The Journal of Systems and Software 147 (2019) 106–123

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Controversy Corner

On the analysis of spectrum based fault localization using hitting sets

Jingxuan Tu[a], Xiaoyuan Xie[a,*], Tsong Yueh Chen[c], Baowen Xu[b]

[a] School of Computer Science, Wuhan University, China
[b] State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, China
[c] Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia

ARTICLE INFO

Article history:
Received 25 February 2018
Revised 17 August 2018
Accepted 8 October 2018
Available online 9 October 2018

Keywords:
Spectrum based fault localization
Risk evaluation formulas
Dynamic slicing
Hitting set
Theoretical analysis

ABSTRACT

Combining spectrum-based fault localization (SBFL) with other techniques is generally regarded as a feasible approach as advantages from both techniques would be preserved. Sendys which combines SBFL with slicing-hitting-set-computation is one of the promising techniques. However, all current evaluations on Sendys were obtained via empirical studies, which have inevitable threats to validity. Besides, purely empirical studies cannot reveal the essential reason that why Sendys performs well or badly, and whether all the complicated computations are necessary. Therefore, in this paper, we provide an in-depth theoretical analysis on Sendys, which can give definite and convincing conclusions. We generalize our previous theoretical framework on SBFL, to make it applicable to combined techniques like Sendys. We first provide a variant of current Sendys by patching its loophole of ignoring "zero or negative risk values" in normalization. This variant plays as a substitution of the original Sendys, as well as one of the baselines in our analysis. Then, by modifying a few steps of this variant, we propose an enhanced Sendys and theoretically prove its superiority over several other methods in single-fault scenario. Moreover, we provide a short-cut reformulation of the enhanced Sendys by preserving its performance, but only requiring very simple computations. And it is proved to be even better than traditional SBFL maximal formulas. As a complementary, our empirical studies with 13 subject programs demonstrate the obvious superiority of the enhanced Sendys, as well as its stability across different formulas in single-fault scenario. For multiple-fault cases, the variant of Sendys is observed to have the best performance. Besides, this variant has shown great helpfulness in improving the bad performance of the original Sendys when encountering the *NOR* problem.

1. Introduction

It is commonly recognized that testing and debugging are expensive and time-consuming activities in software engineering (Liu et al., 2006; Fang et al., 2012). In order to remove the faults in software, 50 to 80% of development and maintenance efforts would be consumed (Collofello and Woodfield, 1989). Determining where the fault lies in a piece of code is one of the most essential but difficult tasks. Therefore, many researchers have proposed various automatic and effective techniques for localizing the faults, in order to reduce the cost of debugging.

Spectrum-based fault localization (referred to as SBFL) (Jones et al., 2002) has been demonstrated to be a light-weight and effective technique through empirical studies (Abreu et al., 2009b; Jones and Harrold, 2005; Wong et al., 2010) and theoretical analysis (Lee et al., 2009; Naish et al., 2011; Xie et al., 2013a; 2013b; Chen et al., 2015). Generally speaking, SBFL considers both passing and failing test executions. By utilizing program spectra information and test results, SBFL estimates the risk value with a risk formula for each entity of a program to indicate how likely it is faulty. In principle, SBFL can be configured with any risk formula. In the past two decades, proposing and analyzing different formulas have been popular research topics (Liblit et al., 2005; Janssen et al., 2009; Abreu et al., 2009b; Wong et al., 2010; Xie et al., 2013a).

As a light-weight technique, SBFL was not intended for sophisticated debugging (such as model-based or state-based approaches). Therefore, there are many studies combining SBFL with other techniques (Wong and Qi, 2006; Tu et al., 2012; Hofer and Wotawa, 2012; Xu et al., 2013; Abreu et al., 2009a), which intended to utilize the strengths of each individual approach for better localization results. Hofer and Wotawa combined SBFL and slicing-hitting-set-computation (SHSC) (Wotawa, 2010), proposing Sendys (Hofer and Wotawa, 2012).

* Corresponding author.
E-mail address: xxie@whu.edu.cn (X. Xie).

https://doi.org/10.1016/j.jss.2018.10.013
0164-1212/

In Sendys, each statement is firstly assigned with a risk value calculated via a traditional SBFL risk formula. The risk value of each statement is referred to as the initial risk value, which will be then updated by means of the minimal hitting sets (described in Algorithm 1 in Section 2). Sendys outputs a final statements ranking based on the updated risk values. Experimental studies have shown that Sendys is helpful in obtaining more accurate fault localization result (Hofer and Wotawa, 2012).

However, all current evaluations about Sendys were exclusively obtained from experiments. As discussed in Xie et al. (2013a), pure experiments have inevitable threats to validity. All the observed results and conclusions strongly depend on the selected experimental setup (i.e. the programs, faults, test suites, etc.). There is no guarantee that the observed results always hold when we change to another setup. Moreover, Sendys requires much more complicated computation than the traditional SBFL. However, the lack of theoretical analysis makes it infeasible to have an in-depth perception about the essence of this technique. In particular, is Sendys good enough and is there any chance that we can modify the algorithm to achieve a definite better performance? Are all of the complicated computations really necessary? To answer these questions, a theoretical analysis is of great help.

In our previous study, we developed a set-theory-based framework to analyze the performance of different SBFL risk formulas (Xie et al., 2013a). We summarized over 30 formulas, and revealed a performance hierarchy that is intrinsically existing among them. Follow-up study of Xie et al. (2013a) further proved a necessary and sufficient condition for any given formula of being "maximal", and explained why a formula performs well by visualizing its landscape (Yoo et al., 2017). In other words, with this theoretical framework, no experimental analysis is required and the conclusions are definite in any scenario (or setup).

However, the above theoretical framework was not designed to compare different fault localization methods, but was exclusively for comparing different risk formulas with fixed localization algorithm (i.e. SBFL). Therefore, it cannot be directly applied on the analysis of this paper, that is, to analyze the entire method as an integration of the fault localization algorithm and the risk formula. Thus, we generalize the traditional framework to make it applicable to our context.

In principle, Sendys is not restricted to any particular risk formula but it has actually overlooked the possibility of zero or negative initial risk values computed by some risk formulas. In such a case, Sendys may encounter a problem in normalizing these zero or negative risk values, which is referred to as *NOR* problem in this paper. This problem makes the theoretical analysis between Sendys and other methods neither feasible nor meaningful. Therefore, we must first make a correction on the original Sendys and propose a variant of it that can well handle the zero and negative risk values. This variant plays as a substitution of the original Sendys, as well as one of the baselines in our analysis.

With the generalized theoretical framework, we prove some interesting properties of the above variant, which inspire us to propose an enhanced Sendys by modifying a few steps of this variant. We theoretically prove the superiority of the enhanced Sendys over both the traditional SBFL and the baseline variant in single-fault scenario. Moreover, we provide a short-cut reformulation of the enhanced Sendys by preserving its performance, but requiring only a few simple computation steps. And it is proved to be even better than traditional SBFL maximal formulas.

In addition to the theoretical analysis, we also conduct empirical studies as complementary. We first compare the traditional SBFL, the original Sendys, the variant that patches the loophole of *NOR* problem and the enhanced Sendys in single-fault scenario. The empirical results are consistent with theoretical analysis, and further demonstrate the significant advantages of the enhanced Sendys over other methods, as well as its stability across different formulas. We also compare the traditional SBFL, the original Sendys, and the variant of Sendys in multiple-fault cases, where the variant demonstrates the best ability in locating multiple faults simultaneously among the three methods. As a reminder, even though the variant provides only a small correction to the original Sendys, our experiments have shown that in both single-fault and multiple-fault cases, it is of great help in improving the bad performance of Sendys when encountering the *NOR* problem.

$$\begin{array}{c} \textit{Test Suite} \\ \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{pmatrix} \end{array} \quad \begin{array}{c} s(1)\ \ s(2)\ \ \cdots\ \ s(N) \\ \begin{pmatrix} 1/0 & 1/0 & \cdots & 1/0 \\ 1/0 & 1/0 & \cdots & 1/0 \\ \vdots & \vdots & \ddots & \vdots \\ 1/0 & 1/0 & \cdots & 1/0 \end{pmatrix} \end{array} \quad \begin{array}{c} \textit{Test Result} \\ \begin{pmatrix} p/f \\ p/f \\ \vdots \\ p/f \end{pmatrix} \end{array}$$

Fig. 1. Spectrum matrix for SBFL.

The contributions of this work can be summarized as follows:

1. We propose a variant of Sendys to patch its loophole of overlooking non-positive initial risk values, and an enhanced version of Sendys to improve its performance.
2. We generalize our previous theoretical framework to support the analysis of fault localization methods which include both the localization algorithm and the risk formulas. Based on the generalized framework, we analyze the properties of the above Sendys variant and prove the superiority of the proposed enhanced Sendys in single-fault scenario.
3. We provide a short-cut reformulation of the enhanced Sendys by preserving its performance, but only requiring very simple computations. Moreover, it is proved to be even better than traditional SBFL maximal formulas.
4. We conduct empirical studies to complement our theoretical analysis. The results demonstrate the significant advantages of the enhanced Sendys, as well as its stability across different formulas in single-fault scenario. We also observed the best performance of the variant in multiple-fault scenarios, and its great helpfulness in improving the bad performance of the original Sendys when encountering the *NOR* problem.

The rest of this paper is organized as follows. Section 2 describes the general process of Sendys and the set theory based theoretical analysis framework which theoretically compares the performance of different risk formulas. Section 3 introduces a variant that patches the loophole of overlooking zero or negative risk values. Section 4 proposes an enhanced Sendys and gives theoretical analysis in single-fault scenario. Section 5 gives experimental studies, followed by Section 6 that discusses the applicability of our analysis. Section 7 presents related work and Section 8 provides the conclusions.

2. Background

2.1. Fault localization combining slicing-hitting-set-computation with SBFL

Hofer and Wotawa developed a fault localization approach named Sendys (Hofer and Wotawa, 2012) which is a combination of SBFL and slicing-hitting-set-computation (SHSC) (Wotawa, 2010). Sendys has the synergy between of SBFL and dynamic slicing which are both based on program execution traces.

First, in SBFL, a program spectrum (shown in Fig. 1) is constituted of a binary matrix whose columns record the run-time profile to various program entities for test executions, and also a test result vector indicating whether the program execution passed

or failed. Run-time profile represents the coverage information of each program entity whether being covered by each test execution. For example, in the first row and column s(1), 1 or 0 represents a test case t_1 executing or not executing program statement s(1) respectively. The corresponding testing result is recorded in the first row of the testing result vector where *p* means passed and *f* means failed. SBFL aims to identify program entities that are most likely to be faulty. Hence, a variety of formulas are proposed to estimate the risk of being faulty for each program entity, based on different intuitions.

A commonly adopted type of program entity is statement. For each statement s_i, the corresponding execution result can be represented as a tuple $i =< a^i_{np}, a^i_{ep}, a^i_{nf}, a^i_{ef} >$, where a^i_{ep} and a^i_{ef} represent the number of test cases that execute the statement and return the testing result of passed or failed, respectively; a^i_{np} and a^i_{nf} represent the number of test cases that do not execute it and return the testing result of passed or failed, respectively. As a reminder, $a^i_{nf} + a^i_{ef} = F$ and $a^i_{np} + a^i_{ep} = P$ where F represents the total failed test executions and P represents the total passed test executions. A risk formula R is then applied to calculate the suspiciousness score indicating the risk of being faulty for each statement based on its tuple. The other part of Sendys is SHSC. In SHSC, a conflict is a set of components in a system causing the misbehavior. The incorrectness of components leads to the inconsistency between observations and expectations. Correspondingly, the components in a conflict cannot be correct simultaneously. The set of all statements in a failed dynamic slice is responsible for a program failure and reveals the detected program misbehavior. Thus, this statement set forms a conflict which means that these statements cannot be correct simultaneously (Wotawa, 2010; 2002; Reiter, 1987). Note that the faults in a program are non-omission faults which are faulty statements not related with missing code. A diagnosis is a possible root cause for the failure. Taking for example the two failed dynamic slices: {1, 3} and {2, 3, 4}. Each failed dynamic slice forms a conflict where the equivalence of slices and conflicts has been proved in Wotawa (2002). From the two conflicts we know that statements 1, 3 cannot be correct simultaneously and neither are statements 2, 3, 4. If we select a statement from each conflict such as {1, 4}, then {1, 4} is a diagnosis which is a possible root cause for the failure.

From the previous work of Wotawa (2010), hitting sets are used to state the relationship between diagnoses and conflicts which are formally defined as follows: For a set of sets *DS*, a set *H* is a hitting set if and only if the intersection between *H* and any element *x* of *DS* is not empty, i.e., $\forall x(x \bigcap H \neq \emptyset)$. The hitting set *H* is minimal if none of its subsets is a hitting set. In the context of program debugging, a minimal diagnosis is a minimal hitting set of all conflicts that correspond to all failed dynamic slices.

For example, the minimal hitting sets for the two failed dynamic slices are: {3}, {1, 2}, {1, 4}. They compute the minimal hitting sets using a variant of the algorithm (Greiner et al., 1989) which is introduced in Wotawa (2010). This algorithm specifies a maximum number of generating hitting sets and does not compute all possible minimal hitting sets. As a reminder, all possible minimal hitting sets can be divided into two categories: single-element minimal hitting sets which contain only one statement; multiple-element minimal hitting sets which contain more than one statement.

In Hofer and Wotawa (2012), they have empirically showed that SHSC combining with SBFL techniques like Tarantula (Jones and Harrold, 2005), Ochiai and Jaccard (Abreu et al., 2009b) can provide better fault localization results than the individual techniques. In Sendys, they used Ochiai as the SBFL formula. Given a program *PG*, a test suite *TS*, the risk formula Ochiai and minimal hitting sets H^s generated by Wotawa (2010), the original approach of Sendys is shown in Algorithm 1 (Sendys): First, they collect spectra matrix *O*, and based on the spectra matrix the initial risk value *R(s)* for each statement is computed using Ochiai and the initial risk values are also normalized; then, compute the risk value $Pr(H_i)$ of each minimal hitting set based on the normalized statement risk values; finally, derive the risk value $R_{new}(s)$ of each statement being faulty associated with the minimal hitting set risk values, and return the final statements ranking in the descending order of the normalized statement risk value $R_{norm}(s)$.

Algorithm 1: Algorithm Sendys.

Input: Program *PG*, risk formula Ochiai, minimal hitting sets H^s and test suite *TS*

Output: Statements ranking

1 Compute the matrix *O* for program *PG* after the execution of test suite *TS*: $O = Matrix(PG, TS)$

2 Compute the risk value *R(s)* for each statement of program *PG*: $R(s) = Ochiai(s, O)$

3 Compute the normalized risk value $R_{norm}(s)$ for each statement of program *PG*: $R_{norm}(s) = R(s) \div \sum_i^{|PG|} R(s_i)$

4 Compute the risk value for each minimal hitting set H_i: $Pr(H_i) = \prod_{s \in H_i} R_{norm}(s) \times \prod_{s' \in PG \setminus H_i} (1 - R_{norm}(s'))$

5 Derive the risk of each statement being faulty based on H^s: $R_{new}(s) = \sum_{H_i \in H^s} Pr(H_i), \forall H_i \; such \; that \; s \in H_i$

6 Normalize the computed risk value of each statement: $R_{norm}(s) = R_{new}(s) \div \sum_i^{|PG|} R_{new}(s_i)$

7 Return the risk value ranking of each statement in the descending order of $R_{norm}(s)$

As a reminder, Sendys is not restricted to any particular risk formula. In Hofer and Wotawa (2012), the authors chose Ochiai to configure Algorithm 1. However, by configuring Algorithm 1 with other formulas, we noticed that this original Sendys has overlooked some possible scenarios. Without giving consideration to these scenarios, the original Sendys may lead to unreasonable results.

First, many risk formulas (such as Naish1, Goodman, etc.) may produce negative initial risk values. But simply normalizing these negative values based on Step 3 could lead to unreasonable results and a series of problems in the subsequent steps.

Secondly, many risk formulas may assign zero initial risk values. According to Step 4 in Algorithm 1 the minimal hitting sets including such statement are assigned zero risk value. In principle, the risk value computation of a minimal hitting set (noted as H_i) needs to consider the contribution of each statement in H_i. Assume the risk value of a statement s_1 in H_i to be zero, then H_i is assigned zero risk value. Unfortunately, the contributions of other statements having non-zero risk values in H_i are overlooked during computing the risk value of H_i.

We denote the problem of negative or zero initial risk values in the normalization step as the *NOR* problem, and we argue that strategies to handle such “negative or zero” cases are not trivial and should be explicitly indicated in the algorithm. We discuss such problem detailedly in Section 3.

2.2. Set theory based framework

In order to investigate the performance of various risk formulas in the single-fault scenario, Xie et al. (2013a) recently have developed a theoretical analysis framework based on set theory. Based on this framework, we make a simple generalization and conduct theoretical analysis on a combination of SBFL and SHSC in this paper. In this section, we first give a brief description of the theoretical analysis framework of Xie et al. (2013a).

Definition 1. Given a program with n statements $PG = \{s_1, s_2, \ldots, s_n\}$, a test suite of m test cases $TS = \{t_1, t_2, \ldots, t_m\}$, one faulty statement $s_f \in PG$ and an SBFL risk formula R. For each statement s_i, a vector $i = < a^i_{ef}, a^i_{ep}, a^i_{nf}, a^i_{np} >$ can be constructed accordingly and $R(s_i)$ is its risk value evaluated by R. Then the following three subsets can be defined with respect to the faulty statement s_f.

$$S^R_B = \{s_i \in PG \mid R(s_i) > R(s_f), 1 \le i \le n\}$$
$$S^R_F = \{s_i \in PG \mid R(s_i) = R(s_f), 1 \le i \le n\}$$
$$S^R_A = \{s_i \in PG \mid R(s_i) < R(s_f), 1 \le i \le n\}$$

Three subsets S^R_B, S^R_F and S^R_A consist of statements of which the risk values are higher than, equal to and lower than that of the faulty statement s_f, respectively. The effectiveness of risk formulas is measured by the percentage of code examination until the faulty statement is identified and the effectiveness measurement is referred to as *m*etric which is defined as follows.

Definition 2 (Expense)**.** Given the statements ranking of a risk formula R, assume the total number of statements in the ranking is n and the ranking of the faulty statement is $rank(f)$, the *Expense* of R is the ratio of $rank(f)$ and n.

It is not difficult to find that the ranking of the faulty statement is determined by the size of S^R_B and the order of the faulty statement in S^R_F. Thus a tie-breaking scheme is required to determine the order of the faulty statement in S^R_F and such a scheme is also required in the theoretical analysis. Intuitively, a tie-breaking scheme should preserve the relative order of any common statements in S^R_F returned by different risk formulas. Hence, the consistent tie-breaking scheme is defined as follows.

Definition 3 (Consistent tie-breaking)**.** Given any two statement sets S_1 and S_2 in which each statement has the same risk value. A tie-breaking scheme returns the ordered statement lists O_1 and O_2 for S_1 and S_2 respectively. If all elements common to S_1 and S_2 have the same relative order in O_1 and O_2, then the tie-breaking scheme is said to be consistent.

Assume the Expenses for the risk formulas R_1 and R_2 with respect to the same faulty statement are denoted as E_1 and E_2 respectively. Relations of better and equivalent between R_1 and R_2 are defined as follows.

Definition 4 (Better)**.** R_1 is said to be better than R_2 (denoted as $R_1 \to R_2$), if for any program, faulty statement s_f, test suite and consistent tie-breaking scheme, we have $E_1 \le E_2$.

Definition 5 (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 $E_1 = E_2$.

Theorem 1. *Given any two risk formulas R_1 and R_2, if we have $S^{R_1}_B \subseteq S^{R_2}_B$ and $S^{R_2}_A \subseteq S^{R_1}_A$ for any program, faulty statement s_f and test suite, then $R_1 \to R_2$.*

Theorem 1 gives a sufficient condition for "better" relation. According to the known condition of Theorem 1, we have that the statements belonging to $S^{R_1}_F$ should not belong to $S^{R_2}_A$, since if these statements belong to $S^{R_2}_A$, they should be in $S^{R_1}_A$. Hence these statements in $S^{R_1}_F$ and not in $S^{R_2}_F$ must belong to $S^{R_2}_B$. Hence, if the tie-breaking is consistent, then the common statements of $S^{R_1}_F$ and $S^{R_2}_F$ have the same ranking, but the non-common statements are ranked before the faulty statement in the ranking of R_2. As a consequence, it is not difficult to obtain $E_1 \le E_2$.

Theorem 2. *Given any two risk formulas R_1 and R_2, if we have $S^{R_1}_B = S^{R_2}_B$, $S^{R_1}_F = S^{R_2}_F$, $S^{R_1}_A = S^{R_2}_A$ for any program, faulty statement s_f and test suite, then $R_1 \leftrightarrow R_2$.*

Theorem 2 gives a sufficient condition for "equivalent" relation. According to the known condition of Theorem 2, if the tie-breaking scheme is consistent, the faulty statement in $S^{R_1}_F$ and $S^{R_2}_F$ has the same ranking. Hence, it is not difficult to obtain $E_1 = E_2$.

Based on Theorems 1 and 2, the performance of different risk formulas can be compared. The maximal risk formula is defined as follows.

Definition 6. A risk formula R_1 is said to be maximal in a set of risk formulas S, if for any risk formula $R_2 \in S$, $R_2 \to R_1$ implies $R_2 \leftrightarrow R_1$.

In this theoretical analysis framework, there are four assumptions as follows.

1. There is no test oracle problem in SBFL, that is, for any test execution, either pass or fail testing result can be determined.
2. Once the faulty statement is examined, the fault can always be identified. This is the assumption of "perfect bug detection".
3. The omission fault which is related with missing code is not in consideration.
4. In SBFL, the test suite contains at least one passing test case and one failing test case.

Based on this theoretical framework, they further investigate all possible formulas (Yoo et al., 2017), denoted as $\mathbb{R}$ which is defined as follows.

Definition 7 (. $\mathbb{R}$**)** $\mathbb{R}$ is a set of all possible mappings $\mathbb{R} = \{R \mid R : I \times I \times I \times I \to Real\}$ where the risk formula R is a mapping of four integers I to a real number *Real*.

They present SBFL formulas using three-dimensional coordinate in which x coordinate represents a_{ef}, y coordinate represents a_{ep} and z coordinate represents the corresponding risk values. According to the three-dimensional coordinate, they proposed the concepts of Faulty Border and Ranking.

Definition 8 (Faulty border)**.** Faulty border is a sequential of points where the value of a_{ef} on x coordinate is equal to F, which is $\langle (F, 0), \ldots, (F, a_{ep}), \ldots, (F, P) \rangle$ $(0 \le a_{ep} \le P)$.

Definition 9 (. Ranking) Given a formula R, let $op^{i,j} = < a^i_{ep}, a^j_{ep}, op >$ denote the comparative relation between two risk values of any two distinct points (F, a^i_{ep}) and (F, a^j_{ep}). Given that $a^i_{ep} < a^j_{ep}$, op can be either " $>$ ", " $<$ ", or " $=$ ". Ranking is such a set of $op^{i,j}$.

Then, they gave the sufficient and necessary condition of any formula being maximal. They use notation U_R to denote the set of points outside the faulty border which have higher than or equal to risk values of some points (F, a^i_{ep}) on the faulty border for a risk formula R. Theorem 3 gives the sufficient and necessary condition for a formula being maximal in $\mathbb{R}$. The concepts of "Faulty Border" and "U_R" are illustrated in Fig. 2.

Theorem 3. *Given any risk formula R, if and only if U_R is empty, then R is a maximal formula in $\mathbb{R}$.*

Based on Theorem 3, they proposed a simple maximal conversion. In this paper, we denote this maximal conversion as conversion $C_{\max}$. Any non-maximal risk formula R can be converted into a maximal formula through the process of conversion $C_{\max}$ which is defined as follows.

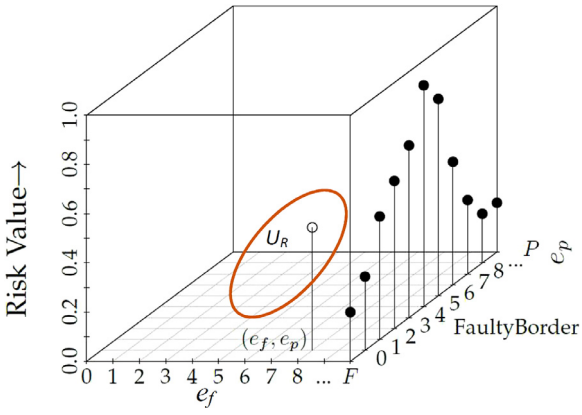


Fig. 2. Landscape for a risk evaluation formula (Yoo et al., 2017).

Definition 10 (Conversion. $C_{\max}$) Given any risk formula R, conversion $C_{\max}$ assigns identical risk values to points on the faulty border as R and to all points outside the faulty border smaller risk values than that of any point on the faulty border.

From Yoo et al. (2017), conversion $C_{\max}$ can be applied on different program spectra, such as execution slice, dynamic slice spectra, etc. In this paper, we theoretically compare conversion G with conversion $C_{\max}$ which is applied on execution slice and relevant dynamic slice respectively.

Finally, they prove that there exists no greatest formula which outperforms any formula belonging to the set of all formulas $\mathbb{R}$. Greatest formula is defined as follows.

Definition 11 (Greatest formula)**.** A risk formula R is said to be a greatest formula in $\mathbb{R}$, if for any formula $R' \in \mathbb{R}$ and $R' \neq R$, $R \rightarrow R'$ always holds.

3. Addressing the *NOR* problem

As mentioned in Section 2.1, in the process of Algorithm 1, negative or zero risk initial values are overlooked in the normalization step which is referred to as the *NOR* problem in this paper. In order to generalize the applicability of Algorithm 1, in this paper, we patch the related loophole.

First, SENDYS (Algorithm 1) does not explicitly indicate how to handle the cases with negative initial risk values in the normalization (i.e. Step 3). There is no problem for the adopted formula Ochiai. But we argue that handling such cases is not trivial for SENDYS being generally applied on other formulas (which have better performance than Ochiai but return negative risk values). Consider a sample scenario where two failed dynamic slices from Section 2.1: {1, 3} and {2, 3, 4}. All possible minimal hitting sets are {3}, {1, 2}, {1, 4}. Assume that statement 3 is faulty and the initial risk values of statements from 1 to 4 are −0.3, −0.1, −0.2 and −0.4, respectively. By simply applying Step 3 in Algorithm 1, the initial risk values for statement 1 to 4 are normalized as 0.3, 0.1, 0.2 and 0.4, which obviously does not make any sense. Thus, it is necessary to explicitly specify an appropriate strategy to handle such cases.

Secondly, in Algorithm 1, initial risk values may be also assigned as zero by some formulas. Then the minimal hitting sets including such statements are assigned zero risk values. In principle, the risk value computation of a minimal hitting set needs to consider the contribution of each statement in this minimal hitting set. Unfortunately, the contributions of other statements having non-zero risk values in this minimal hitting set are overlooked. To investigate the influence of zero initial risk values, let us consider the two possible types of formulas: The first type of formulas (such as Jaccard, Tarantula, Ochiai, etc) produce zero or positive risk values and assign zero values only to statements with $a_{ef} = 0$. Though we can never assert that such statements are fault-free, at least we know that they can never be the faults that trigger the currently observed failures (Xie et al., 2010). In such a case, the performance of SENDYS is not affected, since these statements can be viewed as "correct" in terms of the fault under discussion. According to Algorithm 1 Step 4, diagnoses containing such statements are assigned zero risk values and it is reasonable to assign any diagnosis containing correct statements zero risk values. On the other hand, diagnoses which only contain faulty statements are assigned positive values. From the above, the performance of SENDYS is not affected by such zero risk values.

However, there exists another type of formulas, such as Naish1, CBI, Wong3, etc., which may assign zero risk values to statements with $a_{ef} \neq 0$ and these statements cannot be excluded from potential faulty statements under discussion. In this case, zero initial risk values may give rise to a poor risk prediction. For example, consider two failed dynamic slices from Section 2.1: {1, 3} and {2, 3, 4}. All possible minimal hitting sets are {3}, {1, 2}, {1, 4}. Assume that statement 3 is faulty and the initial risk values of statements from 1 to 4 are 0, −0.3, −0.2 and −0.5 respectively. The initial rank of the faulty statement is 2. Directly following Algorithm 1 Step 3, the computed normalized risk values of the four statements are the same as their initial ones. Furthermore, according to Algorithm 1 Step 4, the risk values $Pr(H_i)$ of the three minimal hitting sets are −0.39, 0 and 0. Actually, from Algorithm 1 Step.5, final risk values become −0.39, 0, 0 and 0 for statements 1 to 4 respectively. The final rank of the faulty statement decreases to 4 due to the zero initial risk values.

In the original SENDYS (Hofer and Wotawa, 2012), the used formula Ochiai belongs to the first type. It is true that zero risk values assigned by Ochiai indicate such statements are non-faulty because no failed test cases execute such statements ($a_{ef} = 0$) according to the definition of Ochiai. From the above discussion, there is no issue of zero or negative initial risk values in the original SENDYS method.

However, in principle, the process of SENDYS is not designed for any particular formula. Formula Ochiai in Algorithm 1 can be replaced by any other formulas, with which the above *NOR* problems may be triggered. This problem makes the theoretical analysis between SENDYS and other methods neither feasible nor meaningful. Thus, we must first patch the loophole in the normalization process (Step 3) of Algorithm 1. Correspondingly, we propose M_1 to address the *NOR* problem, which is given in Algorithm 2 .

Algorithm 2: Algorithm M_1.

Input: Program *PG*, risk formula *Formula*, minimal hitting sets H^s and test suite *TS*
Output: Statements ranking
1 Compute the matrix *O* for program *PG* after the execution of test suite *TS*: $O = Matrix(PG, TS)$
2 Compute the risk value *R(s)* for each statement of program *PG*: $R(s) = Formula(s, O)$
3 Get minimum risk value p of all the initial risk values: $p = Min(R(s))$
4 If the minimum risk value p is non-positive or zero, then add $-p$ and a small real number to the risk value of each statement: $R(s) = R(s) - p + 0.00000001$
5 Compute the refined normalized risk value $R_{norm}(s)$ for each statement of program *PG*: $R_{norm}(s) = R(s) \div \sum_i^{|PG|} R(s_i)$
6 Compute the final statements ranking with Algorithm 1 (SENDYS). Start with Step.4 in Algorithm 1 (SENDYS)

During normalizing the initial risk values, in order to have a consistent performance, it is necessary to keep the relative order of the statements ranking. Given a statements ranking, applying addition mathematical operation to the risk value of each statement should not change the relative order of each statement and addition operation is an order-preserving operation. Hence, if there exist non-positive risk values in statements ranking, we add an appropriate positive value to the initial risk value of each statement.

As a consequence, M_1 has patched the loophole of the original SENDYS after a few corrections. It can be found that M_1 delivers identical performance with the original SENDYS if the adopted formula has no *NOR* problem.

4. Theoretical analysis in single-fault scenario

In this section, we will conduct a theoretical analysis in single-fault scenario. We first provide a simple generalization of the traditional framework to make it applicable to current context. Next we investigate the properties of M_1 and propose enhanced SENDYS (M_2). Then we prove the superiority of M_2 over M_1 and the traditional SBFL, and further reformulate this complicated algorithm M_2 into a simple conversion G. This conversion gives a short-cut explanation to M_2. Finally, we prove that the enhanced SENDYS can even outperform traditional SBFL maximal formulas. As a reminder, in the following discussion, we denote the basic SBFL method without using any hitting-set information as M_0 for convenience.

4.1. Preliminary - Generalized set theory based framework

The traditional theoretical framework in Xie et al. (2013a), Yoo et al. (2017) was designed to compare different risk formulas only, where the fault localization algorithm is fixed to SBFL. However, in the analysis on SENDYS, objects for comparison become the entire methods that integrate fault localization algorithm and risk formula. As a consequence, the traditional framework is not applicable. Therefore, in this subsection we provide a simple but indispensable generalization by extending the formula in the framework into the entire method. As a consequence, all the definitions and theorems become applicable to our context.

Statements ranking of such fault localization techniques can be also divided into three subsets. Correspondingly, we can extend Definition 1 in Section 2.2 to our context.

Definition 12. Given a program with n statements $PG = \{s_1, s_2, \ldots, s_n\}$, a test suite of m test cases $TS = \{t_1, t_2, \ldots, t_m\}$, one faulty statement $s_f \in PG$ and a method M which gives a whole ranking list to all the statements in PG. Let $M(s_i)$ denote the risk value assigned by M to a statement s_i. With respect to the faulty statement s_f, we have the three following subsets.

$$S_B^M = \{s_i \in PG \mid M(s_i) > M(s_f), 1 \le i \le n\}$$
$$S_F^M = \{s_i \in PG \mid M(s_i) = M(s_f), 1 \le i \le n\}$$
$$S_A^M = \{s_i \in PG \mid M(s_i) < M(s_f), 1 \le i \le n\}$$

Similar to the traditional SBFL, after applying method M, all statements will finally be ranked according to their risk values $M(s_i)$. Hence all statements of S_B^M will be ranked before s_f, all statements of S_F^M will be tied with s_f, and all statements of S_A^M will be ranked after s_f.

When adopting expense as performance metric, the ranking of s_f, which is decided by its relative of $M(s_f)$ to other s_i in the statements ranking, is the determinant of method $M's$ performance. It is not difficult to find that, by replacing R with M in theoretical analysis framework of Xie et al. (2013a) (introduced in Section 2.2), we can extend the theorems of "Better" and "Equivalent" in Section 2.2 to our context correspondingly.

Theorem 4. *Given any two combination methods M_i and M_j, if we have $S_B^{M_i} \subseteq S_B^{M_j}$ and $S_A^{M_j} \subseteq S_A^{M_i}$ for any program, faulty statement s_f and test suite, then $M_i \to M_j$.*

Theorem 5. *Given any two combination methods M_i and M_j, if we have $S_B^{M_i} = S_B^{M_j}$, $S_F^{M_i} = S_F^{M_j}$, $S_A^{M_i} = S_A^{M_j}$ for any program, faulty statement s_f and test suite, then $M_i \leftrightarrow M_j$.*

Based on Theorems 4 and 5, we can compare the performance of different combination methods and furthermore analyze the performance of different risk formulas being combined in M_1 in the single-fault scenario theoretically.

4.2. Properties of M_1 in the single-fault scenario

From Section 2.1, we know that each minimal hitting set is computed based on failed relevant dynamic slices (Wotawa, 2010; 2002). In practice, if the size of the set of failed relevant dynamic slices is very large, in consideration of the time limit and efficiency, a stop criterion is set in the algorithm of computing the minimal hitting sets (Greiner et al., 1989; Wotawa, 2010) and correspondingly not all the minimal hitting sets are computed. However, in our theoretical analysis, we need to consider all the possible minimal hitting sets.

Based on the definition of minimal hitting set in SENDYS, any statement in each minimal hitting set must appear in at least one failed relevant dynamic slice. However, omission faults which are related with missing code cannot be executed by any test execution and not covered by any failed relevant dynamic slice. Thus, omission faults which are related with missing code are not considered in our analysis.

First, we introduce some notions of minimal hitting sets. From Section 2, we have formally introduced the concept of the minimal hitting set and divided the minimal hitting sets into two types: (i) single-element minimal hitting set (denoted as H_S), which contains one and only one statement which is involved in all the dynamic slices; (ii) multiple-element minimal hitting set (denoted as H_M), which contains more than one statement. In our theoretical analysis, we denote the union of all the possible H_S as U_S and the union of all the possible H_M as U_M.

Lemma 1. *Given any statement $s_i \in U_S$, $M_1(s_i) = Pr(H_{S_i})$ where $Pr(H_{S_i})$ is the risk value of single-element hitting set H_{S_i} that consists of s_i, and $M_1(s_i)$ is the updated risk value of s_i computed by M_1.*

Proof. Given any statement $s_i \in U_S$, let H_{S_i} denote the single-element minimal hitting set including s_i. Referring to Algorithm 2 (M_1) in Section 3, the risk value of s_i is the sum of risk values of all the minimal hitting sets containing s_i. Hence, the risk value of s_i is the risk value of H_{S_i}. □

Then, we investigate the computation of risk values for single-element hitting sets. Referring to Algorithm 2 (M_1), for each H_S, we have $Pr(H_S) = R_{norm}(s_i) \times \prod_{s' \in PG \setminus H_S}(1 - R_{norm}(s'))$ where $s_i \in H_S$. We define such mathematical operation as τ (Hitting-set Mathematical Operation).

Definition 13. (τ). Given n real numbers $S = \{p_1, p_2, \ldots, p_n\}$ where $0 < p_i \le 1 (1 \le i \le n)$ and $\sum_{i=1}^{n} p_i = 1$, τ is defined as $p_i \times \prod_{j \ne i}(1 - p_j)$ where $1 \le i \le n$ and $1 \le j \le n$.

In the following, we can prove that τ is an order-preserving mathematical operation for the normalized statements ranking.

Lemma 2. *Given any normalized statements ranking γ, after applying τ on each statement in γ, the relative order of each statement in the ranking γ still remains the same.*

Proof. Given a program with n statements set $PG = \{s_1, s_2, \ldots, s_n\}$, the risk value of any statement s_i is denoted as $R(s_i)$, the statements in γ is ranked in the descending order of risk value. Given any two statements s_i and s_j where $1 \le i \le n$, $1 \le j \le n$ and s_j is ranked before s_i, we have $R(s_i) = R(s_j)$ or $R(s_i) < R(s_j)$. After applying operation τ on the two statements, we have $\tau(s_i) = R(s_i) \times (1 - R(s_j)) \times \prod_k (1 - R(s_k))$ and $\tau(s_j) = R(s_j) \times (1 - R(s_i)) \times \prod_k (1 - R(s_k))$ where $k \neq i$, $k \neq j$ and $1 \le k \le n$.

(a) For $R(s_i) = R(s_j)$. Immediately, we have $\tau(s_i) = \tau(s_j)$. If the tie-breaking scheme is consistent, after applying τ, statement s_j is still ranked before s_i.

(b) For $R(s_i) < R(s_j)$. We have $R(s_i) < R(s_j)$ and $(1 - R(s_j)) < (1 - R(s_i))$. $\prod_k (1 - R(s_k))$ is the common part of $\tau(s_i)$ and $\tau(s_j)$. Thus, $\tau(s_i) < \tau(s_j)$. After applying τ, statement s_j is still ranked before s_i.

In summary, after applying τ, s_j is still ranked before s_i. Hence, τ is an order-preserving operation for the statement ranking. □

4.3. Enhanced SENDYS in the single-fault scenario

When focusing on the single-fault scenario, the faulty statement s_f is covered by all failed test executions (Xie et al., 2013a). It is not difficult to conclude that s_f is covered by all failed relevant dynamic slices as well. Referring to the definition of hitting sets, U_S is the intersection of all the failed relevant dynamic slices. Hence, we can immediately obtain Lemma 3.

Lemma 3. *For the faulty statement s_f, we have $s_f \in U_S$.*

Referring to Lemma 3, we can observe that in M_1 the computation of risk values for multiple-fault diagnoses is unnecessary in the single-fault scenario. Hence, in our theoretical analysis, we only need to focus on the single-fault diagnoses. Accordingly, we propose an enhanced SENDYS algorithm in the single-fault scenario, namely M_2 which assigns the lowest risk values (negative or zero value) to each multiple-element hitting set. Following after Algorithm 2 (M_1), the risk value of any minimal hitting set must be positive, hence, the assignment of negative or zero value can be the lowest risk value. Algorithm 3 (M_2) illustrates the process of M_2.

Algorithm 3: Algorithm M_2.

Input: Program PG, risk formula $Formula$, minimal hitting sets H^s and test suite TS

Output: Statements ranking

1 Normalize the initial risk values with Algorithm 2 (M_1). Stop at Step.5 in Algorithm 2 (M_1)Compute the risk value for minimal hitting set H_i which is a single-element hitting set, $H_i = \{s_i\}$: $Pr_s(H_i) = R_{norm}(s_i) \times \prod_{s' \in PG \setminus H_i} (1 - R_{norm}(s'))$

2 Assigning zero risk value to each minimal hitting set H_i which is a multiple-element hitting set: $Pr_m(H_i) = 0$

3 Compute the final statements ranking with Algorithm 1 (SENDYS). Start with Step.5 in Algorithm 1 (SENDYS)

Since Lemmas 1 and 2 are properties of focusing on single-fault diagnoses, it is not difficult to find that the properties of M_1 in the single-fault scenario are also held for M_2.

4.4. Analysis of enhanced Sendys in the single-fault scenario

We use notation M^R to represent the fault localization method which are the integration of the risk formula R and the localization technique M. Notice that the following theoretical analysis and conclusions are in the single-fault scenario.

In the following, based on the generalized set theory based framework, referring to Definition 12, we divide statements ranking of M_1 and M_2 into three subsets respectively. Based on the divided subsets, we prove that given any risk formula R, M_2^R performs better than M_1^R and M_0^R (i.e. the original SBFL method without using any hitting set information). Moreover, we propose a simple formula conversion G which is proved to be equivalent to the process of Algorithm M_2. We conduct analysis on conversion G and compare G with conversion $C_{\max}$.

Let U_D^f denote the union of all the failed relevant dynamic slices. In M_1 and M_2, risk values returned by a risk formula will be updated based on the obtained minimal hitting sets. Referring to Algorithm 2 (M_1), only these statements belonging to U_D^f ($U_S \bigcup U_M$) are assigned to a new positive risk values while others are assigned with new risk values of 0. Therefore, statements outside U_D^f (i.e. $PG \setminus U_D^f$) having lower risk values than the ones within U_D^f belong to $S_A^{M_1}$. Referring to Algorithm 3 (M_2), not only the statements outside U_D^f, but also the statements in U_M are reassigned with the lowest value (zero value) because each H_M is assigned the lowest risk value 0 according to Step 3. On the other hand, statements in U_S are reassigned with new positive risk values. Thus statements outside U_S having lower risk values than the ones within U_S belong to $S_A^{M_2}$.

Notice that for both M_1 and M_2, the faulty statement is always reassigned with a positive value (but may not necessarily be the highest one in U_S). Therefore, we have the following statement subsets division for M_1 and M_2.

Corollary 1. *Given any risk formula R, let PG denote the program statements set. For M_1, we have the following three subsets*

$$S_B^{M_1} = \{s_i \mid M_1(s_i) > M_1(s_f) \text{ and } s_i \in U_D^f\}$$
$$S_F^{M_1} = \{s_i \mid M_1(s_i) = M_1(s_f) \text{ and } s_i \in U_D^f\}$$
$$S_A^{M_1} = \{s_i \mid (M_1(s_i) < M_1(s_f) \text{ and } s_i \in U_D^f) \text{ or } s_i \in PG \setminus U_D^f\}$$

Corollary 2. *Given any risk formula R, let PG denote the program statements set. For M_2, we have the following three subsets*

$$S_B^{M_2} = \{s_i \mid M_2(s_i) > M_2(s_f), \text{ and } s_i \in U_S\}$$
$$S_F^{M_2} = \{s_i \mid M_2(s_i) = M_2(s_f), \text{ and } s_i \in U_S\}$$
$$S_A^{M_2} = \{s_i \mid (M_2(s_i) < M_2(s_f), \text{ and } s_i \in U_S) \text{ or } s_i \in U_M \text{ or } s_i \in PG \setminus U_D^f\}$$

Based on the properties of Algorithm M_1 in the single-fault scenario (Lemmas 1 and 2), we can obtain Lemma 4 that is the basis upon which M_2^R performs better than M_0^R and M_1^R for any risk formula R (see Propositions 1 and 2).

Lemma 4. *Given any risk formula R, statements in U_S have the same relative order in M_0^R, M_1^R and M_2^R.*

Proof. For any two distinct statements s_k and s_j belonging to U_S, given any risk formula R, assume $R(s_j) \ge R(s_k)$. Obviously, we have $M_0^R(s_j) \ge M_0^R(s_k)$. Besides we have two single-element s_k and s_j constitute $H_{S_1} = \{s_k\}$ and $H_{S_2} = \{s_j\}$ respectively. Let $Pr(H_{S_1})$ and $Pr(H_{S_2})$ denote the risk values of H_{S_1} and H_{S_2} respectively. Referring to Lemma 1, we have $M_1^R(s_k) = Pr(H_{S_1})$ and $M_1^R(s_j) = Pr(H_{S_2})$, $M_2^R(s_k) = Pr(H_{S_1})$ and $M_2^R(s_j) = Pr(H_{S_2})$.

Referring to Lemma 2, after computing the risk value of each single-element minimal hitting set, we still have $Pr(H_{S_2}) \ge Pr(H_{S_1})$. Consequently, we have $M_1^R(s_j) \ge M_1^R(s_k)$ and $M_2^R(s_j) \ge M_2^R(s_k)$. If the tie-breaking scheme is consistent, s_j and s_k still have the same relative order in M_0^R, M_1^R and M_2^R.

From the above, we obtain that statements in U_S have the same relative order in M_0^R, M_1^R and M_2^R. □

Proposition 1. *For any given risk formula R, in the single-fault scenario, we have M_2^R (i.e. the enhanced SENDYS) → M_0^R (i.e. the original SBFL without using any hitting set information.).*

Proof. Given any risk evaluation formula R, for the convenience of illustration we use M_0 and M_2 to represent M_0^R and M_2^R respectively in this proof.

For M_0, referring to Definition 12, we have:

$$S_B^{M_0} = \{s_i \mid M_0(s_i) > M_0(s_f) \text{ and } s_i \in PG\}$$
$$S_F^{M_0} = \{s_i \mid M_0(s_i) = M_0(s_f) \text{ and } s_i \in PG\}$$
$$S_A^{M_0} = \{s_i \mid M_0(s_i) < M_0(s_f) \text{ and } s_i \in PG\}$$

For M_2, referring to Corollary 2, we have:

$$S_B^{M_2} = \{s_i \mid M_2(s_i) > M_2(s_f) \text{ and } s_i \in U_S\}$$
$$S_F^{M_2} = \{s_i \mid M_2(s_i) = M_2(s_f) \text{ and } s_i \in U_S\}$$
$$S_A^{M_2} = \{s_i \mid (M_2(s_i) < M_2(s_f) \text{ and } s_i \in U_S) \text{ or } s_i \in U_M \text{ or } s_i \in PG \setminus U_D^f\}$$

(a) To prove that $S_B^{M_2} \subseteq S_B^{M_0}$.
Assume statement $s_i \in S_B^{M_2}$. Referring to Lemma 4, the statements in U_S have the same relative order in both M_0 and M_2. Since $U_S \subset PG$, we have $S_B^{M_2} \subseteq S_B^{M_0}$.

(b) To prove that $S_A^{M_0} \subseteq S_A^{M_2}$.
$S_A^{M_0}$ and $S_A^{M_2}$ can be expressed using another form: $S_A^{M_0} = PG \setminus S_B^{M_0} \setminus S_F^{M_0}$; $S_A^{M_2} = PG \setminus S_B^{M_2} \setminus S_F^{M_2}$.
Similar to (a), we can also prove that $S_F^{M_2} \subseteq S_F^{M_0}$. Referring to (a), we have $S_B^{M_2} \subseteq S_B^{M_0}$, thus, $S_A^{M_0} \subseteq S_A^{M_2}$.

In conclusion, we have $S_B^{M_2} \subseteq S_B^{M_0}$ and $S_A^{M_0} \subseteq S_A^{M_2}$. Immediately after Theorem 4, $M_2^R \to M_0^R$. □

Proposition 2. *For any given risk formula R, in the single-fault scenario, we have $M_2^R \to M_1^R$.*

Proof. Given any risk evaluation formula R, for the convenience of illustration we use M_1 and M_2 to represent M_1^R and M_2^R respectively in this proof.

For M_1, referring to Corollary 1, we have:

$$S_B^{M_1} = \{s_i \mid M_1(s_i) > M_1(s_f) \text{ and } s_i \in U_D^f\}$$
$$S_F^{M_1} = \{s_i \mid M_1(s_i) = M_1(s_f) \text{ and } s_i \in U_D^f\}$$
$$S_A^{M_1} = \{s_i \mid (M_1(s_i) < M_1(s_f) \text{ and } s_i \in U_D^f) \text{ or } s_i \in PG \setminus U_D^f\}$$

For M_2, referring to Corollary 2, we have:

$$S_B^{M_2} = \{s_i \mid M_2(s_i) > M_2(s_f) \text{ and } s_i \in U_S\}$$
$$S_F^{M_2} = \{s_i \mid M_2(s_i) = M_2(s_f) \text{ and } s_i \in U_S\}$$
$$S_A^{M_2} = \{s_i \mid (M_2(s_i) < M_2(s_f) \text{ and } s_i \in U_S) \text{ or } s_i \in U_M \text{ or } s_i \in PG \setminus U_D^f\}$$

(a) To prove that $S_B^{M_2} \subseteq S_B^{M_1}$.
Since $U_S \subset U_D^f$, besides, referring to Lemma 4, we have the relative order of statements in U_S is the same in M_1 and M_2. Therefore, $S_B^{M_2} \subseteq S_B^{M_1}$.

(b) To prove that $S_A^{M_1} \subseteq S_A^{M_2}$.
From the above, we have

$$S_A^{M_1} = \{s_i \mid M_2(s_i) < M_2(s_f) \text{ and } s_i \in U_D^f\} \bigcup \{s_i \mid s_i \in PG \setminus U_D^f\}$$
$$S_A^{M_2} = \{s_i \mid M_2(s_i) < M_2(s_f) \text{ and } s_i \in U_S\} \bigcup \{s_i \mid s_i \in U_M\} \bigcup \{s_i \mid s_i \in PG \setminus U_D^f\}$$

Besides, referring to Lemma 4, the relative order of statements in U_S is the same in M_1 and M_2.
Hence, we have $S_A^{M_1} \subseteq S_A^{M_2}$.

In conclusion, we have $S_B^{M_2} \subseteq S_B^{M_1}$ and $S_A^{M_1} \subseteq S_A^{M_2}$. Immediately after Theorem 4, $M_2^R \to M_1^R$. □

Propositions 1 and 2 tell us that in single-fault scenario, M_2 (i.e. the enhanced SENDYS) never gives worse performance than M_1 (i.e. the correction of SENDYS) and M_0 (i.e. the original SBFL).

In our previous analysis on traditional SBFL, we provided a conversion (denoted as $C_{\max}$ in Definition 10). By applying $C_{\max}$ on any formula, we can transform this formula into a maximal one. We illustrated this conversion by visualizing the landscape of a formula in 3D figure (shown in Fig. 2), which has given the first explanation to maximal formulas in traditional SBFL (Yoo et al., 2017). Inspired by this analyzing method, we are wondering whether we can have a similar conversion on formulas, such that the original SBFL (i.e. M_0) with this transformed formula has equally good performance as M_2 that has shown definite superiority over other two methods. If there exists such a conversion, the complicated M_2 algorithm can be easily and vividly reformulated. By analyzing the properties of M_2, we found such a conversion, denoted as G and presented in Algorithm 4 .

Algorithm 4: Conversion G.

Input: Program PG, risk formula R, and test suite TS
Output: A transformed R (denoted as G^R)
1 Compute the matrix of relevant dynamic slices O_d for PG after the execution of TS
2 Denote the number of failed test cases as F, which is also the number of failed dynamic slices
3 Compute d_{ef}^i from O_d for each statement s_i
4 For each s_i with $d_{ef}^i = F$: define $G^R(s_i) = R(s_i)$
5 Get minimum risk value p of all s_i with $d_{ef}^i = F$: $p = Min(R(s_i))$ $(\forall s_i$ with $d_{ef}^i = F)$
6 For each s_i with $d_{ef}^i < F$: define $G^R(s_i)$ as a constant value lower than p

In Algorithm 4, we first compute O_d, which includes both passed and failed dynamic slices. From O_d, we can have F that is the number of failed dynamic slices, and d_{ef}^i that is the number of failed dynamic slices including statement s_i. Then, for each statement with $d_{ef}^i = F$, define the output of $G^R(s_i)$ as $R(s_i)$ (i.e. the original risk value computed by formula R); while for all the remaining statements, define their $G^R(s_i)$ outputs as a constant lower than p which is the minimum of risk values $R(s_i)$ among all s_i with $d_{ef}^i = F$.

Actually, this G conversion is very similar to $C_{\max}$ defined in Definition 10. The difference is that $C_{\max}$ only decreases the risk values of statements outside the faulty border (shown in Fig. 2), but G decreases the risk values of statements both outside the fault border and on the faulty border but having $d_{ef}^i < F$.

Given any risk formula R, we denote G^R as the transformed formula by using G conversion. Next we will prove that SBFL with

G^R has equally superiority as the complicated algorithm M_2. In other words, G actually reformulates M_2 by providing a short-cut to achieve the same goal.

In the following analysis, we will reuse the concepts of Faulty Border, RANKING and conversion $C_{\max}$ in Yoo et al. (2017) which are introduced in Section 2.2. Given any risk formula R, we use $M_0^{C_{\max}}$ to represent SBFL with the maximal version of any formula R transformed by $C_{\max}$.

Lemma 5. *Given any M_2^R where R belonging to $\mathbb{R}$, the points associated with any statement in U_S are on the faulty border.*

Proof. Referring to the definition of minimal hitting sets, we have U_S as the intersection of all the failed relevant dynamic slices. Hence, given any statement $s_i \in U_S$, we have s_i is executed by all the failed executions. As a consequence, we have $a_{ef} = F$. Therefore, the points associated with all the statements in U_S are on the faulty border. □

Lemma 6. *For any statement $s_i \in U_S$, let d^i_{ef} denote the number of failed relevant dynamic slices including s_i, we have $d^i_{ef} = F$.*

Proof. Referring to the definition of minimal hitting sets, we have U_S is the intersection of all failed relevant dynamic slices, hence for any statement $s_i \in U_S$, we have $d^i_{ef} = F$. □

Given any risk formula R, we denote G^R as the transformed formula by using G conversion. Because traditional SBFL is referred to as M_0 in this paper, we use $M_0^{G^R}$ to denote the SBFL configuring with the transformed formula G^R. Then, we have the following propositions.

Proposition 3. *For any given risk formula R, in the single-fault scenario, we have $M_0^{G^R} \leftrightarrow M_2^R$.*

Proof. Given any risk formula R, for the convenience of illustration, we use M_2 to denote M_2^R in the proof. Consider the points outside the faulty border and on the faulty border in G^R respectively.

For M_2, referring to Corollary 2, we have:

$$S_B^{M_2} = \{s_i \mid M_2(s_i) > M_2(s_f) \text{ and } s_i \in U_S\}$$
$$S_F^{M_2} = \{s_i \mid M_2(s_i) = M_2(s_f) \text{ and } s_i \in U_S\}$$
$$S_A^{M_2} = \{s_i \mid (M_2(s_i) < M_2(s_f) \text{ and } s_i \in U_S) \text{ or } s_i \in U_M \text{ or } s_i \in PG \setminus U_D^f\}$$

Referring to Algorithm 4 (conversion G), for the points outside the faulty border, risk values of these statements are always lower than that of faulty statement, thus these statements belong to S_A^G which denotes the set of statement having smaller risk values than that of the faulty statement in G^R. Notice that we can have the notations of S_B^G and S_F^G for G^R similarly. The corresponding d_{ef} values of these statements are smaller than F, referring to Lemma 6, we have these statements do not belong to U_S. Thus, these statements belong to $S_A^{M_2}$. Thus $S_A^{M_2} = S_A^G$.

For the points on the faulty border, in G^R, all these points are associated with statements having $d_{ef} = F$, referring to Lemma 6, we have the set of these statements is equal to U_S. These statements have the same relative order in M_0^R and G^R. Referring to Lemma 4, the relative order of these statements is the same in M_0^R and M_2^R. Thus, in both M_2^R and G^R the relative order of these statements is the same. If the tie-breaking is consistent, we can have $S_B^{M_2} = S_B^G$, $S_F^{M_2} = S_F^G$.

Following immediately from Theorem 5, we have proved $M_0^{G^R} \leftrightarrow M_2^R$. □

Referring to Proposition 3, we prove that given any risk formula R, M_2^R is equivalent to the variant of formula R after applying conversion G. Besides, from Propositions 1 and 2, we have that for any risk formula R, M_2^R performs better than M_1^R and M_0^R. It indicates that in the single-fault scenario the updated risk values of statements through the computation of single-element minimal hitting set in Algorithm 3 can be substituted for a simple formula conversion G. Hence, in the following, we focus on the analysis of conversion G. Through investigating different risk formulas being applied conversion G, we have the following proposition.

Proposition 4. *For any two risk formulas R_1 and R_2, in the single-fault scenario, if R_1 and R_2 have the same RANKING, then $M_0^{G^{R1}} \leftrightarrow M_0^{G^{R2}}$.*

Proof. Firstly, we prove that $M_2^{R_1} \leftrightarrow M_2^{R_2}$ is held.

Let s_f denote the faulty statement and (F, a^f_{ep}) denote the corresponding point on the faulty border. Referring to Lemma 4, these statements in U_S have the same relative order in both R_1 and $M_2^{R_1}$, R_2 and $M_2^{R_2}$. From the known condition, we have R_1 and R_2 have the same RANKING, thus, $M_2^{R_1}$ and $M_2^{R_2}$ also have the same RANKING. We use M_2^1 to denote $M_2^{R_1}$ and M_2^2 to denote $M_2^{R_2}$ for illustrating conveniently.

For M_2^1, referring to Corollary 2, we have:

$$S_B^{M_2^1} = \{s_i \mid M_2^1(s_i) > M_2^1(s_f) \text{ and } s_i \in U_S\}$$
$$S_F^{M_2^1} = \{s_i \mid M_2^1(s_i) = M_2^1(s_f) \text{ and } s_i \in U_S\}$$
$$S_A^{M_2^1} = \{s_i \mid (M_2^1(s_i) < M_2^1(s_f) \text{ and } s_i \in U_S) \text{ or } s_i \in U_M \text{ or } s_i \in PG \setminus U_D^f\}$$

For M_2^2, referring to Corollary 2, we have:

$$S_B^{M_2^2} = \{s_i \mid M_2^2(s_i) > M_2^2(s_f) \text{ and } s_i \in U_S\}$$
$$S_F^{M_2^2} = \{s_i \mid M_2^2(s_i) = M_2^2(s_f) \text{ and } s_i \in U_S\}$$
$$S_A^{M_2^2} = \{s_i \mid (M_2^2(s_i) < M_2^2(s_f) \text{ and } s_i \in U_S) \text{ or } s_i \in U_M \text{ or } s_i \in PG \setminus U_D^f\}$$

(a) To prove that $S_B^{M_2^1} = S_B^{M_2^2}$.

Assume statement $s_i \in S_B^{M_2^1}$. Referring to Lemma 5, we have the point associated with s_i is on the faulty border which is denoted as (F, a^i_{ep}). Whatever $a^i_{ep} > a^f_{ep}$ and $a^i_{ep} < a^f_{ep}$, the risk value of the point associated with s_i is larger than that of the point associated with s_f for $M_2^{R_1}$. Besides, $M_2^{R_1}$ and $M_2^{R_2}$ have the same RANKING, then, for $M_2^{R_2}$, the risk value of s_i is still higher than that of s_f. Thus, we have $s_i \in S_B^{M_2^2}$. Therefore, $S_B^{M_2^1} \subseteq S_B^{M_2^2}$. In a similar way, we can prove that $S_B^{M_2^2} \subseteq S_B^{M_2^1}$.

In summary, we have proved $S_B^{M_2^1} = S_B^{M_2^2}$.

(b) To prove that $S_F^{M_2^1} = S_F^{M_2^2}$.

Assume statement $s_i \in S_F^{M_2^1}$, referring to Lemma 5, we have the point associated with s_i is on the faulty border which is denoted as (F, a^i_{ep}). Whatever $a^i_{ep} > a^f_{ep}$ and $a^i_{ep} < a^f_{ep}$, the risk value of the point associated with s_i is equal to that of the point associated with s_f for $M_2^{R_1}$. Besides, $M_2^{R_1}$ and $M_2^{R_2}$ have the same RANKING, if the tie-breaking is consistent, the relative order of s_i and s_f is still the same for $M_2^{R_2}$. Thus, we have $s_i \in S_F^{M_2^2}$. Therefore, $S_F^{M_2^1} \subseteq S_F^{M_2^2}$. In a similar way, we can prove that $S_F^{M_2^2} \subseteq S_F^{M_2^1}$.

In summary, we have proved $S_F^{M_2^1} = S_F^{M_2^2}$.

From the above, it is not difficult to get $S_A^{M_2^1} = S_A^{M_2^2}$. Following immediately Theorem 5, we have proved $M_2^{R_1} \leftrightarrow M_2^{R_2}$. Finally, referring to Proposition 3, we have $M_0^{G^{R1}} \leftrightarrow M_0^{G^{R2}}$. □

Proposition 4 provides a sufficient condition for the equivalence of any two formulas being applied conversion G. We take the 34 risk formulas investigated in Xie et al. (2013a,b) as examples to illustrate Proposition 4: (1) Formulas that are not from *ER*5, *GP*2, *GP*3 and *GP*19 have the same Ranking: given any two points on the faulty border which are denoted as (F, a_{ep}^i) and (F, a_{ep}^j) respectively, if $a_{ep}^i < a_{ep}^j$, then the risk value of (F, a_{ep}^i) is higher that that of (F, a_{ep}^j). We refer to such Ranking as descending Ranking. Given any R_1 and R_2 having decreasing Ranking, G^{R_1} and G^{R_2} are equivalent to each other. (2) Formulas from *ER*5 have the same Ranking: the risk values of any two points on the faulty border are equal what we refer to as equal Ranking. Given any two formulas having equal Ranking, G^{R_1} and G^{R_2} are equivalent to each other. (3) For any formula having the same Ranking with *GP*3: given any two points on the faulty border (F, a_{ep}^i) and (F, a_{ep}^j), if $a_{ep}^i < a_{ep}^j$, then the risk value of (F, a_{ep}^i) is lower that that of (F, a_{ep}^j). Such Ranking is referred to as ascending Ranking. Given any R_1 and R_2 having ascending Ranking, G^{R_1} and G^{R_2} are equivalent to each other. And for *GP*2 and *GP*19, M_2^{GP2} and G^{GP19} have distinct performance.

From Proposition 4, it is not difficult to find that the formulas which have the same Ranking after applying conversion G constitute their own maximal groups.

So far, we have proved that the after reformulating M_2^R with a more simple $M_0^{G^R}$, the fault localization performance is preserved, in single-fault scenario. Moreover, as long as two formulas have the same monotonicity about a_{ep} on the faulty border (i.e. the same Ranking according to Definition 9), M_2 with these two formulas have equivalent performance, and transformations of these two formulas via G will also deliver equivalent performance in SBFL.

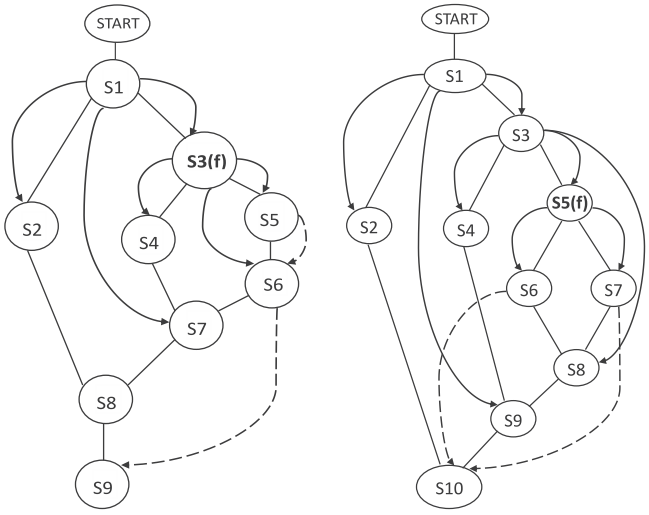
In the following discussion, we will compare the performance between the transformed formula G^R and the traditional maximal formula proved in Yoo et al. (2017). Due to the equivalence between $M_0^{G^R}$ and M_2^R, this comparison also reveals the relation between the enhanced Sendys (i.e. M_2) and traditional SBFL with maximal formulas. Given any formula R, denote G^R and $C_{\max}^R$ as the transformed formula via G and $C_{\max}$, respectively. According to the proof in Yoo et al. (2017), the latter one is maximal formula in traditional SBFL. We denote $M_0^{C_{\max}}$ as traditional SBFL working on execution slice with maximal formula $C_{\max}^R$.

Proposition 5. *For any given risk formula R, in the single-fault scenario, we have* $M_0^{G^R}(M_2^R) \rightarrow M_0^{C_{\max}}$.

Proof. Referring to Algorithm 4, we have that after applying conversion G on R, points associated with statements having $a_{ef} = F$ and $d_{ef} \neq F$ which have higher risk values than that of the faulty statement can be removed. Thus, we can obtain that $M_0^{G^R}(M_2^R) \rightarrow M_0^{C_{\max}}$. □

As discussed after Algorithm 4, G further removes noises on the faulty border left by $C_{\max}$. That is, any point associated with statements having $a_{ef} = F$ but $d_{ef} < F$ is assigned with a risk value lower than that of the faulty statement. Such statements cannot be faulty in the single-fault scenario. As a reminder, to what extent $M_0^{G^R}$ can improve $M_0^{C_{\max}}$ depends on the number of statements with $d_{ef} < F$ and $a_{ef} = F$, which varies in different scenarios.

Notice that in the above analysis, $M_0^{G^R}$ and $M_0^{C_{\max}}$ are applied on execution slice. It is known that M_0 (i.e. traditional SBFL) can be applied on various types of spectra. For example, Lei et al. (2012) showed that by applying M_0 on dynamic slices,



(a) PG_1 (b) PG_2

Fig. 3. Control flow graphs with dependency.

Table 1
TS_1 executed on PG_1 (Scenario 1).

Pass or Fail	Execution slices	Dynamic slices
Fail	$(S_1, S_3, S_5, S_6, S_7, S_8, S_9)$	$(S_1, S_3, S_5, S_6, S_9)$
Fail	$(S_1, S_3, S_5, S_6, S_7, S_8, S_9)$	$(S_1, S_3, S_5, S_6, S_9)$
Pass	(S_1, S_2, S_8, S_9)	(S_9)
Pass	$(S_1, S_3, S_5, S_6, S_7, S_8, S_9)$	$(S_1, S_3, S_5, S_6, S_9)$
Pass	$(S_1, S_3, S_4, S_7, S_8, S_9)$	(S_9)
Pass	$(S_1, S_3, S_4, S_7, S_8, S_9)$	(S_9)

the fault localization performance can be improved. For each statement s_i, the results of relevant dynamic slices can be represented as a tuple $d =< d_{ef}^i, d_{ep}^i, d_{nf}^i, d_{np}^i >$, where d_{ef}^i and d_{ep}^i represent the number of failed and passed relevant dynamic slices including s_i, respectively; d_{nf}^i and d_{np}^i represent the number of failed and passed relevant dynamic slices not including s_i, respectively. Given any risk formula R, we denote the $C_{\max}$ version of R applied on relevant dynamic slices as $\widetilde{M_0^{C_{\max}}}$. In the following, we are interested in the comparison of performance between $M_0^{G^R}$ and $\widetilde{M_0^{C_{\max}}}$.

Proposition 6. *For any given risk formula R, in the single-fault scenario, we have* $M_0^{G^R}(M_2^R) \nleftrightarrow \widetilde{M_0^{C_{\max}}}$.

Proof. We prove this proposition by constructing a counter-example to show neither $M_0^{G^R}(M_2^R) \rightarrow \widetilde{M_0^{C_{\max}}}$ nor $\widetilde{M_0^{C_{\max}}} \rightarrow M_0^{G^R}(M_2^R)$ holds.

Let us consider the two sample programs PG_1 and PG_2, whose control flow graphs with dependency information are shown in Fig. 3. In Fig. 3, the dashed arrows represent data dependency, the solid arrows represent control dependency, and the solid lines indicate control flow relations. The faulty statements of PG_1 and PG_2 are S_3 and S_5 respectively.

- *Scenario 1:* Suppose we have a test suite TS_1 execute on PG_1, whose results are shown in Table 1. The testing results include two failed test cases and four passed test cases (i.e. $F = 2$). The execution slices and dynamic slices (by considering both control and data dependency) are listed in the second and third columns, respectively. Accordingly, we have $(a_{ef}, a_{ep}, a_{nf}, a_{np})$ and $(d_{ef}, d_{ep}, d_{nf}, d_{np})$ shown in Table 2.
  Suppose we adopt CBI as a sample formula (Xie et al., 2013a), which should be $\frac{a_{ef}}{a_{ef}+a_{ep}} - \frac{a_{ef}+a_{nf}}{a_{ef}+a_{nf}+a_{ep}+a_{np}}$ for M_0 (that uses exe-

Table 2
$M_0^{G^R}(M_2^R) \nrightarrow \widetilde{M_0^{C_{max}}}$ (Scenario 1) .

Statements of PG_1	$(a_{ef}, a_{ep}, a_{nf}, a_{np})$	$(d_{ef}, d_{ep}, d_{nf}, d_{np})$	$M_0^{G^R}$	$\widetilde{M_0^{C_{max}}}$	(M_2^R)
S_1	(2,4,0,0)	(2,1,0,3)	0.0	0.333	0.112
S_2	(0,1,2,3)	(0,0,2,4)	−0.333	−0.333	0.0
$S_3(f)$	(2,3,0,1)	(2,1,0,3)	0.067	0.333	0.143
S_4	(0,2,2,2)	(0,0,2,4)	−0.333	−0.333	0.0
S_5	(2,1,0,3)	(2,1,0,3)	0.333	0.333	0.317
S_6	(2,1,0,3)	(2,1,0,3)	0.333	0.333	0.317
S_7	(2,3,0,1)	(0,0,2,4)	−0.333	−0.333	0.0
S_8	(2,4,0,0)	(0,0,2,4)	−0.333	−0.333	0.0
S_9	(2,4,0,0)	(2,4,0,0)	0.0	0.0	0.112
Ranking			3	2	3

Table 3
TS_2 for PG_2 (Scenario 2).

Pass or Fail	Execution slices	Dynamic slices
Fail	$(S_1, S_3, S_5, S_6, S_8, S_9, S_{10})$	$(S_1, S_3, S_5, S_6, S_{10})$
Fail	$(S_1, S_3, S_5, S_7, S_8, S_9, S_{10})$	$(S_1, S_3, S_5, S_7, S_{10})$
Pass	(S_1, S_2, S_{10})	(S_{10})
Pass	$(S_1, S_3, S_4, S_9, S_{10})$	(S_{10})
Pass	$(S_1, S_3, S_5, S_6, S_8, S_9, S_{10})$	$(S_1, S_3, S_5, S_6, S_{10})$
Pass	$(S_1, S_3, S_5, S_7, S_8, S_9, S_{10})$	$(S_1, S_3, S_5, S_7, S_{10})$
Pass	$(S_1, S_3, S_4, S_9, S_{10})$	(S_{10})
Pass	$(S_1, S_3, S_5, S_7, S_8, S_9, S_{10})$	$(S_1, S_3, S_5, S_7, S_{10})$

cution slices) and $\frac{d_{ef}}{d_{ef}+d_{ep}} - \frac{d_{ef}+d_{nf}}{d_{ef}+d_{nf}+d_{ep}+d_{np}}$ for $\widetilde{M_0}$ (that uses dynamic slices). As a consequence, we have $M_0^{G^R}$ assign risk values to statements with d_{ef}=2 as $\frac{a_{ef}}{a_{ef}+a_{ep}} - \frac{a_{ef}+a_{nf}}{a_{ef}+a_{nf}+a_{ep}+a_{np}}$, and assign "min_G - constant" to the remaining statements (where "min_G" is the minimum among all the above risk values with d_{ef}=2 in $M_0^{G^R}$). Similarly, we have $\widetilde{M_0^{C_{max}}}$ assign risk values to statements with d_{ef}=2 as $\frac{d_{ef}}{d_{ef}+d_{ep}} - \frac{d_{ef}+d_{nf}}{d_{ef}+d_{nf}+d_{ep}+d_{np}}$, and assign "min_C - constant" to the remaining statements (where "min_C" is the minimum among all the risk values with d_{ef}=2 in $\widetilde{M_0^{C_{max}}}$).[1]

Accordingly, the risk values are shown in Table 2. By adopting the "*Original Order*" tie-breaking scheme (Xie et al., 2013a), we have the ranking shown in Table 2, which demonstrates the possibility of "$\widetilde{M_0^{C_{max}}}$ performing better than $M_0^{G^R}$". As a reminder, we also list the risk values and ranking given by M_2^R for illustrating the equivalence between M_2^R and $M_0^{G^R}$. But we do not repeat the calculation of M_2^R here, since it is not necessary to this proof, and the equivalence between M_2^R and $M_0^{G^R}$ has been theoretically proved in Proposition 3.

In summary, we have proved $M_0^{G^R}(M_2^R) \nrightarrow \widetilde{M_0^{C_{max}}}$.

- *Scenario 2:* Suppose we have a test suite TS_2 execute on PG_2, whose results are shown in Table 3. The testing results include two failed test cases and six passed test cases (i.e. $F = 2$). The execution slices and dynamic slices (by considering both control and data dependency) are listed in the second and third columns, respectively. Accordingly, we have $(a_{ef}, a_{ep}, a_{nf}, a_{np})$ and $(d_{ef}, d_{ep}, d_{nf}, d_{np})$ shown in Table 4.

  Again, suppose we adopt CBI as illustration. The risk values are shown in Table 4.[2] By adopting the "*Original Order*" tie-breaking scheme, we have the ranking shown in Table 4, which demonstrates the possibility of "$M_0^{G^R}$ outperforming $\widetilde{M_0^{C_{max}}}$". Thus, we have proved $\widetilde{M_0^{C_{max}}} \nrightarrow M_0^{G^R}(M_2^R)$.

From the above two examples, we can have Thus, we have $M_0^{G^R}(M_2^R) \nleftrightarrow \widetilde{M_0^{C_{max}}}$. □

Up to now, we have proved the superiority of $M_0^{G^R}$ (and M_2^R) over the maximal formula in traditional SBFL with execution slices.

5. Empirical studies

On one hand, theoretical analysis reveals some definite properties among different fault localization methods, indicating which one is better. On the other hand, we also want to learn more quantitative details about the superiority of a particular method. Therefore, we conduct experimental studies to complement our theoretical analysis.

5.1. Research questions

In our experiment, we aim to compare SBFL (i.e. M_0), the original Sendys (this is referred to as Sendys for short in the following discussion), M_1, and M_2. As a reminder, since we have theoretically proved the equivalence between M_2 and M_0^G, analyzing M_2 is essentially the same as analyzing M_0^G. Specifically, we will answer the following three research questions.

- *RQ1:* How do the original SBFL (denoted as M_0), Sendys, M_1, and M_2 (i.e. M_0^G) compare with each other, in single-fault scenario?
  According to our analysis, there is no definite performance relation among M_0, Sendys, and M_1. Thus, we are interested in the following comparisons: "(Sendys v.s M_0)", "(M_1 v.s M_0)" and "(M_1 v.s Sendys)". Besides, since we have proved that M_2 outperforms M_0 and M_1, we are interested to know the quantitative advantage of M_2 in practice. Thus, we have the comparisons of "(M_2 v.s M_1)", "(M_2 v.s Sendys)" and "(M_2 v.s M_0)".
- *RQ2:* How do different risk formulas affect the above comparisons?
  As discussed above, M_0, Sendys, M_1 and M_2 can be configured with various risk formulas. Thus, we will investigate whether the comparison results obtained in RQ1 differ across different formulas.
- *RQ3:* How do M_0, Sendys, and M_1 compare with each other, in terms of locating multiple faults simultaneously?
  Apart from locating single fault, we will investigate cases where these methods locate multiple faults simultaneously. Specifically, we will consider two-fault, three-fault and four-fault scenarios. As a reminder, this analysis will not involve M_2. According to the third line of Algorithm 3, M_2 assigns zero risk value to each minimal hitting set which contains multiple-element

[1] According to the definition, the constant can be any value. In Table 2, we have constant as 0.333.

[2] We have the constant as 0.15 in Table 4.

Table 4
$M_0^{C_{max}} \nrightarrow M_0^{C^R}(M_2^R)$ (Scenario 2) .

Statements of PG_1	$(a_{ef}, a_{ep}, a_{nf}, a_{np})$	$(d_{ef}, d_{ep}, d_{nf}, d_{np})$	$M_0^{C^R}$	$\widetilde{M_0^{C_{max}}}$	(M_2^R)
S_1	(2,6,0,0)	(2,3,0,3)	0.0	0.15	0.146
S_2	(0,1,2,5)	(0,0,2,6)	−0.15	−0.15	0.0
S_3	(2,5,0,1)	(2,3,0,3)	0.036	0.15	0.173
S_4	(0,2,2,4)	(0,0,2,6)	-0.15	−0.15	0.0
$S_5(f)$	(2,3,0,3)	(2,3,0,3)	0.15	0.15	0.272
S_6	(1,1,1,5)	(1,1,1,5)	−0.15	−0.15	0.0
S_7	(1,2,1,4)	(1,2,1,4)	−0.15	−0.15	0.0
S_8	(2,3,0,3)	(0,0,2,6)	−0.15	−0.15	0.0
S_9	(2,5,0,1)	(0,0,2,6)	−0.15	−0.15	0.0
S_{10}	(2,6,0,0)	(2,6,0,0)	0.0	0.0	0.146
Ranking			1	3	1

Table 5
Description of investigated programs.

Programs	Faulty versions	LOC	Test cases	Description
JTopas v1	1	1368	127	Text data parser
JTopas v2	1	1485	115	
JTopas v3	3	3931	183	
Jtcas	24	77	1545	Altitude separation
Atms	2	1573	14	Assumption-based truth maintenance system
Reflec. Visitor	5	338	14	Implementation of visitor-pattern
Traffic Light	2	33	7	Simulation system of traffic light
Static Example	1	16	8	Demo program with static members and methods
Mid	1	17	8	Demo program returning median of three numbers
Bank Account	1	17	5	Demo program simulating a bank account
NanoXML	5	7646	100	XML parser for Java
MultiViewer	9	13,321	100	Change impact analysis tool for GitHub
HybridTag	7	16,729	100	Tagging system for GitHub issue
Total faulty versions	**62**			

hitting. As a consequence, M_2 is not suitable to locate multiple faults simultaneously, and hence is not considered in this research question.

5.2. Experimental set-up

In our experiments, we adopted 10 small-scale programs from Hofer and Wotawa (2012), one middle-scale program from http://nanoxml.sourceforge.net/orig/index.html, and two large-scale programs from our previous works (Wang et al., 2017; Wang, 2018). Information about these programs are shown in Table 5. Jtcas is a java implementation of Tcas. Jtcas and JTopas are obtained from Software-artifact Infrastructure Repository (SIR) website.[3] Traffic Light program is from Jade project.[4] Programs Bank Account, Mid and Static Example are toy examples. We obtain Atms and Reflec. Visitor and these three toy programs from the author of Hofer and Wotawa (2012). NanoXML is a small non-validating parser for Java. MultiViewer is a tool in Java that evaluates and visualizes change impacts and change coupling relations for GitHub commits (Wang et al., 2017). HybridTag is automatic tagging system in Java for issues in GitHub (Wang, 2018).

In our experiments, we adopted the faulty versions for the 10 small-scale programs from Hofer and Wotawa (2012) and randomly seeded 10 faults to each of the remaining three programs. Then, we excluded versions for which (i) no slices can be computed for faulty program versions; (ii) the incorrect slices computation for faulty program versions due to the limitation of the slicer in use; (iii) program faulty versions containing faults which are not in predicate or assignment statements. Finally, we have 62 faulty program versions in total.

[3] http://sir.unl.edu/php/index.php
[4] http://www.dbai.tuwien.ac.at/proj/Jade/

Table 6
Number of versions for multiple-fault scenarios.

Programs	2-fault	3-fault	4-fault
JTopas v3	3	1	0
Jtcas	10	10	10
Atms	1	0	0
Reflec. Visitor	10	10	5
Traffic Light	1	0	0
NanoXML	10	10	5
MultiViewer	10	10	10
HybridTag	10	10	10
Total	55	51	40

To investigate multiple-fault scenario, we combine faults of one program to construct two-fault, three-fault and four-fault versions. As a reminder, we excluded programs with only one fault. From each of the remaining programs with n faults ($n > 1$), for "i-fault" scenario, we can construct $C(n, i)$ versions. To avoid too many versions, we set the upper bound of the version number for each program and each "i-fault" scenario as 10. As a consequence, we have the numbers of multiple-fault program versions shown in Table 6.

To address RQ2, we select 34 risk formulas which have been comprehensively investigated (Xie et al., 2013a; Yoo et al., 2017). Due to the limit of the space, we have listed all of their definitions in Section Appendix A.

We adopt "*Expense*" in Definition 2 as the measure of fault localization performance for single-fault case. For multiple-fault scenarios, we use *ExpenseM* to measure the effectiveness of a method that simultaneously locates faults, which is defined in Definition 14.

Definition 14 (ExpenseM)**.** Consider a program with n statements, which contain k faults $(s_1, s_2, \ldots, s_k)$. Suppose a method M_i configured with formula R gives rankings to the k faults as $r_1, r_2, \ldots, r_k$,

Table 7
Comparison for single-fault scenario.

Formula R	(SENDYS, M_0)			(M_1, M_0)			(M_1, SENDYS)			(M_2, M_1)			(M_2, SENDYS)			(M_2, M_0)		
	E	B	W	E	B	W	E	B	W	E	B	W	E	B	W	E	B	W
GP13	26	36	0	26	36	0	62	0	0	62	0	0	62	0	0	26	36	0
Naish1	3	3	56	26	36	0	3	59	0	62	0	0	3	59	0	26	36	0
Naish2	2	5	55	26	36	0	5	57	0	62	0	0	5	57	0	26	36	0
Kulczynski2	21	37	4	21	37	4	62	0	0	57	5	0	57	5	0	21	41	0
Wong3	2	8	52	18	42	2	9	53	0	44	18	0	3	59	0	17	45	0
M2	21	41	0	21	41	0	62	0	0	61	1	0	61	1	0	21	41	0
Ochiai	12	50	0	12	50	0	62	0	0	61	1	0	61	1	0	12	50	0
GP3	14	45	3	13	46	3	61	1	0	56	6	0	55	7	0	16	46	0
AMPLE2	5	12	45	13	49	0	5	57	0	62	0	0	5	57	0	13	49	0
Anderberg	7	55	0	7	55	0	62	0	0	62	0	0	62	0	0	7	55	0
Dice	7	55	0	7	55	0	62	0	0	62	0	0	62	0	0	7	55	0
Goodman	4	16	42	7	55	0	9	53	0	62	0	0	9	53	0	7	55	0
Jaccard	7	55	0	7	55	0	62	0	0	62	0	0	62	0	0	7	55	0
Sørensen-Dice	7	55	0	7	55	0	62	0	0	62	0	0	62	0	0	7	55	0
Arithmetic Mean	5	11	46	9	53	0	4	58	0	61	1	0	4	58	0	9	53	0
Cohen	4	12	46	6	55	1	5	57	0	59	3	0	4	58	0	6	56	0
GP2	16	45	1	16	45	1	62	0	0	56	6	0	56	6	0	17	45	0
CBI	4	14	44	5	57	0	6	56	0	44	18	0	2	60	0	3	59	0
q_e	5	57	0	5	57	0	62	0	0	44	18	0	44	18	0	3	59	0
Tarantula	5	57	0	5	57	0	62	0	0	59	3	0	59	3	0	5	57	0
GP19	6	56	0	6	56	0	62	0	0	61	1	0	61	1	0	6	56	0
Fleiss	1	17	44	7	53	2	6	56	0	34	28	0	2	60	0	1	61	0
Rogot1	7	55	0	7	55	0	62	0	0	42	20	0	42	20	0	0	62	0
Scott	1	17	44	7	54	1	6	56	0	34	28	0	2	60	0	0	62	0
Euclid	7	55	0	7	55	0	62	0	0	44	18	0	44	18	0	0	62	0
Hamann	0	17	45	5	56	1	3	59	0	33	29	0	0	62	0	0	62	0
Hamming	7	55	0	7	55	0	62	0	0	41	21	0	41	21	0	0	62	0
Rogers&Tanimoto	7	55	0	7	55	0	62	0	0	41	21	0	41	21	0	0	62	0
Simple Matching	7	55	0	7	55	0	62	0	0	41	21	0	41	21	0	0	62	0
Sokal	7	55	0	7	55	0	62	0	0	41	21	0	41	21	0	0	62	0
Wong2	0	18	44	5	56	1	3	59	0	35	27	0	0	62	0	0	62	0
Binary	1	61	0	1	61	0	62	0	0	62	0	0	62	0	0	1	61	0
Russel&Rao	1	61	0	1	61	0	62	0	0	62	0	0	62	0	0	1	61	0
Wong1	1	61	0	1	61	0	62	0	0	62	0	0	62	0	0	1	61	0
Total Sum	230	1307	571	332	1760	16	1427	681	0	1793	315	0	1239	869	0	266	1842	0

respectively. ExpenseM is the average Expense of all the k faults, that is, $(\sum_{i=1}^{i=k}(\frac{r_i}{n}))/k$.

5.3. Experimental results

5.3.1. To address RQ1

Results for the six comparisons "(SENDYS v.s M_0)", "(M_1 v.s M_0)", "(M_1 v.s SENDYS)", '(M_2 v.s M_1)", "(M_2 v.s SENDYS)" and "(M_2 v.s M_0)" over all the 62 single-fault versions are shown in Table 7. In this table, under the column of (M_i, M_j), Column "E" indicates the number of faulty versions where these two methods are observed to give "equal *Expense*" value, Column "B" indicates the number of faulty versions where M_i gives "better *Expense*" (i.e. lower *Expense*) value than M_j, and Column "W" means the number of faulty versions where M_i gives "worse *Expense*" (i.e. higher *Expense*) value than M_j. For example, let us consider column (SENDYS, M_0) on the row of *Kulczynski2*, which means both these two methods are configured with formula *Kulczynski2*. Among the 62 faulty versions, we observed 37 versions where SENDYS gives better *Expense* than M_0, 4 versions where SENDYS gives worse *Expense* than M_0, and 21 versions where these two methods have the same performance.

Given a method of M_0, SENDYS, M_1 or M_2, we denote a fault localization with one formula on one faulty version as an experimental trial. Then for each method, there are 62×34 trials in single-fault case. The bottom line of Table 7 sums up the total number of experimental trials, under the corresponding comparison result.

Columns 2 to 5 of Table 8 present the average "*Expense*" value over the 62 single-fault program versions for one fault localization algorithm (i.e. M_0, SENDYS, M_1 or M_2) configured with one risk formula. As a reminder, to better illustrate our observations, we have ordered the formulas according to their average *Expense* on M_0 ascendingly, and Table 7 adopts the same order. From Columns 6 to 12 of this table, "Reduced *Expense*" for column (M_i, M_j) and formula R is equal to "the average *Expense* of M_j for R" minus "the average *Expense* of M_i for R". Positive values mean that M_i delivers better results than M_j; negative values mean that M_i delivers worse results than M_j; and "zero" means M_i returns the same result as M_j. For simplicity, we omit the "%" in each cell of this table.

For example, the cell for (M_0, GP13) is 26.5 (i.e. 26.5%), which is the average *Expense* for the experimental trials with M_0 and GP13 on all the 62 faulty versions. And the reduced percentage under "(SENDYS, M_0)" for GP13 is 5.5%, because the average *Expense* for M_0 is 26.5% and for SENDYS is 21.0%. Thus we have $26.5\% - 21.0\% = 5.5\%$. The bottom line of Table 8 gives the average of *Expense* over all the 62×34 trials.

From Tables 7 and 8 we can address RQ1.

1. Though SENDYS does not always outperform M_0, it still demonstrates superiority over M_0. From Table 7 we know that among all the 62×34 trials, SENDYS delivers better results than M_0 for 1307 times, equal results to M_0 for 230 times and worse results than M_0 for 571 times. However, from the bottom line of Table 8, we observe that in terms of the average *Expense* over all the experimental trials, the advantage of SENDYS seems to be marginal (i.e. reduced *Expense* is only 0.6%). By checking the formulas line by line, we learn that SENDYS gives much better results than M_0 in many cases, but the reduced *Expense* is offset by the unstable performance of SENDYS. We will discuss how the comparison varies on different formulas in RQ2.
2. Though from the analysis in Section 4, we did not find any definite theoretical relation between M_1, M_0, we observed in our

Table 8
Average *Expense* for single-fault scenario.

Formula R	Average *expense* (%)				Reduced *expense* (%)					
	M_0	Sendys	M_1	M_2	(Sendys, M_0)	(M_1, M_0)	(M_1, Sendys)	(M_2, M_1)	(M_2, Sendys)	(M_2, M_0)
GP13	26.5	21.0	21.0	21.0	5.5	5.5	0.0	0.0	0.0	5.5
Naish1	26.5	58.0	21.0	21.0	−31.5	5.5	37.0	0.0	37.0	5.5
Naish2	26.5	57.0	21.0	21.0	−30.5	5.5	36.0	0.0	36.0	5.5
Kulczynski2	27.2	21.2	21.2	21.0	6.0	6.0	0.0	0.2	0.2	6.2
Wong3	27.7	55.8	21.5	21.0	−28.2	6.1	34.3	0.6	34.9	6.7
M2	28.5	21.0	21.0	21.0	7.6	7.6	0.0	0.0	0.0	7.6
Ochiai	29.7	21.0	21.0	21.0	8.8	8.8	0.0	0.0	0.0	8.8
GP3	31.2	22.5	21.6	21.5	8.7	9.6	0.9	0.1	0.9	9.7
AMPLE2	31.9	57.2	21.0	21.0	−25.2	11.0	36.2	0.0	36.2	11.0
Anderberg	32.6	21.0	21.0	21.0	11.6	11.6	0.0	0.0	0.0	11.6
Dice	32.6	21.0	21.0	21.0	11.6	11.6	0.0	0.0	0.0	11.6
Goodman	32.6	54.0	21.0	21.0	−21.4	11.6	33.0	0.0	33.0	11.6
Jaccard	32.6	21.0	21.0	21.0	11.6	11.6	0.0	0.0	0.0	11.6
Sørensen-Dice	32.6	21.0	21.0	21.0	11.6	11.6	0.0	0.0	0.0	11.6
Arithmetic Mean	32.6	57.5	21.0	21.0	−24.9	11.6	36.6	0.0	36.6	11.6
Cohen	32.8	57.0	21.1	21.0	−24.2	11.7	35.9	0.2	36.1	11.8
GP2	33.3	22.0	22.0	21.9	11.3	11.3	0.0	0.1	0.1	11.4
CBI	34.3	56.6	21.2	21.0	−22.4	13.1	35.5	0.2	35.7	13.3
q_e	34.3	21.2	21.2	21.0	13.1	13.1	0.0	0.2	0.2	13.3
Tarantula	34.3	21.1	21.1	21.0	13.2	13.2	0.0	0.1	0.1	13.3
GP19	35.9	22.7	22.7	22.7	13.2	13.2	0.0	0.0	0.0	13.2
Fleiss	40.8	58.1	26.6	21.0	−17.2	14.2	31.4	5.7	37.1	19.9
Rogot1	41.2	25.0	25.0	21.0	16.1	16.1	0.0	4.1	4.1	20.2
Scott	41.2	58.1	26.8	21.0	−16.9	14.3	31.2	5.9	37.1	20.2
Euclid	44.6	24.0	24.0	21.0	20.6	20.6	0.0	3.0	3.0	23.6
Hamann	44.6	60.3	26.9	21.0	−15.8	17.7	33.4	5.9	39.4	23.6
Hamming	44.6	25.8	25.8	21.0	18.7	18.7	0.0	4.9	4.9	23.6
Rogers&Tanimoto	44.6	26.1	26.1	21.0	18.5	18.5	0.0	5.1	5.1	23.6
Simple Matching	44.6	25.9	25.9	21.0	18.7	18.7	0.0	4.9	4.9	23.6
Sokal	44.6	25.6	25.6	21.0	19.0	19.0	0.0	4.6	4.6	23.6
Wong2	44.6	60.1	27.0	21.0	−15.6	17.5	33.1	6.1	39.2	23.6
Binary	47.9	31.2	31.2	31.2	16.6	16.6	0.0	0.0	0.0	16.6
Russel&Rao	47.9	31.2	31.2	31.2	16.6	16.6	0.0	0.0	0.0	16.6
Wong1	47.9	31.2	31.2	31.2	16.6	16.6	0.0	0.0	0.0	16.6
Overall Average	36.3	35.7	23.5	22.0	0.6	12.8	12.2	1.5	13.7	14.4

experiments that M_1 is better than M_0 in most cases. Among all the 62×34 trials, M_1 gives better and worse results than M_0 for 1760 and 16 times, respectively. And in the remaining 332 trials, these two methods give the same results. Moreover, from Table 8, M_1 shows obvious superiority over M_0 where the average reduced *Expense* over all the 62×34 trials is 12.8%.

3. According to the discussion in Section 3, M_1 is essentially the same as Sendys for formulas which (1) always return positive values; or (2) return zero values *iff* $a_{ef} = 0$. Formulas that do not fulfill these two requirements may trigger the *NOR* problem. As discussed above, Sendys cannot handle this case, while M_1 corrects this problem. Overall, we observed 681 trials where M_1 is better than Sendys, and 1427 trials where M_1 is equal to Sendys. There is no case where M_1 is worse than Sendys.
4. After Propositions 1 and 2, we have $M_2 \rightarrow M_0$ and $M_2 \rightarrow M_1$. (As a reminder, after Definition 4, the "better" relation $\rightarrow$ means that M_2 never delivers worse *Expense* results than M_0 or M_1.) Our empirical results are consistent with the theoretical conclusions. From Table 7, we found 1842 and 315 trials where M_2 is better than M_0 and M_1, respectively; and 266 and 1793 trials where M_2 is equal to M_0 and M_1, respectively. Overall, the average *Expense* values of M_2 and M_1 are very close: the difference is only 1.5%. But the difference of the average *Expense* between M_2 and M_0 is as high as 14.4%. Finally, as compared with Sendys, M_2 shows better and equal performance for 869 and 1239 times, respectively; and the difference of overall average of *Expense* is 13.7%.

5.3.2. To address RQ2

Next, we will analyze how the comparisons vary across different risk formulas.

1. In Table 8, the formulas have been ordered according to their average *Expense* on M_0 ascendingly. We can observe that for M_0 the performance relations among all the formulas are consistent to previous theoretical studies (Xie et al., 2013a; 2013b; Yoo et al., 2017). As a reminder, the performance relations are not linear, but form a hierarchy (Xie et al., 2013a; Yoo et al., 2017).
2. As discussed in RQ1, M_1 has different performance from Sendys only when the formula triggers *NOR* problem.
   By checking the definitions in Section Appendix A, we can learn that "Naish1, Naish2, Wong3, GP3, AMPLE2, Goodman, Arithmetic Mean, Cohen, CBI, Scott, Fleiss, Hamann, and Wong2" may trigger this *NOR* problem. In our experiments, we have observed that M_1 significantly improves the performance of Sendys in all of these formulas except GP3, and performs equally to Sendys in the other formulas. As a reminder, Sendys performs equally to M_1 in GP3 because the condition for GP3 to reveal the *NOR* problem is $(a_{ef})^4 = a_{ep}$, which has never been satisfied by our experimental set-up. In other words, though GP3 theoretically has *NOR* problem, it has never triggered this problem in the experiments. Therefore, in summary, these observations are consistent with the theoretical analysis.
   Moreover, we can find from Table 8 that M_1 can reduce the *Expense* of Sendys in the formulas with *NOR* problem by 33 to 37%. These observations demonstrate the usefulness of M_1.
3. In the comparison between M_1 and M_0, we observed that the advantage of M_1 over M_0 is getting more obvious in formulas

Table 9
Average *ExpenseM* values for multiple-fault scenarios.

Formula R	1-fault (*Expense* %)			2-fault (%)			3-fault (%)			4-fault (%)		
	M_0	M_1	Sendys	M_0	M_1	Sendys	M_0	M_1	Sendys	M_0	M_1	Sendys
Naish1	26.5	21.0	58.0	42.5	36.7	65.0	51.6	40.8	63.9	49.5	41.7	62.8
Naish2	26.5	21.0	57.0	35.6	27.6	65.9	37.9	28.7	68.0	41.3	29.5	69.1
Wong3	27.7	21.5	55.8	36.9	30.2	61.4	37.6	31.4	65.6	41.1	32.1	66.7
AMPLE2	31.9	21.0	57.2	36.5	31.2	64.5	40.2	32.1	65.1	41.5	31.2	62.8
Goodman	32.6	21.0	54.0	36.3	31.2	61.0	41.2	32.4	63.1	43.2	33.1	62.6
Arithmetic Mean	32.6	21.0	57.5	42.3	33.2	63.6	46.5	36.2	61.5	47.9	37.0	64.0
Cohen	32.8	21.1	57.0	41.3	33.2	64.6	41.2	35.6	63.3	48.5	35.4	65.4
CBI	34.3	21.2	56.6	39.8	31.6	62.7	39.8	33.5	66.3	43.5	32.5	68.6
Fleiss	40.8	26.6	58.1	45.6	33.1	65.1	47.6	38.4	64.7	48.5	37.6	66.3
Scott	41.2	26.8	58.1	41.2	33.1	63.6	47.5	39.1	67.1	47.9	36.5	69.0
Hamann	44.6	26.9	60.3	49.8	36.2	65.9	53.2	38.3	65.2	53.9	39.2	67.0
Wong2	44.6	27.0	60.1	49.8	36.2	64.6	53.2	38.5	64.0	53.9	39.7	64.9
Average	34.7	23.0	57.5	41.5	32.8	64.0	44.8	35.4	64.8	46.7	35.5	65.8

with worse performance. In Table 7, M_1 has about 1/3 of the 62 faulty versions give equal *Expense* with M_0 on top formulas. And with the decreasing of formulas' performance, M_1 tends to have more faulty versions with better *Expense* than M_0. For example, M_1 delivers better *Expense* on "Binary, Russel&Rao and Wong1" for 61 times.

Similarly, in Table 8, the reduced *Expense* tends to be lower in top formulas, and higher in bottom formulas. The smallest difference on *Expense* is 5.5%, but rises to about 20% on the bottom few formulas.

4. As discussed above, due to the "*NOR*" problem, Sendys does not perform well on formulas "Naish1, Naish2, Wong3, AMPLE2, Goodman, Arithmetic Mean, Cohen, CBI, Scott, Fleiss, Hamann, and Wong2". Actually, apart from M_1, M_0 also outperforms Sendys on these formulas. And in the other formulas, since Sendys is equivalent to M_1, its comparison between M_0 has the same rule as the comparison between M_1 and M_0.
5. After Proposition 4, M_2 should give equal *Expense* when configured with formulas sharing the same monotonicity on the faulty border. According to previous studies (Yoo et al., 2017), we can learn that (i) apart from GP2, GP3, GP19, Binary, Russel&Rao and Wong1, all the other formulas share the same monotonicity; (ii) Binary, Russel&Rao and Wong1 share the same monotonicity; and (iii) GP2, GP3 and GP19 have different monotonicity from all the other formulas. As a consequence, in M_2, we should have all formulas except "GP2, GP3, GP19, Binary, Russel&Rao and Wong1" share the same *Expense*, "Binary, Russel&Rao and Wong1" share the same *Expense*, and GP2, GP3 and GP19 have different *Expense* from others. Our experimental results are consistent with this theoretical conclusion. This can be seen from Table 8, where apart from the above formulas, M_2 always gives *Expense* as 21.0%, "Binary, Russel&Rao and Wong1" share the same *Expense* of 31.2%, and GP2, GP3 and GP19 give *Expense* values as 21.9%, 21.5%, and 22.7%, respectively.

   As a consequence, by individually comparing the relatively stable M_2 with M_0, Sendys, and M_1, we can find that M_2 demonstrates the most obvious advantage in bottom formulas. More importantly, M_2 performs better than M_0 in theoretically maximal formulas. For example, in GP13, Naish1 and Naish which are proved to be maximal, M_2 gives better performance than M_0 for 36 times, and equal performance than M_0 for 26 times. On average, *Expense* of M_2 is 5.5% lower than M_0 in these three formulas. This is consistent with Proposition 5.

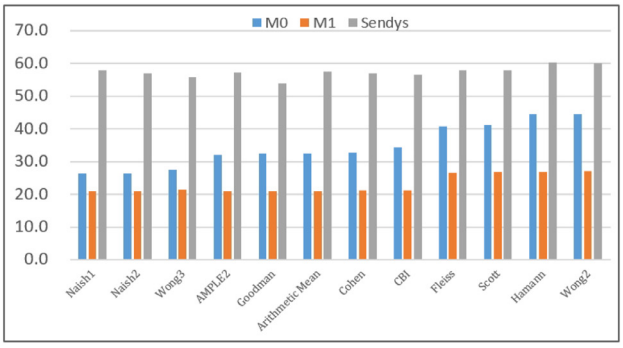
5.3.3. To address RQ3

Table 9 presents the comparison among M_0, Sendys, and M_1 in two-fault, three-fault and four-fault scenarios. Cells under these three scenarios record the average of *ExpenseM* for methods configured with each formula, over the corresponding faulty versions shown in Table 6. We also list the average *Expense* of single-fault scenario for reference. As discussed above, among all the investigated formulas, Sendys has different performance from M_1 only on the 12 formulas, namely, "Naish1, Naish2, Wong3, AMPLE2, Goodman, Arithmetic Mean, Cohen, CBI, Scott, Fleiss, Hamann, and Wong2". Therefore, the analysis of RQ3 focuses on these formulas. The bottom line of Table 9 gives the average of *ExpenseM* (*Expense* for single-fault case) over all the $m \times 12$ trials, where m is the number of faulty versions under the corresponding scenario (shown in Table 6), and 12 is the number of formulas. Besides, as explained in Section 5.1, M_2 should not be considered in this experiment. Fig. 4 depicts these data via bar-charts, where the vertical axis is the *Expense* or *ExpenseM* value. Accordingly, we have the following observations.

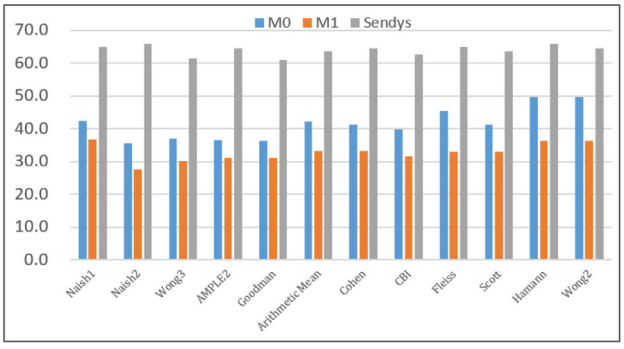
1. The performance relation in single-fault case among formulas for M_0 no longer holds in two-fault, three-fault and four-fault scenarios. In particular, one of the maximal formula in single-fault case, Naish1, does not perform well in multiple-fault cases. For M_0, Naish1 gives average *Expense* of 26.5% in single-fault case, but gives average *ExpenseM* as high as 42.5%, 51.6% and 49.5% in two-fault, three-fault and four-fault scenarios, respectively. However, equivalent formulas "Hamann" and "Wong2" in single-fault case still remain equivalent in multiple-fault cases, which is consistent with the theoretical explanation that these two formulas always produce identical ranking list for all statements.

   The overall average performance of M_0 decreases when the fault number increases, but the largest difference exists between "1-fault" and "2-fault" scenarios, that is, $41.5\% - 34.7\% = 6.8\%$. For fault number greater than 2, the decreases become smaller. The difference between "2-fault" and "3-fault" is $44.8\% - 41.5\% = 3.3\%$, and between "3-fault" and "4-fault" is $46.7\% - 44.8\% = 1.9\%$.
2. Similarly, performance of M_1 decreases significantly from "1-fault" to "2-fault", where the difference of average performance measure is $32.8\% - 23.0\% = 9.8\%$. But the average measure in "3-fault" and "4-fault" cases are 35.4% and 35.5%, respectively, which are actually very close to "2-fault" case. Besides, different formulas do not show obvious performance relation in multiple-fault cases for M_1.

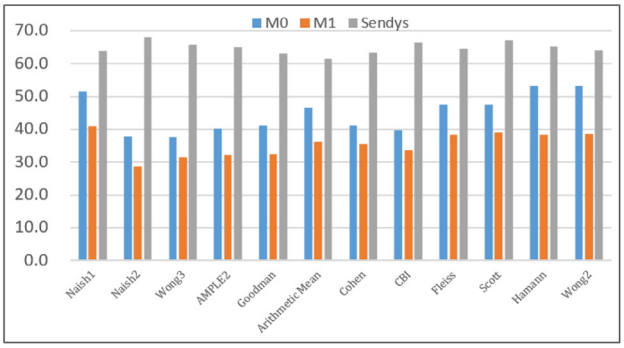
   As compared with M_0, M_1 shows obvious advantage in all these four scenarios. The difference between their average *Expense* in single-fault, two-fault, three-fault and four-fault scenarios are $(23.0\% - 34.7\% = -11.7\%)$, $(32.8\% - 41.5\% = -8.7\%)$, $(35.4\% - 44.8\% = -9.4\%)$ and $(35.5\% - 46.7\% = -11.2\%)$, respectively.



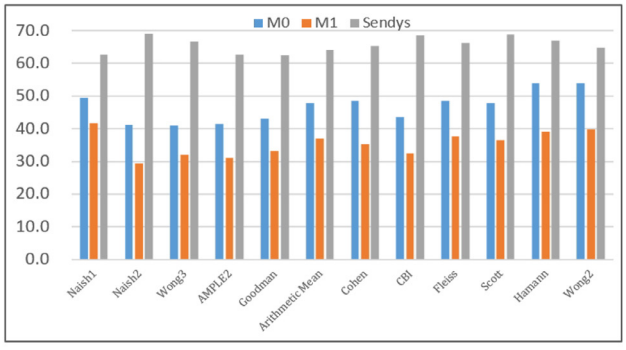
(a) 1-fault



(b) 2-fault



(c) 3-fault



(d) 4-fault

Fig. 4. Average performance for multiple-fault scenarios.

3. Due to the *NOR* problem, Sendys does not perform well for these listed formulas, in all the four scenarios. Its performance is relatively stable approximately between 60% and 70% when the number of faults increases.

RQ1 and RQ2 have demonstrated the usefulness of M_1 in improving the performance of Sendys in single-fault scenario. In RQ3, we demonstrate that M_1 can also significantly improve Sendys in multiple-fault cases. The difference between their average *Expense* in two-fault, three-fault and four-fault scenarios are $(64.0\% - 32.8\% = 31.2\%)$, $(64.8\% - 35.4\% = 29.4\%)$, and $(65.8\% - 35.5\% = 30.3\%)$, respectively.

6. Discussion

Results in multiple-fault cases have triggered us to consider an obvious but important question: *what is a better way to perform multiple-fault localization*.

In practice, the quantity of the faults is actually unknown. Therefore, some researchers proposed methods, such as Sendys, for simultaneously locating faults. However, from our experiments, we found that even though M_1 has given relatively good *ExpenseM* and shown obvious advantages over traditional SBFL in multiple-fault cases, it still has worse performance than itself in single-fault case. Therefore, we conjecture that it may better to locate unknown number of faults by iteratively performing M_1 or M_2 on "single-fault case", rather than locate them simultaneously. In fact, this is not unrealistic (Xie et al., 2013a).

In the community of fault localization, there are two approaches for locating unknown number of faults, namely "*sequential debugging*" (Jones et al., 2007) and "*parallel debugging*" (Dickinson et al., 2001; Podgurski et al., 2003; Liu and Han, 2006; Zheng et al., 2006).

Sequential debugging localizes one fault each time and conducts regression testing to localize the next fault (DiGiuseppe and Jones, 2011). According to the investigation in DiGiuseppe and Jones (2011), the *Expense* of the highest ranked faulty statement is not affected significantly by the quantity of faults. In other words, given a program with unknown number of faults, developers can simply treat it as single fault and perform fault localization. After fixing the first fault, they can repeat the process to locate another one (DiGiuseppe and Jones, 2011).

Parallel debugging (which is different from simultaneous debugging) adopts the "divide-and-conquer" idea. It first clusters all test cases into several specialized test suites based on various execution information, and each of the test suites targets an individual fault. In practice, each specialized test suite is dispatched to a particular debugger, who is supposed to focus on the corresponding single fault (Liu and Han, 2006; Zheng et al., 2006; Jones et al., 2007). As a consequence, single-fault localization methods can be adopted.

To summarize, both of the above two approaches essentially transform "simultaneously locating multiple faults" into "iteratively locating single fault". As a consequence, all the analysis in this paper becomes applicable.

More importantly, we have actually found methods M_2 and M_0^G can even outperform theoretical maximal method given in previous studies (Xie et al., 2013a; Yoo et al., 2017). The major reason is the use of dynamic slices, which can help to eliminate noises in the spectra information.

7. Related work

In recent years, performance comparison among SBFL risk formulas has been conducted through empirical and theoretical analysis. Empirical studies include Tarantula evaluation studies (Jones and Harrold, 2005), the performance comparison among *Ochiai, Jaccard* and *Tarantula* (Abreu et al., 2009b) and Wong's methods investigation (Wong et al., 2010).

However, empirical studies are still not comprehensive enough to support a fair evaluation for the investigated SBFL techniques. Therefore, theoretical analysis for SBFL risk formulas has been conducted. Naish et al. (2011) proposed a model ITE2 to theoretically

evaluate the performance of risk formulas and two optimal risk formulas for the specific program structures are proposed according to their theoretical analysis and the performance measurement.

Furthermore, Xie et al. (2013a) proposed a theoretical framework based on set theory to analyze the performance of risk formulas. Performance hierarchies of risk formulas are proved for any faulty program and test suites. In their follow-up work (Xie et al., 2013b), they applied the theoretical framework on risk formulas designed automatically by a search-based technique, namely Genetic Programming (Yoo, 2012). In Yoo et al. (2017), Yoo et al. proved the sufficient and necessary condition of being local maximal for any given formula and the non-existence of the greatest risk formula.

The above research work focuses on the analysis of SBFL risk formulas through empirical studies and theoretical analysis. However, different from the previous investigation work, our work focuses on the performance estimation of a combination work with different SBFL risk formulas through theoretical analysis.

8. Conclusion

Researchers proposed various approaches combining different fault localization techniques with SBFL so that the individual strength of two techniques can be utilized. Sendys, which is a combination of slicing-hitting-set-computation and SBFL, has been demonstrated to be an effective technique to locate the faults. However, all previous evaluations about Sendys were exclusively obtained from experiments, which contain inevitable threats to validity. Therefore, in this paper, we perform a theoretical analysis on Sendys to reveal an in-depth perception of this technique.

We first patch a loophole of *NOR* problem in the original Sendys, proposing a variant of it. We generalize the framework in Xie et al. (2013a) to support the analysis on the integration of fault localization algorithm and the risk formula. After theoretically analyzing some properties of the above variant, we propose an enhanced version of Sendys, and prove its superiority over the above variant and traditional SBFL in single-fault scenario. Moreover, we reformulate the enhanced Sendys into a short-cut without changing its performance, and prove that this enhancement can even outperform any traditional SBFL maximal formulas.

To complement the theoretical analysis, we also conduct empirical studies to give quantitative comparisons among different methods. With empirical studies, we demonstrate the significant advantages of the enhanced Sendys, as well as its stability across different formulas in single-fault scenario. And in multiple-fault cases, the variant of Sendys is observed to have the best performance. Besides, this variant has shown great helpfulness in improving the bad performance of the original Sendys when encountering the *NOR* problem.

To summarize, we take the first step to provide an in-depth analysis on a combined fault localization technique, from both theoretical and empirical perspectives. In our future studies, we will further extend our framework and conduct more experiments to validate our conjecture in Section 6 about “better solution for multiple-fault localization”.

Acknowledgment

This work was supported by the National Key R&D Program of China under grant number of 2018YFB1003901, the National Natural Science Foundation of China under grant numbers of 61572375, 61772263 and 61472178.

Appendix A. Definitions for all investigated formulas

In this appendix, we list definitions for all investigated formulas, which are obtained from Xie et al. (2013a) and Yoo et al. (2017).

Name		Formula expression
ER1	GP13	$a_{ef}(1+\frac{1}{2a_{ep}+a_{ef}})$
	Naish1	$\begin{cases}-1 & \text{if } a_{ef}<F\\ P-a_{ep} & \text{if } a_{ef}=F\end{cases}$
	Naish2	$a_{ef}-\frac{a_{ep}}{a_{ep}+a_{np}+1}$
Kulczynski2		$\frac{1}{2}(\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ef}}{a_{ef}+a_{ep}})$
Wong3		$a_{ef}-h$, where $h=\begin{cases}a_{ep} & \text{if}a_{ep}\leq 2\\ 2+0.1(a_{ep}-2) & \text{if}2<a_{ep}\leq 10\\ 2.8+0.001(a_{ep}-10) & \text{if}a_{ep}>10\end{cases}$
M2		$\frac{a_{ef}}{a_{ef}+a_{np}+2(a_{nf}+a_{ep})}$
Ochiai		$\frac{a_{ef}}{\sqrt{(a_{ef}+a_{nf})(a_{ef}+a_{ep})}}$
GP3		$\sqrt{\lvert a_{ef}^2-\sqrt{a_{ep}}\rvert}$
AMPLE2		$\frac{a_{ef}}{a_{ef}+a_{nf}}-\frac{a_{ep}}{a_{ep}+a_{np}}$
ER2	Anderberg	$\frac{a_{ef}}{a_{ef}+2(a_{nf}+a_{ep})}$
	Dice	$\frac{2a_{ef}}{a_{ef}+a_{nf}+a_{ep}}$
	Goodman	$\frac{2a_{ef}-a_{nf}-a_{ep}}{2a_{ef}+a_{nf}+a_{ep}}$
	Jaccard	$\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}}$
	Sørensen-Dice	$\frac{2a_{ef}}{2a_{ef}+a_{nf}+a_{ep}}$
Arithmetic mean		$\frac{2a_{ef}a_{np}-2a_{nf}a_{ep}}{(a_{ef}+a_{ep})(a_{np}+a_{nf})+(a_{ef}+a_{nf})(a_{ep}+a_{np})}$
Cohen		$\frac{2a_{ef}a_{np}-2a_{nf}a_{ep}}{(a_{ef}+a_{ep})(a_{np}+a_{ep})+(a_{ef}+a_{nf})(a_{nf}+a_{np})}$
GP2		$2(a_{ef}+\sqrt{a_{np}})+\sqrt{a_{ep}}$
ER3	CBI	$\frac{a_{ef}}{a_{ef}+a_{ep}}-\frac{a_{ef}+a_{nf}}{a_{ef}+a_{nf}+a_{ep}+a_{np}}$
	qe	$\frac{a_{ef}}{a_{ef}+a_{ep}}$
	Tarantula	$\frac{a_{ef}}{a_{ef}+a_{nf}}/(\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ep}}{a_{ep}+a_{np}})$
GP19		$a_{ef}\sqrt{\lvert a_{ep}-a_{ef}+a_{nf}-a_{np}\rvert}$
Fleiss		$\frac{4a_{ef}a_{np}-4a_{nf}a_{ep}-(a_{nf}-a_{ep})^2}{(2a_{ef}+a_{nf}+a_{ep})+(2a_{np}+a_{nf}+a_{ep})}$
ER6	Rogot1	$\frac{1}{2}(\frac{a_{ef}}{2a_{ef}+a_{nf}+a_{ep}}+\frac{a_{np}}{2a_{np}+a_{nf}+a_{ep}})$
	Scott	$\frac{4a_{ef}a_{np}-4a_{nf}a_{ep}-(a_{nf}-a_{ep})^2}{(2a_{ef}+a_{nf}+a_{ep})(2a_{np}+a_{nf}+a_{ep})}$
ER4	Euclid	$\sqrt{a_{ef}+a_{np}}$
	Hamann	$\frac{a_{ef}+a_{np}-a_{nf}-a_{ep}}{a_{ef}+a_{nf}+a_{ep}+a_{np}}$
	Hamming	$a_{ef}+a_{np}$
	Rogers&Tanimoto	$\frac{a_{ef}+a_{np}}{a_{ef}+a_{np}+2(a_{nf}+a_{ep})}$
	Simple Matching	$\frac{a_{ef}+a_{np}}{a_{ef}+a_{nf}+a_{ep}+a_{np}}$
	Sokal	$\frac{2(a_{ef}+a_{np})}{2(a_{ef}+a_{np})+a_{nf}+a_{ep}}$
	Wong2	$a_{ef}-a_{ep}$
ER5	Binary	$\begin{cases}0 & \text{if}a_{ef}<F\\ 1 & \text{if}a_{ef}=F\end{cases}$
	Russel & Rao	$\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}+a_{np}}$
	Wong1	a_{ef}

References

Abreu, R., Zoeteweij, P., van Gemund, A. J. C., 2009a. Spectrum-based multiple fault localization. In: Proceedings of Twenty–Fourth IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, pp. 88–99.

Abreu, R., Zoeteweij, P., Golsteijn, R., Gemund, A.J.C.V., 2009b. A practical evaluation of spectrum-based fault localization. J. Syst. Softw. 82 (11), 1780–1792.

Chen, T. Y., Xie, X., Kuo, F.-C., Xu, B., 2015. A revisit of a theoretical analysis on spectrum-based fault localization. In: Proceedings of Thirty–Ninth Annual international Computers, Software and Applications Conference. IEEE, Taichung, pp. 17–22.

Collofello, J.S., Woodfield, S.N., 1989. Evaluating the effectiveness of reliability-assurance techniques. J. Syst. Softw. 9 (3), 191–195.

Dickinson, W., Leon, D., Podgurski, A., 2001. Finding failures by cluster analysis of execution profiles. In: Proceedings of the Twenty–Third International Conference on Software Engineering, pp. 339–348.

DiGiuseppe, N., Jones, J. A., 2011. On the influence of multiple faults on coverage-based fault localization. In: Proceedings of the International Symposium on Software Testing and Analysis. Toronto, Canada, pp. 199–209.
Fang, C., Chen, Z., Xu, B., 2012. Comparing logic coverage criteria on test case prioritization. Sci. CHINA Inf. Sci. 55 (12), 2826–2840.
Greiner, R., Smith, B.A., Wilkerson, R.W., 1989. A correction to the algorithm in Reiter's theory of diagnosis. Artif. Intell. 41 (1), 79–88.
http://nanoxml.sourceforge.net/orig/index.html.
Hofer, B., Wotawa, F., 2012. Spectrum enhanced dynamic slicing for better fault localization. In: Proceedings of the Twentieth European Conference on Artificial Intelligence. IOS Press, Montpellier, France, pp. 420–425.
Janssen, T., Abreu, R., Van Gemund, A. J. C., 2009. Zoltar: a toolset for automatic fault localization. In: Proceedings of the Twenty–Fourth IEEE/ACM International Conference on Automated Software Engineering. IEEE, Auckland, New Zealand, pp. 662–664.
Jones, J. A., Bowring, J. F., Harrold, M. J., 2007. Debugging in parallel. In: Proceedings of the International Symposium on Software Testing and Analysis. New York, USA, pp. 16–26.
Jones, J. A., Harrold, M. J., 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In: Proceedings of the Twentieth International Conference on Automated Software Engineering. ACM, Long Beach, USA, pp. 273–282.
Jones, J. A., Harrold, M. J., Stasko, J., 2002. Visualization of test information to assist fault localization. In: Proceedings of the Twenty–Fourth International Conference on Software Engineering. ACM, Orlando, USA, pp. 467–477.
Lee, H. J., Naish, L., Ramamohanarao, K., 2009. Study of the relationship of bug consistency with respect to performance of spectra metrics. In: Proceedings of the Second IEEE International Conference on Computer Science and Information Technology. IEEE, Beijing, China, pp. 501–508.
Lei, Y., Mao, X., Dai, Z., Wang, C., 2012. Effective statistical fault localization using program slices. In: Proceedings of the Thirty–Sixth Annual international Computers, Software and Applications Conference. IEEE, Izmir, Turkey, pp. 1–10.
Liblit, B., Naik, M., Zheng, A. X., Aiken, A., Jordan, M. I., 2005. Scalable statistical bug isolation. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, Chicago, Illinois, USA, pp. 15–26.
Liu, C., Han, J., 2006. Failure proximity: a fault localization-based approach. In: Proceedings of the Fourteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, USA, pp. 46–56.
Liu, C., Yan, X., Fei, L., Han, J., Midkiff, S., 2006. Sober: statistical model-based bug localization. IEEE Trans. Softw. Eng. 30 (5), 286–295.
Naish, L., Lee, H.J., Ramamohanarao, K., 2011. A model for spectra-based software diagnosis. ACM Trans. Software Eng. Method 20 (3), 1–32.
Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., Wang, B., 2003. Automated support for classifying software failure reports. In: Proceedings of the Twenty–Fifth International Conference on Software Engineering. Portland, Oregon, USA, pp. 465–475.
Reiter, R., 1987. A theory of diagnosis from first principles. Artif. Intell. 32 (1), 57–95.
Tu, J., Chen, L., Zhou, Y., Zhao, J., Xu, B., 2012. Leveraging method call anomalies to improve the effectiveness of spectrum-based fault localization techniques for object-oriented programs. In: Proceedings of the Twelfth International Conference on Quality Software. IEEE, Xi'an, China, pp. 1–8.
Wang, C., 2018. Analysis on the Issues in Large-scaled Open Source Software Repository. Wuhan University.
Wang, C., Xie, X., Liang, P., Xuan, J., 2017. Multi-perspective visualization to assist code change review. In: Proceedings of Twenty–Fourth Asia-Pacific Software Engineering Conference, pp. 564–569.
Wong, W.E., Debroy, V., Choi, B., 2010. A family of code coverage-based heuristics for effective fault localization. J. Syst. Softw. 83 (2), 188–208.
Wong, W.E., Qi, Y., 2006. Effective program debugging based on execution slices and inter-block data dependency. J. Syst. Softw. 79 (7), 891–903.
Wotawa, F., 2002. On the relationship between model-based debugging and program slicing. Artif. Intell. 135, 125–143.
Wotawa, F., 2010. Fault localization based on dynamic slicing and hitting-set computation. In: Proceedings of the Tenth International Conference on Quality Software. IEEE, Zhangjiajie, China, pp. 161–170.
Xie, X., Chen, T.Y., Kuo, F.-C., Xu, B., 2013a. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans. Softw. Eng. Method. 22 (4), 1–40.
Xie, X., Chen, T. Y., Xu, B., 2010. Isolating suspiciousness from spectrum-based fault localization techniques. In: Proceedings of the Tenth International Conference on Quality Software. IEEE, Zhangjiajie, China, pp. 385–392.
Xie, X., Kuo, F.-C., Chen, T., Yoo, S., Harman, M., 2013b. Provably optimal and human-competitive results in SBSE for spectrum based fault localisation. In: Proceedings of the Fifth Symposium on Search Based Software Engineering. Springer, St. Petersburg, Russia, pp. 224–238.
Xu, J., Zhang, Z., Chan, W., Tse, T., Li, S., 2013. A general noise-reduction framework for fault localization of java programs. Inf. Softw. Technol. 55 (5), 880–896.
Yoo, S., 2012. Evolving human competitive spectra-based fault localisation techniques. Search Based Softw. Eng. 7515, 244–258.
Yoo, S., Xie, X., Kuo, F.-C., Chen, T.Y., Harman, M., 2017. Human competitiveness of genetic programming in spectrum-based fault localisation: theoretical and empirical analysis. ACM Trans. Softw. Eng. Method. 26 (1), 4:1–4:30.
Zheng, A. X., Jordan, M. I., Liblit, B., Naik, M., Aiken, A., 2006. Statistical debugging: simultaneous identification of multiple bugs. In: Proceedings of the Twenty–Third International Conference on Machine Learning. New York, USA, pp. 1105–1112.

Jingxuan Tu received Ph.D. degree in 2017. He is currently working in Huawei Research Center (Shanghai). His research interests include software analysis, testing and debugging.

Xiaoyuan Xie, received B.Sc. and M.Phil. degrees in Computer Science from Southeast University, China in 2005 and 2007, respectively, and received Ph.D. degree in Computer Science from Swinburne University of Technology, Australia in 2012. She is currently a professor in School of Computer Science, Wuhan University, China. Her research interests include software analysis, testing, debugging, and search-based software engineering.

Tsong Yueh Chen received the B.Sc. and M.Phil. degrees from the University of Hong Kong, China, the M.Sc. degree and DIC from the Imperial College of Science and Technology, London, U.K., and the Ph.D. degree from the University of Melbourne, Australia. He is currently a Professor of Software Engineering in the Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia. He is the originator of metamorphic testing and adaptive random testing. His current research interests include software testing, debugging, and program repair.

Baowen Xu is currently a professor in the Department of Computer Science and Technology, Nanjing University. He has published more than 200 papers in the international journals and conferences like the ACM Trans. Software Engineering and Methodology (TOSEM), the IEEE Trans. Software Engineering (TSE), the Journal of Artificial Intelligence Research and Int. Joint Conf. Artificial Intelligence (IJCAI).