Provably Optimal and Human-Competitive Results in SBSE for Spectrum Based Fault Localisation

Xiaoyuan Xie[1], Fei-Ching Kuo[1], Tsong Yueh Chen[1], Shin Yoo[2], and Mark Harman[2]

[1] Swinburn University, John St, Hawthorn VIC3122, Australia
{xxie,dkuo,tychen}@swin.edu.au
[2] University College London, Gower Street, London WC1E 6BT, UK
{shin.yoo,mark.harman}@ucl.ac.uk

Abstract. Fault localisation uses so-called risk evaluation formulæ to guide the localisation process. For more than a decade, the design and improvement of these formulæ has been conducted entirely manually through iterative publication in the fault localisation literature. However, recently we demonstrated that SBSE could be used to automatically design such formulæ by recasting this as a problem for Genetic Programming(GP). In this paper we prove that our GP has produced four previously unknown globally optimal formulæ. Though other human competitive results have previously been reported in the SBSE literature, this is the first SBSE result, in any application domain, for which human competitiveness has been formally proved. We also show that some of these formulæ exhibit counter-intuitive characteristics, making them less likely to have been found solely by further human effort.

1 Introduction

Early work demonstrated the wide applicability of SBSE to many different software engineering domains, perhaps surprising some software engineers, who had previously thought computational search inadmissible in their areas of activity. However, now that SBSE is a mature [7] and well-established 'standard' approach to software engineering [9, 11], the SBSE research agenda should become more ambitious in order to continue to stimulate further development.

One area in which more work is needed lies in the development of techniques that are human competitive, a long-sought goal of all optimisation approaches. Such results are inherently compelling demonstrations of the value of SBSE for which the scientific evidence should be sufficient to convince even the most skeptical software engineer.

Recent work has produced specific claims for human competitive results in SBSE [19], while much other SBSE work is already implicitly partly human competitive, since it automates aspects of software engineering for which human effort is simply too expensive [11, 15, 17]. In this paper we seek to go a step further. We seek not only to demonstrate that our SBSE results are human competitive, but also that we have provably optimal results in an area for which many years of human effort have been expended by very capable scientists to construct just such optimal results.

The area for which we are able to demonstrate provably optimal and human competitive results is fault localisation. We focus on Spectrum-Based Fault Localisation (SBFL), a well-known and widely-studied fault localisation approach. SBFL ranks statements according to a risk evaluation formula. The faulty statement should ideally be ranked at the top. Designing an effective risk evaluation formula has been one of the most widely studied aspects of SBFL: known formulæ include Tarantula [14], Ochiai [1], Wong [20] and many others.

There has been more than a decade of risk evaluation formulæ development, all of which has remained entirely manual. This development has called upon the considerable ingenuity of many different groups of researchers, all of which have peer-reviewed expertise and results on the introduction of each of their proposed formulæ. Therefore, any approach which could automatically find an equivalent or better performing formula would clearly be human competitive, and at the highest level of intellectual challenge too.

Recently, Genetic Programming (GP) has been successfully applied to automatic design of risk evaluation formulæ [23]. Empirical results showed that, among the 30 GP-evolved formulæ, six are very effective and can outperform some human-designed formulæ. However, this analysis was entirely empirical; we cannot be *sure* that the evaluation formulæ found by our GP approach are always superior.

Fortunately, Xie et al. developed a framework to support the theoretical analysis of risk evaluation formulæ performance [21,22]. Xie et al. analysed 30 manually designed risk evaluation formulæ, identifying a fault localisation effectiveness hierarchy between formulæ. The results of the theoretical analysis showed that there exist two maximal groups of human defined formulæ, namely ER1 and ER5, for programs with single fault.

In this paper, we apply the same theoretical framework to the 30 GP-evolved formulæ discovered by GP and reported by Yoo at SSBSE 2012 [23]. The results show that, among these 30 GP-evolved formulæ, four formulæ, namely GP02, GP03, GP13, and GP19 are optimal: GP13 is proved to be equivalent to the human-discovered optima ER1, while the remaining three formulæ form three distinct and entirely new groups of optima.

Interestingly, some of the optimal GP-evolved formulæ display characteristics that are best described as 'unintuitive'. This is a common observation for computational search; it finds niche results that are not always obvious and sometimes highly counter-intuitive; SBSE is no exception [11]. Since our results are both optimal, yet counter-intuitive, they are not only human competitive with respect to the past decade of human effort, but also unlikely to have been discovered by further decade of human effort.

The contributions of this paper are as follows:

– We prove that one of the risk-evaluation formulæ from the previous work [23] belongs to the same equivalence group as two known maximal formulæ,

extending the maximal group ER1 [21] to ER1'. This shows provable human competitiveness for the first time in SBSE.
- We also prove that three other formulæ from the previous work [23] form their own maximal groups.
- Our analysis of the evolved formulæ shows the flexibility of GP in designing risk evaluation formulæ. For some formulæ, GP follows the same design intuition as humans; for others, GP does not conform to the human intuition but still produces maximal formulæ.

The rest of the paper is organised as follows. Section 2 describes the foundations of Spectrum-Based Fault Localisation (SBFL) and the theoretical framework that uses set-membership to provably compare risk evaluation formulæ. Section 3 contains proofs of maximality for GP02, GP03, GP13, and GP19. Section 4 discusses the insights gained from an in-depth analysis of GP-evolved formulæ. Section 5 presents related work and Section 6 concludes.

2 Background

2.1 Spectrum-Based Fault Localisation (SBFL)

SBFL uses testing results and program spectrum to do fault localisation. The testing result is whether a test case is *failed* or *passed*. While the program spectrum records the run-time profiles about various program entities for a specific test suite. The program entities could be statements, branches, paths, etc.; and the run-time information could be the binary coverage status, the execution frequency, etc. The most widely used program spectrum involves statement and its binary coverage status in a test execution [2, 14].

$$\begin{array}{cccc} & & \text{TS:} & \begin{pmatrix} t_1\ t_2\ \ldots\ t_m \end{pmatrix} \\ \text{PG:} & \begin{pmatrix} s_1 \\ s_2 \\ \cdot \\ \cdot \\ \cdot \\ s_n \end{pmatrix} & \text{MS:} & \begin{pmatrix} 1/0\ 1/0\ \ldots\ 1/0 \\ 1/0\ 1/0\ \ldots\ 1/0 \\ \ldots \\ \ldots \\ \ldots \\ 1/0\ 1/0\ \ldots\ 1/0 \end{pmatrix} \\ & & \text{RE:} & \begin{pmatrix} p/f\ p/f\ \ldots\ p/f \end{pmatrix} \end{array}$$

Fig. 1. Information for conventional SBFL

Consider a program $PG = <s_1, s_2, ..., s_n>$ with n statements and a test suite of m test cases $TS = \{t_1, t_2, ..., t_m\}$. Figure 1 shows the information required by SBFL. *RE* records all the testing results, in which p and f indicate *passed* and *failed*, respectively. Matrix *MS* represents the program spectrum, where the (i^{th}, j^{th}) element represents the coverage information of statement s_i, by test case t_j, with 1 indicating s_i is executed, and 0 otherwise. In fact, the j^{th} column represents the *execution slice* of t_j.

For each statement s_i, its relevant testing result can be represented as a tuple i=($e_f^i, e_p^i, n_f^i, n_p^i$), where e_f^i and e_p^i represent the number of test cases in TS that execute it and return the testing result of *failure* or *pass*, respectively; n_f^i and n_p^i denote the number of test cases that do not execute it, and return the testing result of *failure* or *pass*, respectively. A risk evaluation formula R is then applied to the tuple corresponding to each statement s_i to calculate the *suspiciousness* score that indicates its risk of being faulty. Ideally, the faulty statement should be at or near the top of the ranking, so that the developer can save time if the program statements are examined following the ranking order.

The most commonly adopted intuition in designing risk evaluation formulæ is that statements associated with more *failed* or less *passed* testing results should not have lower risks. Formulæ that comply with this intuition include Tarantula [12], Jaccard [4], Ochiai [1], Naish1 and Naish2 [16], among others.

2.2 Theoretical Framework

With the development of more and more risk evaluation formulæ, people began to investigate their performance. Xie et al. [21] have recently developed a theoretical framework to analysis the performance between different formulæ. Since we will apply this theoretical framework in this paper, thus we briefly describe it before presenting the analysis on GP-evolved formulæ.

Definition 1. *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 vector i=<$e_f^i, e_p^i, n_f^i, n_p^i$> can be constructed from TS, and $R(s_i)$ is a function of i. For any faulty statement s_f, following three subsets are defined.*

$$S_B^R = \{s_i \in S | R(s_i) > R(s_f), 1 \leq i \leq n\}$$
$$S_F^R = \{s_i \in S | R(s_i) = R(s_f), 1 \leq i \leq n\}$$
$$S_A^R = \{s_i \in S | R(s_i) < R(s_f), 1 \leq i \leq n\}$$

That is, S_B^R, S_F^R and S_A^R consist of statements of which the risk values are higher than, equal to and lower than the risk value of s_f, respectively.

In practice, a tie-breaking scheme may be required to determine the order of the statements with same risk values. The theoretical analysis only investigates consistent tie-breaking schemes, which are defined as follows.

Definition 2. *Given any two sets of statements S_1 and S_2, which contain elements having the same risk values. 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 O_1 and O_2.*

The effectiveness measurement is referred to as Expense metric, which is the percentage of code that needs to be examined before the faulty statement is identified [23]. A lower Expense of formula R indicates a better performance.

Let E_1 and E_2 denote the Expenses with respect to the same faulty statement for risk evaluation formulæ R_1 and R_2, respectively. We define two types of relations between R_1 and R_2 as follows.

Definition 3 (Better). *R_1 is said to be* better *than R_2 (denoted as $R_1 \rightarrow R_2$) if for any program, faulty statement s_f, test suite and consistent tie-breaking scheme, we have $E_1 \leq E_2$.*

Definition 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, we have $E_1 = E_2$.*

It is obvious from the definition that $R_1 \rightarrow R_2$ means R_1 is equal to or more effective than R_2. As a reminder, 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. In the theoretical framework, there are several assumptions, which are listed as follows.

1. A testing oracle exists, that is, for any test case, the testing result of either *fail* or *pass* can be decided.
2. We have the assumption of perfect bug detection that the fault can always be identified once the faulty statement is examined.
3. We exclude omission faults, because SBFL is designed to assign risk values to the existent statements.
4. We assume that the test suite contains at least one passing test case and one failing test case.

As a reminder, our analysis only focuses on statements that are covered by the given test suite (that is, any statement s_i such that $e_p^i + e_f^i > 0$). This is because a statement that is never covered by any test case in the given test suite cannot be the faulty statement that triggers the observed failure and hence should be ignored (or effectively deemed to have the lowest risk values). For readers who are interested in all the detailed justifications, validity and impacts of the above assumptions, please refer to [21].

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. Immediately after the definitions and the above assumptions, we have $1 \leq F < T$, $1 \leq P < T$, and $P + F = T$, as well as the following lemmas.

Lemma 1. *For any $i = <e_f^i, e_p^i, n_f^i, n_p^i>$, we have $e_f^i + e_p^i > 0$, $e_f^i + n_f^i = F$, $e_p^i + n_p^i = P$, $e_f^i \leq F$ and $e_p^i \leq P$.*

Lemma 2. *For any faulty statement s_f with $f = <e_f^f, e_p^f, n_f^f, n_p^f>$, if s_f is the only faulty statement in the program, we have $e_f^f = F$ and $n_f^f = 0$.*

A sufficient condition for the equivalence between two risk evaluation formulae is as follows.

Theorem 1. *Let R_1 and R_2 be two risk evaluation formulæ. If we have $S_B^{R_1}=S_B^{R_2}$, $S_F^{R_1}=S_F^{R_2}$ and $S_A^{R_1}=S_A^{R_2}$ for any program, faulty statement s_f and test suite, then $R_1 \leftrightarrow R_2$.*

Xie et al. [21] have applied the above theoretical framework on 30 manually designed formulæ, identifying two groups of most effective formulæ for programs with single fault, namely the maximal groups of formulæ. The definition of *maximal formula* is as follows.

Definition 5. *A risk evaluation formula R_1 is said to be a maximal formula of a set of formulæ, if for any element R_2 of this set of formulæ, $R_2 \rightarrow R_1$ implies $R_2 \leftrightarrow R_1$.*

3 Theoretical Analysis of GP-Evolved Risk Evaluation Formulæ

3.1 Risk Evaluation Formulæ Generated by GP

Yoo [23] has generated 30 GP-evolved formulæ. There are 10 out of the 30 formulæ which need unreasonable additional assumptions, and, hence, are excluded in this study[1]. Therefore, our investigation will focus on the remaining 20 formulæ (namely, GP01, GP02, GP03, GP06, GP08, GP11, GP12, GP13, GP14, GP15, GP16, GP18, GP19, GP20, GP21, GP22, GP24, GP26, GP28 and GP30). As a reminder, the following analysis is for programs with single fault.

The above mentioned theoretical framework has proved the equivalence of the formulae within ER1 (consists of Naish1 and Naish2) and ER5 (consists of Wong1, Russel & Rao, and Binary), as well as their maximality, for programs with single fault [22]. By using the theoretical framework above, we are able to prove that among the 20 GP-evolved formulæ, GP02, GP03, GP13 and GP19 are maximal formulæ for programs with single fault. More specifically, GP02, GP03 and GP19 are distinct maximal formulæ to ER1 and ER5; while GP13 is equivalent to ER1. In the following discussion, the group which consists of Naish1, Naish2 and GP13 will be referred to as ER1'. We have also proved that ER1' is strictly better than all the other remaining 16 GP-evolved formulæ under investigation. However, since the focus of this paper is to identify the maximal (that is, maximally effective) GP-evolved formulæ, we will only provide the detailed proofs for the maximality of GP02, GP03, GP13 and GP19. Definitions of the involved formulæ are listed in Table 1.

3.2 Maximal GP-Evolved Risk Evaluation Formulæ

Before presenting our proof, we need the following lemmas for ER1 (consists of Naish1 and Naish2) and GP13.

[1] The reason for exclusion is primarily to avoid division by zero. For example, GP04 [23] contains $\frac{1}{e_p - n_p}$, i.e., it assumes $e_p \neq n_p$. We consider assumptions of this kind unrealistic.

Table 1. Investigated formulæ

Name		Formula expression
ER1'	Naish1	$\begin{cases} -1 & \text{if } e_f{<}F \\ P - e_p & \text{if } e_f{=}F \end{cases}$
	Naish2	$e_f - \frac{e_p}{e_p+n_p+1}$
	GP13	$e_f(1 + \frac{1}{2e_p+e_f})$
ER5	Wong1	e_f
	Russel & Rao	$\frac{e_f}{e_f+n_f+e_p+n_p}$
	Binary	$\begin{cases} 0 \text{ if } e_f{<}F \\ 1 \text{ if } e_f{=}F \end{cases}$
GP02		$2(e_f + \sqrt{n_p}) + \sqrt{e_p}$
GP03		$\sqrt{\lvert e_f^2 - \sqrt{e_p}\rvert}$
GP19		$e_f\sqrt{\lvert e_p - e_f + n_f - n_p\rvert}$

Lemma 3. *For Naish1 and Naish2, which are shown to be equivalent to each other in the previous work [22], we have* $S_B^{N1}{=}S_B^{N2}{=}X^{Op}$, $S_F^{N1}{=}S_F^{N2}{=}Y^{Op}$ *and* $S_A^{N1}{=}S_A^{N2}{=}Z^{Op}$*, where*

$$X^{Op}{=}\{s_i|e_f^i{=}F \text{ and } e_p^f{>}e_p^i, 1{\le}i{\le}n\} \tag{1}$$

$$Y^{Op}{=}\{s_i|e_f^i{=}F \text{ and } e_p^f{=}e_p^i, 1{\le}i{\le}n\} \tag{2}$$

$$Z^{Op}{=}S\backslash X^{Op}\backslash Y^{Op} \tag{3}$$

Lemma 4. *For GP13, we have* $S_B^{GP13}{=}X^{Op}$, $S_F^{GP13}{=}Y^{Op}$ *and* $S_A^{GP13}{=}Z^{Op}$*, respectively.*

Proof. Since $e_f^f{=}F$, it follows immediately from the definition of GP13 that

$$S_B^{GP13}{=}\{s_i|e_f^i(1{+}\frac{1}{2e_p^i + e_f^i}){>}F(1{+}\frac{1}{2e_p^f{+}F}), 1{\le}i{\le}n\} \tag{4}$$

$$S_F^{GP13}{=}\{s_i|e_f^i(1{+}\frac{1}{2e_p^i + e_f^i}){=}F(1{+}\frac{1}{2e_p^f{+}F}), 1{\le}i{\le}n\} \tag{5}$$

1. To prove that $S_B^{GP13} = X^{Op}$.
   (a) To prove $X^{Op}{\subseteq}S_B^{GP13}$.
   For any $s_i{\in}X^{Op}$, we have $F(1{+}\frac{1}{2e_p^i{+}F}){>}F(1{+}\frac{1}{2e_p^f{+}F})$ because $e_p^f{>}e_p^i$ and $F{>}0$. Since $e_f^i{=}F$, we have $e_f^i(1{+}\frac{1}{2e_p^i{+}e_f^i}){>}F(1{+}\frac{1}{2e_p^f{+}F})$, which implies $s_i{\in}S_B^{GP13}$. Thus, we have proved $X^{Op}{\subseteq}S_B^{GP13}$.
   (b) To prove $S_B^{GP13}{\subseteq}X^{Op}$.
   For any $s_i{\in}S_B^{GP13}$, we have $e_f^i(1{+}\frac{1}{2e_p^i{+}e_f^i}){>}F(1{+}\frac{1}{2e_p^f{+}F})$. Let us consider the following two exhaustive cases.

– Case (i) $e^i_f{<}F$. First, consider the sub-case that $e^i_f{=}0$. Then we have $e^i_f(1{+}\frac{1}{2e^i_p+e^i_f}){=}0$. It follows from the definition of S^{GP13}_B that $0{>}F(1{+}\frac{1}{2e^f_p+F})$, which is however contradictory to $F{>}0$ and $e^f_p{\geq}0$. Thus, it is impossible to have $e^i_f{=}0$. Now, consider the sub-case that $0{<}e^i_f{<}F$. After re-arranging the terms, the expression $e^i_f(1{+}\frac{1}{2e^i_p+e^i_f}) - F(1{+}\frac{1}{2e^f_p+F})$ becomes $(\frac{e^i_f}{2e^i_p+e^i_f} - \frac{F}{2e^f_p+F}) - (F{-}e^i_f)$. Since $0{<}e^i_f{<}F$, this expression can be further re-written as $(\frac{1}{1+2\frac{e^i_p}{e^i_f}} - \frac{1}{1+2\frac{e^f_p}{F}}){-}(F{-}e^i_f)$. Since $\frac{e^i_p}{e^i_f}{\geq}0$ and $\frac{e^f_p}{F}{\geq}0$, we have $0{<}\frac{1}{1+2\frac{e^i_p}{e^i_f}}{\leq}1$ and $0{<}\frac{1}{1+2\frac{e^f_p}{F}}{\leq}1$. As a consequence, we have $(\frac{1}{1+2\frac{e^i_p}{e^i_f}}{-}\frac{1}{1+2\frac{e^f_p}{F}}){<}1$. Since both F and e^i_f are positive and non-negative integers, respectively, $e^i_f{<}F$ implies $(F{-}e^i_f){\geq}1$. Thus, we have $(\frac{1}{1+2\frac{e^i_p}{e^i_f}}{-}\frac{1}{1+2\frac{e^f_p}{F}}){-}(F{-}e^i_f)$ <0, which however is contradictory to $e^i_f(1{+}\frac{1}{2e^i_p+e^i_f}){>}F(1{+}\frac{1}{2e^f_p+F})$. Therefore, it is impossible to have $0{<}e^i_f{<}F$. Therefore, we have proved that if $s_i{\in}S^{GP13}_B$, we cannot have $e^i_f{<}F$.

– Case (ii) $e^i_f{=}F$. Assume further $e^i_p{\geq}e^f_p$. Obviously, we have $F(1 + \frac{1}{2e^i_p+F}){\leq}F(1{+}\frac{1}{2e^f_p+F})$, which can be re-written as $e^i_f(1{+}\frac{1}{2e^i_p+e^i_f}) \leq F(1{+}\frac{1}{2e^f_p+F})$. However, this is contradictory to $F(1{+}\frac{1}{2e^i_p+F}) > F(1{+}\frac{1}{2e^f_p+F})$. Thus, the only possible case is $e^f_p{>}e^i_p$.

Therefore, we have proved that if $s_i{\in}S^{GP13}_B$, then $e^i_f{=}F$ and $e^f_p{>}e^i_p$, which imply $s_i{\in}X^{Op}$. Therefore, $S^{GP13}_B{\subseteq}X^{Op}$.

In conclusion, we have proved $X^{Op}{\subseteq}S^{GP13}_B$ and $S^{GP13}_B{\subseteq}X^{Op}$. Therefore, $S^{GP13}_B = X^{Op}$.

2. To prove that $S^{GP13}_F = Y^{Op}$.

(a) To prove $Y^{Op}{\subseteq}S^{GP13}_F$.
For any $s_i{\in}Y^{Op}$, we have $e^i_f(1{+}\frac{1}{2e^i_p+e^i_f}){=}F(1{+}\frac{1}{2e^f_p+F})$ because $e^i_f{=}F$ and $e^f_p{=}e^i_p$. After the definition of S^{GP13}_F, $s_i{\in}S^{GP13}_F$. Thus, we have proved $Y^{Op}{\subseteq}S^{GP13}_F$.

(b) To prove $S^{GP13}_F{\subseteq}Y^{Op}$.
For any $s_i{\in}S^{GP13}_F$, we have $e^i_f(1{+}\frac{1}{2e^i_p+e^i_f}){=}F(1{+}\frac{1}{2e^f_p+F})$. Let us consider the following two exhaustive cases.

– Case (i) $e^i_f{<}F$. First, consider the sub-case that $e^i_f{=}0$. Then we have $e^i_f(1{+}\frac{1}{2e^i_p+e^i_f}){=}0$. It follows from the definition of S^{GP13}_F that $0{=}F(1{+}\frac{1}{2e^f_p+F})$, which is however contradictory to $F{>}0$ and $e^f_p{\geq}0$. Thus, it is impossible to have $e^i_f{=}0$. Now, consider the sub-case that $0{<}e^i_f{<}F$. Similar to the above proof of $S^{GP13}_B{\subseteq}X^{Op}$, we can prove that $(\frac{1}{1+2\frac{e_p^i}{e_f^i}}-\frac{1}{1+2\frac{e_p^f}{F}})<(F-e_f^i)$, which is however contradictory to $e_f^i(1+\frac{1}{2e_p^i+e_f^i})=F(1+\frac{1}{2e_p^f+F})$. Therefore, it is impossible to have $0<e_f^i<F$. Therefore, we have proved that if $s_i \in S_F^{GP13}$, then we cannot have $e_f^i<F$.

– Case (ii) $e_f^i=F$. Assume further $e_p^i \neq e_p^f$. Obviously, we have $F(1+\frac{1}{2e_p^i+F}) \neq F(1+\frac{1}{2e_p^f+F})$, which can be re-written as $e_f^i(1+\frac{1}{2e_p^i+e_f^i}) \neq F(1+\frac{1}{2e_p^f+F})$. However, this is contradictory to $e_f^i(1+\frac{1}{2e_p^i+e_f^i}) = F(1+\frac{1}{2e_p^f+F})$. Thus, the only possible case is $e_p^f=e_p^i$.

We have proved that if $s_i \in S_F^{GP13}$, then $e_f^i=F$ and $e_p^f=e_p^i$, which imply $s_i \in Y^{Op}$. Therefore, $S_F^{GP13} \subseteq Y^{Op}$.

In conclusion, we have proved $Y^{Op} \subseteq S_F^{GP13}$ and $S_F^{GP13} \subseteq Y^{Op}$. Therefore, we have $S_F^{GP13} = Y^{Op}$.

3. To prove that $S_A^{GP13} = Z^{Op}$.
After Definition 1, we have $S_A^{GP13}=S \setminus S_B^{GP13} \setminus S_F^{GP13}$ and $Z^{Op}=S \setminus X^{Op} \setminus Y^{Op}$, where S denotes the set of all investigated statements. Since we have proved $S_B^{GP13} = X^{Op}$ and $S_F^{GP13} = Y^{Op}$, it is obvious that $S_A^{GP13} = Z^{Op}$.

Now, we are ready to prove that GP13, Naish1 and Naish2 belong to the same group of equivalent formulæ (referred to as ER1').

Proposition 1. *GP13 $\leftrightarrow$ Naish1 and GP13 $\leftrightarrow$ Naish2.*

Proof. Refer to Lemma 3 and Lemma 4, we have $S_B^{N1} = S_B^{N2} = S_B^{GP13}$, $S_F^{N1} = S_F^{N2} = S_F^{GP13}$ and $S_A^{N1}=S_A^{N2}=S_A^{GP13}$, respectively. After Theorem 1, GP13 $\leftrightarrow$ Naish1 and GP13 $\leftrightarrow$ Naish2.

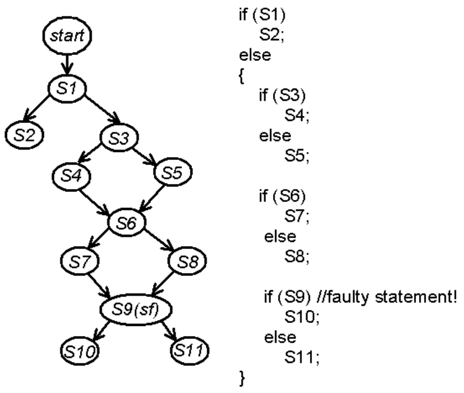
Apart from GP13, we have three new maximal GP-evolved formulæ for programs with single fault, namely, GP02, GP03 and GP19. Unlike GP13, these three formulæ do not belong to ER1' or ER5.

Proposition 2. *GP02, GP03, GP19, ER1' and ER5 are distinct maximal formulæ (or groups of equivalent formulæ).*

Proof. To prove this, we will demonstrate that neither $R_1 \rightarrow R_2$ nor $R_2 \rightarrow R_1$ is held, where R_1 and R_2 are any two of these five formulæ (or groups of equivalent formulæ). Consider the following two program PG_1 and PG_2 as shown in Figure 2 and Figure 3, respectively. Suppose two test suites $TS1_1$ and $TS1_2$ are applied on PG_1 and two test suites $TS2_1$ and $TS2_2$ are applied on PG_2. Vector i with respect to these test suites and programs are listed in Table 2.

Table 3 lists the statement divisions for these five formulæ with respect to $TS1_1$ and $TS1_2$ applied on PG_1, while Table 4 lists the statement divisions for these five formulæ with respect to $TS2_1$ and $TS2_2$ applied on PG_2.

Suppose we adopt the “ORIGINAL ORDER” as the tie-breaking scheme. Then the corresponding rankings of the faulty statement for these five formulæ are as Table 5. From this table, we have demonstrated that

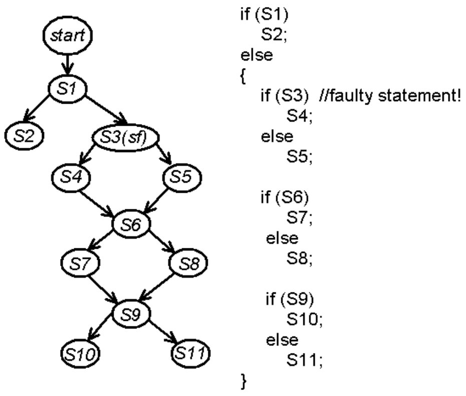


```
if (S1)
    S2;
else
{
   if (S3)
      S4;
   else
      S5;

   if (S6)
      S7;
   else
      S8;

   if (S9) //faulty statement!
      S10;
   else
      S11;
}
```

Fig. 2. Program PG_1



```
if (S1)
    S2;
else
{
   if (S3)  //faulty statement!
      S4;
   else
      S5;

   if (S6)
      S7;
   else
      S8;

   if (S9)
      S10;
   else
      S11;
}
```

Fig. 3. Program PG_2

Table 2. i for PG_1 and PG_2 with different test suites

Statement	i=<$e_f^i, e_p^i, n_f^i, n_p^i$>			
	$TS1_1$	$TS1_2$	$TS2_1$	$TS2_2$
s_1	<1, 6, 0, 0>	<1, 8, 0, 0>	<2, 15, 0, 0>	<10, 15, 0, 0>
s_2	<0, 1, 1, 5>	<0, 6, 1, 2>	<0, 1, 2, 14>	<0, 1, 10, 14>
s_3	<1, 5, 0, 1>	<1, 2, 0, 6>	<2, 14, 0, 1>	<10, 14, 0, 1>
s_4	<1, 4, 0, 2>	<1, 1, 0, 7>	<1, 7, 1, 8>	<9, 0, 1, 15>
s_5	<0, 1, 1, 5>	<0, 1, 1, 7>	<1, 7, 1, 8>	<1, 14, 9, 1>
s_6	<1, 5, 0, 1>	<1, 2, 0, 6>	<2, 14, 0, 1>	<10, 14, 0, 1>
s_7	<1, 4, 0, 2>	<1, 1, 0, 7>	<1, 8, 1, 7>	<5, 6, 5, 9>
s_8	<0, 1, 1, 5>	<0, 1, 1, 7>	<1, 6, 1, 9>	<5, 8, 5, 7>
s_9	<1, 5, 0, 1>	<1, 2, 0, 6>	<2, 14, 0, 1>	<10, 14, 0, 1>
s_{10}	<1, 4, 0, 2>	<1, 1, 0, 7>	<1, 9, 1, 6>	<1, 12, 9, 3>
s_{11}	<0, 1, 1, 5>	<0, 1, 1, 7>	<1, 5, 1, 10>	<9, 2, 1, 13>

- With $TS1_2$ ER1' $\rightarrow$ GP02 does not hold; with $TS2_1$ GP02 $\rightarrow$ ER1' does not hold.
- With $TS1_2$ ER5 $\rightarrow$ GP02 does not hold; with $TS2_1$ GP02 $\rightarrow$ ER5 does not hold
- With $TS1_1$ ER1' $\rightarrow$ GP03 does not hold; with $TS1_2$ GP03 $\rightarrow$ ER1' does not hold.
- With $TS1_1$ ER5 $\rightarrow$ GP03 does not hold; with $TS1_2$ GP03 $\rightarrow$ ER5 does not hold.
- With $TS1_1$ ER1' $\rightarrow$ GP19 does not hold; with $TS1_2$ GP19 $\rightarrow$ ER1' does not hold.
- With $TS1_1$ ER5 $\rightarrow$ GP19 does not hold; with $TS1_2$ GP19 $\rightarrow$ ER5 does not hold.

Table 3. Statement division for PG_1 with $TS1_1$ and $TS1_2$

Statement	$TS1_1$	$TS1_2$
ER1'	$S_B^R = \{s_4, s_7, s_{10}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_5, s_8, s_{11}\}$	$S_B^R = \{s_4, s_7, s_{10}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_5, s_8, s_{11}\}$
ER5	$S_B^R = \emptyset$ $S_F^R = \{s_1, s_3, s_4, s_6, s_7, s_9, s_{10}\}$ $S_A^R = \{s_2, s_5, s_8, s_{11}\}$	$S_B^R = \emptyset$ $S_F^R = \{s_1, s_3, s_4, s_6, s_7, s_9, s_{10}\}$ $S_A^R = \{s_2, s_5, s_8, s_{11}\}$
GP02	$S_B^R = \{s_4, s_7, s_{10}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_5, s_8, s_{11}\}$	$S_B^R = \emptyset$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$
GP03	$S_B^R = \{s_1\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$	$S_B^R = \{s_1, s_2, s_5, s_8, s_{11}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_4, s_7, s_{10}\}$
GP19	$S_B^R = \{s_1\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$	$S_B^R = \{s_1, s_4, s_7, s_{10}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_2, s_5, s_8, s_{11}\}$

- With $TS1_1$ GP02 $\rightarrow$ GP03 does not hold; with $TS1_2$ GP03 $\rightarrow$ GP02 does not hold.
- With $TS1_1$ GP02 $\rightarrow$ GP19 does not hold; with $TS1_2$ GP19 $\rightarrow$ GP02 does not hold.
- With $TS2_1$ GP03 $\rightarrow$ GP19 does not hold; with $TS2_2$ GP19 $\rightarrow$ GP03 does not hold.

In summary, we have proved that for any two of these five formulæ (or groups of equivalent formulæ) R_1 and R_2, neither $R_1 \rightarrow R_2$ nor $R_2 \rightarrow R_1$ is held. Therefore, GP02, GP03, GP19, ER1' and ER5 are five distinct maximal formulæ (or groups of equivalent formulæ).

4 Discussion

Yoo [23] used a small number of programs and faults to evolve new risk evaluation formulæ: more precisely, four subject programs and 20 mutants for evolution. To quote Yoo, "the results should be treated with caution" since "there is no guarantee that the studied programs and faults are representative of all possible programs and faults".

In this paper, we use the theoretical framework recently proposed by Xie et al. [21] to analyse Yoo's GP-evolved risk evaluation formulæ for programs with single fault. Among Yoo's formulæ, four have been proved to be maximal, namely, GP02, GP03, GP13 and GP19, where GP13 forms a new maximal group of equivalent formulæ with Naish1 and Naish2. This new maximal group is referred to as ER1'); while GP02, GP03 and GP19 are distinct to ER1' and ER5. Moreover, ER1' is strictly better than the remaining 16 GP-evolved formulæ under investigation.

Results in this paper are exempt from the inherent disadvantages of experimental studies, and hence are definite conclusions for any program and fault

Table 4. Statement division for PG_2 with $TS2_1$ and $TS2_2$

Statement	$TS2_1$	$TS2_2$
ER1'	$S_B^R = \emptyset$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$	$S_B^R = \emptyset$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$
ER5	$S_B^R = \emptyset$ $S_F^R = \{s_1, s_3, s_6, s_9\}$ $S_A^R = \{s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$	$S_B^R = \emptyset$ $S_F^R = \{s_1, s_3, s_6, s_9\}$ $S_A^R = \{s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$
GP02	$S_B^R = \{s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2\}$	$S_B^R = \{s_4, s_{11}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_5, s_7, s_8, s_{10}\}$
GP03	$S_B^R = \{s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1\}$	$S_B^R = \emptyset$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_1, s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$
GP19	$S_B^R = \{s_1\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_2, s_4, s_5, s_7, s_8, s_{10}, s_{11}\}$	$S_B^R = \{s_1, s_4, s_{11}\}$ $S_F^R = \{s_3, s_6, s_9\}$ $S_A^R = \{s_2, s_5, s_7, s_8, s_{10}\}$

Table 5. Rankings of faulty statement for five formulæ

Statement	PG_1 (s_f=s_9)		PG_2 (s_f=s_3)	
	$TS1_1$	$TS1_2$	$TS2_1$	$TS2_2$
ER1'	6	6	1	1
ER5	6	6	2	2
GP02	6	3	7	3
GP03	4	8	8	1
GP19	4	7	2	4

under the assumptions that are commonly adopted by the SBFL community. It is a surprise that without exhausting all possible programs and faults, GP can still deliver maximal formulæ. Moreover, the process of evolving a risk evaluation formula is totally automatic and does not need any human intelligence. Thus, the cost of designing risk evaluation formulæ can be significantly reduced.

From analysing formulæ in ER1', we note some common features. First, they all involve two independent parameters[2] e_f and e_p. Secondly, all these three formulæ comply with the commonly adopted intuition that statements associated with more *failed* or less *passed* testing results should never have lower risks. Finally, in all these three formulæ, any statement s_i with $e_f^i{<}F$ always has lower risk value than statement s_j with $e_f^j{=}F$. With respect to ER1', the evolved formula follows the known intuition. However, interestingly enough, the other maximal formulæ, GP02, GP03, and GP19, do not conform to the same intuition. Let us elaborate. Given two statements, s_1 and s_2:

[2] By definition, $n_p = P{-}e_p$ and $n_f = F{-}e_f$.

- **GP02**: If ${e_p}^1={e_p}^2$, then ${e_f}^1>{e_f}^2$ implies GP02(s_1)>GP02(s_2), which is consistent with the commonly adopted intuition. However, if ${e_f}^1={e_f}^2$, then ${e_p}^1<{e_p}^2$ does not necessarily imply GP02(s_1)≥GP02(s_2). For example, ${e_f}^1 = {e_f}^2 = 1$, P=8, ${e_p}^1$=1 and ${e_p}^2$=2, then we have GP02(s_1)=$2\cdot(1+\sqrt{8-1})+1$, which is less than GP02(s_2)=$2\cdot(1+\sqrt{8-2})+\sqrt{2}$. This does not comply with the commonly adopted intuition.
- **GP03**: If ${e_p}^1 = {e_p}^2$, then ${e_f}^1 > {e_f}^2$ does not necessarily imply GP03(s_1) ≥ GP03(s_2). For example, ${e_p}^1={e_p}^2$=25, ${e_f}^1$=2 and ${e_f}^2$=1, then we have GP03(s_1)=1, which is less than GP03(s_2)=2. This does not comply with the commonly adopted intuition. Moreover, if ${e_f}^1={e_f}^2$, then ${e_p}^1<{e_p}^2$ does not necessarily imply GP03(s_1)≥GP03(s_2). For example, ${e_f}^1={e_f}^2$=1, ${e_p}^1$=16 and ${e_p}^2$=25, then we have GP03(s_1)=$\sqrt{3}$, which is less than GP03(s_2)=2. As a consequence, the commonly adopted intuition is not complied.
- **GP19**: If ${e_p}^1={e_p}^2$, then ${e_f}^1>{e_f}^2$ does not necessarily imply GP19(s_1) ≥ GP19(s_2). For example, P=20, ${e_p}^1={e_p}^2$=10; F=4, ${e_f}^1$=2 and ${e_f}^2$=1, then we have GP19(s_1)=0, which is less than GP19(s_2)=$\sqrt{2}$. This example demonstrates that the commonly adopted intuition is not complied. Moreover, if ${e_f}^1={e_f}^2$, then ${e_p}^1<{e_p}^2$ does not necessarily imply GP19(s_1)≥GP19(s_2). For example, F=2, ${e_f}^1={e_f}^2$=1; P=10, ${e_p}^1$=8 and ${e_p}^2$=9, then we have GP19(s_1)=$\sqrt{6}$, which is less than GP19(s_2)=$\sqrt{8}$. This does not comply with the commonly adopted intuition.

Formulæ defined by human beings are more likely to be confined to the perceived intuition and background of the designer. Thus, it is possible that some maximal formulæ may be overlooked by humans. However, GP does not suffer from this problem and has the advantage of being unbiased. As explained in the above examples for GP02, GP03 and GP19, GP is able to define maximal formulæ based on intuitions that humans would rarely consider.

5 Related Work

Spectrum-Based Fault Localisation (SBFL) is also referred to as statistical fault localisation: it aims to identify statements that are suspected to contain the root cause for software failure by examining a large number of passing and failing test executions. Tarantula [14] was the first SBFL risk evaluation formula that originally started its life as a visualisation tool. Many other formulaæ followed, applying different statistical analysis to compute the ranking of suspiciousness statements [2, 3, 5, 18, 20], all of which have been designed manually: Yoo [23] is the first to use Genetic Programming to automatically evolve an SBFL formula.

The predominant method for evaluating SBFL risk evaluation formulæ in the literature has been empirical studies [6, 13, 24]. However, recent advances in theoretical analysis of SBFL have provided optimality proof for specific program structures [16], as well as proofs of equivalence/dominance relations for arbitrary combinations of faulty source code and test suites [21].

6 Conclusion

Search-based techniques have been widely used in software engineering, such as testing, maintenance, etc [8, 10]. Recently, Yoo [23] has successfully utilized a search-based technique, namely, Genetic Programming, to generate effective risk evaluation formulæ for SBFL. In this paper, by using the recently developed theoretical framework by Xie et al. [21] on Yoo's GP-evolved formulæ, we have demonstrated that four formulæ are maximal for programs with single fault, namely, GP02, GP03, GP13 and GP19. The results provide a strong support that Genetic Programming can be an ideal tool for designing risk evaluation formulæ. GP not only can deliver maximal formulæ having the same features as some maximal formulæ designed by humans, but also can help to provide novel insights and intuitions about effective formulæ that humans may overlook.

Acknowledgement. This work was partly funded by the Engineering and Physical Sciences Research Council [grant no. EP/J017515/1] and the Australian Research Council [grant no. DP 120104773].

References

1. Abreu, R., Zoeteweij, P., van Gemund, A.J.C.: An evaluation of similarity coefficients for software fault localization. In: Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing, pp. 39–46. Riverside, USA (2006)
2. Abreu, R., Zoeteweij, P., van Gemund, A.J.C.: An observation-based model for fault localization. In: Proceedings of the 2008 International Workshop on Dynamic Analysis: Held in Conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), WODA 2008, pp. 64–70. ACM, New York (2008)
3. Artzi, S., Dolby, J., Tip, F., Pistoia, M.: Directed test generation for effective fault localization. In: Proceedings of the 19th International Symposium on Software Testing and Analysis, ISSTA 2010, pp. 49–60. ACM, New York (2010)
4. Chen, M., Kiciman, E., Fratkin, E., Fox, A., Brewer, E.: Pinpoint: problem determination in large, dynamic internet services. In: Proceedings of the 32th IEEE/IFIP International Conference on Dependable Systems and Networks, Washington DC, USA, pp. 595–604 (2002)
5. Dallmeier, V., Lindig, C., Zeller, A.: Lightweight bug localization with ample. In: Proceedings of the Sixth International Symposium on Automated Analysis-driven Debugging, AADEBUG 2005, pp. 99–104. ACM, New York (2005)
6. DiGiuseppe, N., Jones, J.A.: On the influence of multiple faults on coverage-based fault localization. In: Proceedings of the 2011 International Symposium on Software Testing and Analysis, ISSTA 2011, pp. 210–220. ACM, New York (2011)
7. de Freitas, F.G., de Souza, J.T.: Ten years of search based software engineering: A bibliometric analysis. In: Cohen, M.B., Ó Cinnéide, M. (eds.) SSBSE 2011. LNCS, vol. 6956, pp. 18–32. Springer, Heidelberg (2011)
8. Harman, M., Jones, B.: Search based software engineering. Information and Software Technology 43(14), 833–839 (2001)

9. Harman, M.: The current state and future of search based software engineering. In: FOSE 2007: 2007 Future of Software Engineering, pp. 342–357. IEEE Computer Society, Washington, DC (2007)
10. Harman, M.: The relationship between search based software engineering and predictive modeling. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, Timişoara, Romania, pp. 1:1–1:13 (2010)
11. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys 45(1), 11:1–11:61 (2012)
12. Jones, J.A., Harrold, M.J., Stasko, J.: Visualization of test information to assist fault localization. In: Proceedings of the 24th International Conference on Software Engineering, Florida, USA, pp. 467–477 (2002)
13. Jones, J.A., Harrold, M.J.: Empirical evaluation of the tarantula automatic fault-localization technique. In: Proceedings of the 20th International Conference on Automated Software Engineering (ASE 2005), pp. 273–282. ACM Press (2005)
14. Jones, J.A., Harrold, M.J., Stasko, J.T.: Visualization for fault localization. In: Proceedings of ICSE Workshop on Software Visualization, pp. 71–75 (2001)
15. McMinn, P.: Search-based software test data generation: A survey. Software Testing, Verification and Reliability 14(2), 105–156 (2004)
16. Naish, L., Lee, H.J., Ramamohanarao, K.: A model for spectra-based software diagnosis. ACM Transactions on Software Engineering Methodology 20(3), 11:1–11:32 (2011)
17. Räihä, O.: A survey on search-based software design. Computer Science Review 4(4), 203–249 (2010)
18. Renieres, M., Reiss, S.: Fault localization with nearest neighbor queries. In: Proceedings of the 18th International Conference on Automated Software Engineering, pp. 30–39 (October 2003)
19. de Souza, J.T., Maia, C.L., de Freitas, F.G., Coutinho, D.P.: The human competitiveness of search based software engineering. In: Proceedings of 2nd International Symposium on Search based Software Engineering (SSBSE 2010), pp. 143–152. IEEE Computer Society Press, Benevento (2010)
20. Wong, W.E., Qi, Y., Zhao, L., Cai, K.Y.: Effective fault localization using code coverage. In: Proceedings of the 31st Annual International Computer Software and Applications Conference, COMPSAC 2007, vol. 1, pp. 449–456. IEEE Computer Society, Washington, DC (2007)
21. Xie, X.Y., Chen, T.Y., Kuo, F.C., Xu, B.W.: A Theoretical Analysis of the Risk Evaluation Formulas for Spectrum-Based Fault Localization. Accepted by the ACM Transactions on Software Engineering and Methodology (2012)
22. Xie, X.: On the analysis of spectrum-based fault localization. Ph.D. thesis, Swinburne University of Technology (May 2012)
23. Yoo, S.: Evolving human competitive spectra-based fault localisation techniques. In: Fraser, G., Teixeira de Souza, J. (eds.) SSBSE 2012. LNCS, vol. 7515, pp. 244–258. Springer, Heidelberg (2012)
24. Yu, Y., Jones, J.A., Harrold, M.J.: An empirical study of the effects of test-suite reduction on fault localization. In: Proceedings of the International Conference on Software Engineering (ICSE 2008), pp. 201–210. ACM Press (May 2008)