
    if (z <= 3) {
        // s7
    } else {
        if (2 * x - y < 0) { // faulty, should be: if (x - y < 0)
            // s9
        } else {
            // s10
        }
    }
    return; // s11
}

Fig. 4. Source Code

z \leq 0
z > 0
0 < z \leq 12
z > 12
0 < z \leq 7
z > 7
z > 7 \land 2x - y < 0
z > 7 \land 2x - y \geq 0

Fig. 5. Control Flow

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

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 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 \( S_{A_1} \) and \( S_{A_2} \).

In summary, we have \( S_{A_1} = S_{A_2} \) and \( S_{A_1} = S_{A_2} \). Following Theorem 2.7, \( R_1 \leftrightarrow R_2 \). \( \Box \)

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 \( P_{R_1} \neq P_{R_2} \), we have \( R_1 \leftrightarrow R_2 \) and \( R_2 \leftrightarrow R_1 \).

**Proof.** Since \( P_{R_1} \neq P_{R_2} \), there must exist at least one pair of points on \( E_1 = (F, e_1^p), (F, e_1^p) \) (where \( e_1^p < e_2^p \)), such that \( e_1^p, e_1^p, op_1 \in P_{R_1} \land e_1^p, e_1^p, op_2 \in P_{R_2} \land op_1 \neq op_2 \). It is sufficient to consider the following two cases because other cases can be transformed to these two cases by swapping \( R_1 \) and \( R_2 \):

ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 4, Article 39. Publication date: March 2010.
Case: $R_1(F, e_1^F) < R_1(F, e_2^F)$ and $R_2(F, e_1^F) > R_2(F, e_2^F)$. With the program shown in Figure 6, it is possible to construct a test suite, such that $e_4^F, e_5^F, e_9^F$ and $e_{10}^F$ are smaller than $F$. (As a reminder, it always holds that $e_2^F = e_7^F = 0$.) While for $s_1, s_3, s_5, s_8 (s_f)$ and $s_{11}$, whose $e_f$ values are all equal to $F$, we have $e_p^F = e_p^F < e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F$. Then, for $R_1$, we have $s_1, s_3, s_5$ 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 $e_4^F$ and $e_f^F$ are both smaller than $F$, but $e_2^F$ is equal to $F$. (Correspondingly, $e_{10}^F = 0$.) For $s_1, s_3, s_5, s_8 (s_f), s_9$ and $s_{11}$, whose $e_f$ values are all equal to $F$, we have $e_p^F = e_p^F < e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F$. Then, for $R_1, s_1, s_3, s_5, s_f$, and $s_{11}$ are tied together at the top of the whole list before $s_f$. However, for $R_2, s_f$ is ranked at the top, immediately followed by $s_1, s_3, s_5, s_f$, 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 $R_1(F, e_1^F) < R_1(F, e_2^F)$ while $R_2(F, e_1^F) > R_2(F, e_2^F)$, it is always possible to find examples to demonstrate $R_1 \rightarrow R_2$ and $R_2 \rightarrow R_1$.

Case: $R_1(F, e_1^F) < R_1(F, e_2^F)$ and $R_2(F, e_1^F) = R_2(F, e_2^F)$. With the program shown in Figure 6, it is possible to construct a test suite, such that $e_1^F = F$ (correspondingly, $e_2 = 0$), while $e_9^F$ and $e_{10}^F$ are smaller than $F$. Then, for $s_1, s_3, s_4, s_5, s_8 (s_f)$, and $s_{11}$, whose $e_f$ values are all equal to $F$, it follows that $e_p^F = e_p^F < e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F$. Then, for $R_1, s_1, s_3, s_5, s_f, s_{11}$ are tied together at the top of the entire ranking, before $s_4$. However, for $R_2, s_1, s_3, s_4, s_f, s_{11}$ 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 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 $e_4^F, e_5^F, e_9^F$ and $e_{10}^F$ are smaller than $F$. Then, for $s_1, s_3, s_5, s_8 (s_f)$ and $s_{11}$ whose $e_f$ values are all equal to $F$, we have $e_p^F = e_p^F < e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F = e_p^F$. (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 $s_1, s_3, s_5$ 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 $s_1, s_3, s_5, s_f$, 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 $R_1(F, e_1^F) < R_1(F, e_2^F)$ while $R_2(F, e_1^F) = R_2(F, e_2^F)$, it is possible to demonstrate that $R_1 \rightarrow R_2$ and $R_2 \rightarrow R_1$.

In conclusion, for any two formulæ whose $U_{R_1}$ and $U_{R_2}$ are both $0$, but $P_{R_1} \neq P_{R_2}$, it follows that $R_1 \rightarrow R_2$ and $R_2 \rightarrow R_1$. \hfill \Box

---

9This 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].

10For the feasibility of this test suite, please refer to Test Suite A in Example 5 of the Appendix.

11For the feasibility of this test suite, please refer to Test Suite B in Example 5 of the Appendix.

12For the feasibility of this test suite, please refer to Test Suite C in Example 5 of the Appendix.
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 $E$ (Lemma 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 $e_{p}^{i,j}$ 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 $E$ which 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:** $U_{R'} \neq \emptyset$. Similar to the proof of Lemma 5.8, it is always possible to construct another formula $R''$, such that $U_{R''} = \emptyset$, $R'' \rightarrow R'$ and $R' \not\rightarrow R''$. If $P_{R''} = P_{R}$, after Lemma 5.9, $R'' \leftrightarrow R$, and, consequently, $R' \not\rightarrow R$. Otherwise, if $P_{R''} \neq P_{R}$, after Lemma 5.10, $R' \not\rightarrow R''$ and $R'' \not\rightarrow R$. As a consequence, $R' \not\rightarrow R$.

**Case:** $U_{R'} = \emptyset$. Following the definition of $R'$, after Lemma 5.10, $R' \not\rightarrow R$. Otherwise, if $P_{R'} \neq P_{R}$, after Lemma 5.10, $R' \not\rightarrow R'$.

In summary, if $U_R = \emptyset$, then for any formula $R'$, we have either $R' \not\rightarrow R$ or $R' \not\rightarrow R$. 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 $R$ which 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 $E$.

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 $ER_{3}$) 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 $ER_{3}$ are maximal elements of $\mathcal{F}$.

**Proof.** For any formula $R$ in $ER'_{1}$ or $ER_{3}$, $U_R = \emptyset$. After Proposition 5.11, formulæ in $ER'_{1}$ and $ER_{3}$ 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: $F = 2$, $P = 9$, $s_{f}$ satisfies $e_{f}^{2} = 2 = F \land e_{f}^{4} = 5 < P$, and $s_{f}$ satisfies $e_{f}^{2} = 1 < F \land e_{f}^{9} = 4 < e_{f}^{P}$. Then, following the definition of GP02 (which is $2(e_{f} + \sqrt{n_{P}^{1}}) + \sqrt{n_{P}^{2}}$), the following risk evaluation values are obtained: $GP02(s_{f}) = 13$, which is smaller than $GP02(s_{f}) = 14$. 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 $\mathcal{F}$.

**Case:** for GP03. Construct a test suite that satisfies the following: $F = 2$, $P = 30$, $s_{f}$ satisfies $e_{f}^{2} = 2 = F \land e_{f}^{4} = 25 < P$, and $s_{f}$ satisfies $e_{f}^{2} = 1 < F \land e_{f}^{9} = 25 = e_{f}^{P}$. Then, following the definition

---

13This is how GP02, GP03, and GP19 have been treated for the empirical study in Section 4.
14The feasibility of this scenario is analysed in Example 2 of the Appendix.
of GP03 (which is $\sqrt{|e_f^2 - \sqrt{|e_f^2}}|$), 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 of the point on $E$. Following Proposition 5.11, GP03 is not the maximal to all formulæ in $F$.\textsuperscript{15}

**Case:** for GP19. Construct a test suite that satisfies the following: $F = 5, P = 50, s_f$ satisfies $e_f^f = 5 = F \land e_f^p = 1 < P$, and $s_4$ satisfies $e_f^f = 4 < F \land e_f^p < e_f^p = 49 < P$. Then, following the definition of GP19 (which is $e_f \sqrt{|e_p - e_f + n_f - np|}$), 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 $F$.\textsuperscript{16}

With Proposition 5.11, it becomes possible to identify formulæ that are maximal elements of $F$. 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 $F$: The Non-Existence Proof

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

**Definition 5.14. Greatest Formula.** A risk evaluation formula $R$ is said to be a greatest formula in $F$ if, for any formula $R' \in F \land R' \neq R$, 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 that there can exist points outside $E$ with risk values higher than that of the point on $E$. Following Proposition 5.11, it becomes possible to identify formulæ that are maximal elements of $F$. Let us now turn to the greatest formula, or, in fact, proving the lack of thereof. We have shown that there can exist points outside $E$ with risk values higher than that of the point on $E$. Following Proposition 5.11, it becomes possible to identify formulæ that are maximal elements of $F$.

**Proposition 5.15.** There is no formula which is greatest against the set of all formulæ, $F$.

**Proof.** Assume that there exists a greatest formula $R_g$. Let $P_{R_g}$ denote the set of $o^{i,j}_p$ for all pairs of distinct points $(F, e^i_p)$ and $(F, e^j_p)$ on $E$ (where $e^i_p < e^j_p$) and $U_{R_g}$ denote the sets of points outside $E$ which have risk values higher than or equal to those of some points $(F, e^i_p)$ on $E$, for formulæ in $F$. After Prop. 5.11, $U_{R_g} = \emptyset$ and $R_g$ is a maximal element of $F$.

Consider the two maximal groups of formulæ $ER'_1$ and $ER_5$, which have been proved to be non-equivalent to each other\textsuperscript{52}. Let $U_{ER'_1}$ and $U_{ER_5}$ denote the sets of points outside the faulty border which have risk values higher than or equal to those of some points $(F, e^i_p)$ 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 $o^{i,j}_p$ for all pairs of distinct points $(F, e^i_p)$ and $(F, e^j_p)$ on the faulty border (where $e^i_p < e^j_p$), 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:

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

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

**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 $ER_5$, $U_{ER'_1} = U_{R_g} = \emptyset$ and $P_{ER'_1} \neq R_g$.

\textsuperscript{15}The feasibility of this scenario is analysed in Example 3 of the Appendix.

\textsuperscript{16}The feasibility of this scenario is analysed in Example 4 of the Appendix.
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 $R' \not\rightarrow R_g$ and $R_g \not\rightarrow R'$. 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 formulae, 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 [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 Abreu et al. [1] is the maximal version of $q_e$ [52], which is also equivalent to $ER_i^1$.

6 CONCLUSIONS

Spectrum Based Fault Localisation (SBFL) has received 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 focus of the research mainly 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 paper 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, i.e. 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.

REFERENCES


ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 4, Article 39. Publication date: March 2010.


Human Competitiveness of Genetic Programming in Spectrum Based Fault Localisation


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 “\( \text{if}(x-y<0) \)”, not “\( \text{if}(2x-y<0) \)” and, therefore, is denoted by \( s_f \). Figure 7 shows the corresponding regions of failure inducing inputs in the space of all possible inputs.

![Sample program](image)

Fig. 7. Sample program

First, consider statements \( s_9 \) and \( s_{10} \):

- Any test case \( t_i = (x_i, y_i, z_i) \) such that \( z_i > 7 \land (x_i, y_i) \in \text{Fail}_9 \) 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 \( z_i > 7 \land (x_i, y_i) \in \text{Pass}_9 \) 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 \( z_i > 7 \land (x_i, y_i) \in \text{Fail}_{10} \) 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 \( z_i > 7 \land (x_i, y_i) \in \text{Pass}_{10} \) 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 is executed by any test case. Consequently, \( e_f^9 = n_f^9 + n_p^9 = e_f^{10} + e_p^{10} \), and \( n_p^9 = e_p^{10} \).

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 \( e_p^7 \).

ACM Transactions on Software Engineering and Methodology, Vol. 9, No. 4, Article 39. Publication date: March 2010.
Now, consider statements \( s_6, s_11 \) and \( s_3 \), whose \( e_p \) values are identical to each other, and so are their \( e_f \). It can be seen from Figure 6 that \( e_f^6 = e_f^{11} = e_f^3 = e_f^4 = F \) and \( e_f^6 = e_f^3 = e_f^{11} = (e_f^6 + e_f^7) \). As a reminder, according to the above analysis, \( e_f^p \) and \( e_f^p \) 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 \( e_f^p \).
- Any test case \( t_i =< x_i, y_i, z_i > \) such that \( 7 < z_i \leq 12 \) while \( < x_i, y_i > \in \text{Pass}_9 \cup \text{Pass}_{10} \) will also pass. Let us denote the number of these test cases as \( e_f^p \).
- Any test case \( t_i =< x_i, y_i, z_i > \) such that \( 7 < z_i \leq 12 \) while \( < x_i, y_i > \in \text{Fail}_9 \cup \text{Fail}_{10} \) 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, \( e_f^6 = e_f^3 + e_f^7 \), where \( e_f^3 \) is the size of the subset of passing test cases that cover and satisfy \( 7 < z_i \leq 12 \). Thus, it also follows that \( e_f^3 \leq e_f^6 \). As a reminder, the values of \( e_f^6 \) and \( e_f^7 \) can be decided independently. Therefore, \( e_f^6 \) can be either smaller than, equal to or greater than \( e_f^4 \).

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 \( < x_i, y_i > \in \text{Pass}_9 \cup \text{Pass}_{10} \) will always pass. The number of these test cases is \( e_f^p \).
- Any test case \( t_i =< x_i, y_i, z_i > \) such that \( z_i > 12 \) while \( < x_i, y_i > \in \text{Fail}_9 \cup \text{Fail}_{10} \) will always fail. The number of these test cases is \( e_f^4 \).

Note that \( e_f^5 \) 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 \( e_f^4 \leq e_f^5 = F \) and \( e_f^5 \leq e_f^6 \).

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

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: \( e_f^p \) 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 \( e_f^2 \).

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

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 \( e_f^p \) and \( e_f^p \) are any values, and \( e_f^p < F \). According to the above discussion, for \( s_f \), we can assign any value to \( e_f^p \); while for \( s_4 \), \( e_f^4 \) can be any value within \([0, F]\) and \( e_f^4 \)
can be either smaller than, equal to or greater than $e^f_p$. 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 $F = 2$, $P = 9$, $s_f$ has $e^f_f = 2 = F$ and $e^f_p = 5 < P$, and $s_9$ has $e^0_9 = 1 < F$ and $e^9_9 = 4 < e^f_f$. According to the above analysis, we can assign any value to $e^f_p$ and any value to $P$ that is no less than $e^f_f$. And for $s_9$, we can have any value of $e^0_9$ within $[0, F]$ and any value of $e^9_9$ within $[0, e^f_p]$. 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 $e^f_p = 25 < P$, and $s_9$ has $e^0_9 = 1 < F$ and $e^9_9 = 25 = e^f_f$. 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 $e^f_p = 1 < P$, and $s_4$ has $e^4_f = 4 < F$ and $e^p_f < e^4_f = 49 < P$. According to the above analysis, for $s_f$, we can assign any values to $e^f_f$ (i.e. $F$) and $e^f_p$; while for $s_4$, $e^4_f$ can be any value within $[0, F]$ and $e^p_f$ can be either smaller than, equal to or greater than $e^f_f$; and $P$ can be any value that is no less than either $e^p_f$ or $e^f_f$. 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 $e^H_p$ and $e^H_p$, where $e^H_p < e^H_p$. 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 $e^1_f$, $e^3_f$, $e^0_f$ and $e^{10}_f$ smaller than $F$, $e^1_f = e^3_f = e^6_f = e^f_f = e^1_1 = F$, $e^p_p = e^p_f$ and $e^3_p = e^6_p = e^{11}_p = e^H_p$. According to the above analysis, $e^f_f$, $e^f_f$, $e^f_f$ and $e^{10}_f$ 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 $e^f_p$; while it is always possible to have any equal value of $e_p$ that is larger than $e^f_f$ for $s_1$, $s_3$, $s_6$ and $s_{11}$. Therefore, this test suite is feasible.

**Test Suite B:** we have both $e^4_f$ and $e^5_f$ smaller than $F$, $e^4_f = e^5_f = e^6_f = e^f_f = e^{11}_f = F$, $e^p_p = e^p_f$ and $e^1_p = e^3_p = e^6_p = e^1_p = e^{11}_p = e^H_p$. As discussed above, it is always possible to have both $e^4_f$ and $e^{10}_f$ 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 $e^2_f$ and $e^{10}_f$ smaller than $F$, $e^2_f = e^5_f = e^6_f = e^f_f = e^{11}_f = F$, $e^p_p = e^p_f$ and $e^1_p = e^3_p = e^6_p = e^1_p = e^{11}_p = e^H_p$. As discussed above, it is always possible to have both $e^2_f$ and $e^{10}_f$ 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.

Received February 2007; revised March 2009; accepted June 2009