Accuracy Graphs of Spectrum-Based Fault Localization Formulas

Chung Man Tang, W. K. Chan, *Member, IEEE*, Yuen Tak Yu, *Member, IEEE*, and Zhenyu Zhang, *Member, IEEE*

***Abstract*—The effectiveness of spectrum-based fault localization techniques primarily relies on the accuracy of their fault localization formulas. Theoretical studies prove the relative accuracy orders of selected formulas under certain assumptions, forming a graph of their theoretical accuracy relations. However, it is unclear whether in such a graph the relative positions of these formulas may change when some assumptions are relaxed. On the other hand, empirical studies can measure the actual accuracy of any formula in controlled settings that more closely approximate practical scenarios but in less general contexts. In this paper, we propose an empirical framework of accuracy graphs and their construction that reveal the relative accuracy of formulas. Our work not only evaluates the association between certain assumptions and the theoretical relations among formulas, but also expands our knowledge to reveal new potential accuracy relationships of other formulas which have not been discovered by theoretical analysis. Using our proposed framework, we identified a list of formula pairs in which a formula is consistently statistically more accurate than or similar in accuracy to another, enlightening directions for further theoretical analysis.**

***Index Terms*—Accuracy graph, accuracy relation, fault localization, program debugging, relative accuracy.**

Acronyms

ERAO	Empirical rank-ahead-of (relation).
SBFL	Spectrum-based fault localization.
TRAO	Theoretical rank-ahead-of (relation).
XCKX	Work of Xie, Chen, Kuo, and Xu [63].

Manuscript received June 8, 2016; revised October 31, 2016 and December 28, 2016; accepted February 26, 2017. Date of publication May 3, 2017; date of current version May 31, 2017. This work was supported in part by the General Research Fund of the Research Grants Council of Hong Kong under Project 111313 and 125113, in part by grants from the National Key Basic Research Program of China under Project 2014CB340702, in part by the National Natural Science Foundation of China under Project 61379045, in part by the China Scholarship Council under Project 201604910232, and in part by the Open Project Fund of the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China under Project SYSKF1608. Associate Editor: S. Li. *(Corresponding author: Yuen Tak Yu.)*

C. M. Tang and Y. T. Yu are with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: c.m.tang@my.cityu.edu.hk; csytyu@cityu.edu.hk).

W. K. Chan is with the Department of Computer Science, City University of Hong Kong, Hong Kong, and also with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100864, China (e-mail: wkchan@cityu.edu.hk).

Z. Zhang is with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100864, China, and also with the Institute for Software Research, University of California, Irvine, CA 92697 USA (e-mail: zhangzy@ios.ac.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2017.2688487

Notation

a_{ep}, a_{ef}	Number of passed runs (a_{ep}) and failed runs (a_{ef}) in which a program entity is executed.
a_{np}, a_{nf}	Number of passed runs (a_{np}) and failed runs (a_{nf}) in which a program entity is not executed.
D_y	Dataset *y* for analysis of accuracy relations.
F_j	A spectrum-based fault localization formula.
G_x	The accuracy graph constructed from a dataset.
N	A node in an accuracy graph.
P_k	A subject program in this study.
R_w, R_b, R_a	Ranking schemes for worst, best, average cases.
S	A set $s_1, \ldots, s_m$ of program entities s_i.
susp	Suspiciousness score of a program entity.
T	A test suite, i.e., a set $t_1, \ldots, t_n$ of test cases t_i.
V	A faulty version of a (subject) program.

I. Introduction

PROGRAM debugging is the de facto approach to improve the reliability of program code. One major challenge is to locate the faults that cause failures [1], [66]. The state-of-the-art techniques for this purpose, such as slicing [29], [57], delta debugging [66] and *spectrum-based fault localization* (SBFL) [27], [62], seek to either reduce [29], [57], [66] or prioritize [27] the program code for further examination by the developers.

SBFL has received great attention in the last decade [62]. It utilizes a *program spectrum*, which is an execution profile that records which parts of the program are active during execution. SBFL applies a formula to assign a *suspiciousness score* [27] to each program entity based on the program spectra obtained through a test suite. Such a formula is called an *SBFL formula* (or simply a *formula*). The program entities are then ranked (by their suspiciousness scores) for subsequent debugging activities. Various types of program entities have been studied in SBFL, such as statement [1], branch [69], and path [8].

Many empirical studies [23], [24], [27], [34], [44], [45], [47], [51], [59], [60] have investigated the accuracy of formulas in various contexts [62]. The relative accuracy of formulas is commonly quantified by their difference in the sizes of the recommended program entity lists that include at least one target (faulty) program entity. If the sublist size recommended by a formula is smaller than the sublist size recommended by another formula, then the former is deemed more accurate.

Some theoretical studies [41], [63] analyze the relative accuracy of formulas by mathematical proofs under certain

assumptions adopted in the SBFL literature, which we refer to as *the required assumptions*, or *the assumptions* for short. For example, both the model in [41] and the analysis in [63] are valid only for a program with the fault residing entirely in the same program entity. Under these assumptions, some formulas were proved to be never less accurate than some other formulas, and the accuracy of formulas in some groups were proved to be always identical [41], [63]. These studies pioneered to build a partial order hierarchy of accuracy relations based on the relative accuracy order of formulas. We call such a pairwise relation of formulas a *theoretical rank-ahead-of (TRAO) relation* and the accuracy relation observed in empirical settings as an *empirical rank-ahead-of (ERAO) relation*.

Many formulas associated with known TRAO relations are interconnected [63], forming a directed graph G_{theory} (see Fig. 2) with each formula or each group of equally accurate formulas as a node and the TRAO relation between two nodes as a directed edge [63]. We refer to a graph that captures the TRAO or ERAO relations of formulas as an *accuracy graph*.

The assumptions made in theoretical analyses are, however, seldom all valid in every real scenario. We thus ask two questions: 1) What do the accuracy graphs look like when certain assumptions do not hold? 2) How can one expand known accuracy graphs to include formulas with no currently known TRAO relations?

To answer these questions, we have developed a novel methodology that empirically discovers the ERAO relations among SBFL formulas based on sound and reliable statistical techniques, leading to the systematic construction of empirical accuracy graphs. Section III reports our methodology in detail.

Using our methodology, we performed an empirical study on a benchmark of 17 well-known C and Java programs with real or seeded faults, including 7 programs in the Siemens suite, *space*, 4 UNIX utility programs, and 5 Defects4J programs. By executing each faulty version against each corresponding test suite, we identified whether they satisfied the required assumptions. With this information, we constructed four kinds of combinations of subjects (faulty program versions and test suites).

1) The "base" set, which satisfies all the assumptions;
2) another set, which satisfies all the assumptions but one;
3) a third set, which satisfies all the assumptions but another one;
4) the full set, which contains every valid combination.

We computed the accuracy of the 30 formulas analyzed in the work of Xie, Chen, Kuo, and Xu [63] (abbreviated as *XCKX*). These 30 formulas are hereafter called the *XCKX formulas*.

Specifically, for every pair of XCKX formulas computed over each set of combinations, we analyzed whether 1) one formula was statistically more accurate than another formula, or 2) the two formulas were statistically not significantly different in accuracy. We used the Kruskal–Wallis test, which is a nonparametric test to get rid of the assumption of normal distributions in the dataset, to compare the accuracy of every formula pair and then the multiple comparison test to compare all formulas as a whole. If the relative accuracy of two formulas F_i and F_j is consistent across all subjects, one of the following ERAO relations is established.

1) F_i is consistently more accurate than F_j.
2) F_i and F_j are consistently similar.
3) F_i is consistently less accurate than F_j (see Section III-E).

From these ERAO relations, we constructed accuracy graphs for each set of combinations to enable us to compare the ERAO relations of the XCKX formulas with the TRAO relations. The resulting accuracy graphs are called *XCKX-formula graphs* or *unexpanded graphs*. We then expanded the XCKX-formula graphs with additional nodes by computing the accuracy of 12 other SBFL formulas, called *non-XCKX formulas*. The resulting (expanded) accuracy graphs consist of both *XCKX formulas* and *non-XCKX formulas* and are simply called *expanded graphs*.

The results of our experiment showed that, when all the required assumptions were satisfied, both the expanded and unexpanded graphs included all the TRAO relations in G_{theory} proved in [63] (see Table V). Thus, using our construction methodology, we successfully obtained a set of ERAO relations consistent with G_{theory}. Our results also revealed extra ERAO relations not found by previous theoretical studies. We have identified a set of these extra ERAO relations, which appear consistently in every empirically constructed accuracy graph in our experiments. Since our model is statistical in nature, this set of relations suggests a potential probabilistic version of TRAO dependencies among the formulas for further theoretical studies, which are more general than what we currently know from the literature [63]. Moreover, in the accuracy graphs that relaxed a specific assumption, no edge of TRAO relation was reversed, that is, the TRAO relations were still held in our empirical setting despite the removal of some assumption(s). Furthermore, while previous theoretical results have only proved two formulas to be maximal in accuracy, our results have revealed a *superset* of maximal formulas under typical or more generalized settings. Thus, whenever the original two maximal formulas are used in practice or in an experimental investigation, we suggest to also include the other formulas in this superset of maximal formulas.

To the best of our knowledge, this is the *first* work that can statistically simulate the TRAO relations (obtained from theory) empirically using the ERAO relations (obtained from experiment) constructed through our novel methodology. The ERAO relations that consistently appear in all expanded graphs offer clues that the corresponding formulas may have unrevealed theoretical dependencies, which may suggest preferential regions in the accuracy graphs for further theoretical studies. Unlike previous theoretical analysis frameworks that heavily depend on the investigator's personal insights and ad hoc inspirations to discover and prove TRAO relations for constructing the corresponding accuracy graphs, our methodology has provided a mechanical and objective technique for investigators to systematically reveal empirical accuracy relations and construct the accuracy graphs of any sets of SBFL formulas of interest given the appropriate available experimental subjects and settings. More importantly, unlike mathematical proofs relying on the efforts and intellectual ability of the researchers, our methodology can be automated and thus a mega-scale experiment can be conducted in the future which accelerates the pace of research discovery.

The rest of this paper is organized as follows. Section II revisits the preliminaries. Section III presents our empirical framework and methodology of constructing accuracy graphs. Section IV analyzes the accuracy graphs constructed from our experiment and benchmarks with existing work. Section V reviews the related work. Section VI concludes this paper.

II. Preliminaries

In this section, we introduce the background, notations, and definitions that are essential for understanding this paper.

A. Spectrum-Based Fault Localization

Consider a faulty program version V (e.g., by exposing a failure through testing). SBFL aims at including a faulty entity in the program in its recommendation list by evaluating the program spectra obtained from executing V over a test suite $T = \{t_1, \ldots, t_n\}$, where t_i denotes the *i*th test case and *n* denotes the number of test cases in the suite.

A SBFL formula utilizes four variables (called *SBFL variables*) to compute a *suspiciousness score*, denoted as susp, of each program entity in the set $S = \{s_1, \ldots, s_m\}$ of all entities of the program V. The score, susp, is used as an indication of the fault-suspiciousness of the program entity s.

The four SBFL variables are as follows.

1) a_{ep}, the number of *p*assed runs in which s is *e*xecuted;
2) a_{ef}, the number of *f*ailed runs in which s is *e*xecuted;
3) a_{np}, the number of *p*assed runs in which s is *n*ot executed;
4) a_{nf}, the number of *f*ailed runs in which s is *n*ot executed.

Here, a *run* refers to the execution of a test case of the program.

We chose statements as the program entities for empirical evaluation, but our methodology is applicable to other choices of entities such as branch [69] or object code instruction [54].

The suspiciousness scores are ranked such that a *higher rank* (which has a *smaller rank value*) indicates that the entity is more suspicious to correlate to a fault. In empirical studies, the faulty entity is known and its rank value is usually used to compose a metric of the formula's fault localization accuracy.

The term *accuracy* has different meanings in various contexts [1], [8]. In this paper, it is used strictly as a measure of the precision of a formula in recommending a true positive (faulty statement). It is defined as follows, whereby a smaller accuracy value indicates a higher precision in fault localization.

$$\text{accuracy} = \frac{\text{Smallest rank value of the faulty statement(s)}}{\text{Total number of statements in the faulty version}}$$

For instance, the SBFL formula, Naish2 [41], is defined as

$$\text{susp}_{\text{Naish2}} = a_{ef} - \frac{a_{ep}}{a_{ep} + a_{np} + 1}$$

Fig. 1 shows a code excerpt of the *flex* program version 1 [13]. The excerpt consists of six executable source statements $(s_1 - s_6)$ in the function readin(), in which s_3 is the faulty statement. The correct implementation for s_3 is as follows:

```
if ((fulltbl || fullspd) && reject).
```

	Source Code
s_1	`if ( variable_trailing_context_rules )`
s_2	`  reject = true;`
s_3	`if ( fulltbl || (fullspd && reject) ) // faulty`
	`  {`
s_4	`  if ( real_reject )`
s_5	`  flexerror( "..." );`
	`  else`
s_6	`  flexerror( "..." );`
	`  }`

Fig. 1. Code excerpt of the *flex* program version 1 (lines 847–861) [13]. Note: For clarity, blank line and compiler directives are not shown here.

TABLE I
Example SBFL Variables, Suspiciousness Scores, and Rank Values

Statement	Statement Coverage						SBFL Variables				Suspiciousness Score	Rank Values		
S	t_1	t_2	t_3	t_4	t_5	t_6	a_{ef}	a_{nf}	a_{ep}	a_{np}	$\text{susp}_{\text{Naish2}}$	R_w	R_b	R_a
s_1	1	1	1	1	1	1	2	0	4	0	1.2	2	2	2
s_2	1	0	0	1	0	0	0	2	2	2	–0.4	6	5	5.5
s_3	1	1	1	0	1	1	2	0	3	1	1.4	1	1	1
s_4	1	0	1	0	0	1	1	1	2	2	0.6	4	4	4
s_5	1	0	1	0	0	0	0	2	2	2	–0.4	6	5	5.5
s_6	0	0	0	0	0	1	1	1	0	4	1.0	3	3	3
	p	p	p	p	f	f								

Note: The first column refers to the statements $s_1 - s_6$ in Fig. 1. The bottom row shows the test verdict of the six test cases $t_1 - t_6$ (p = passed, f = failed). The rank values were assigned by three ranking schemes R_w, R_b, and R_a, that represent the *w*orst case, *b*est case, and *a*verage case scenarios, respectively.

Table I illustrates the use of the formula Naish2 to compute the suspiciousness scores and rank values of the executable statements in the code excerpt. The first column refers to the statements $s_1 - s_6$. The next six columns show the coverage statistics of the statements obtained from executing the six test cases $t_1 - t_6$ whereby, a cell having the value 1 (respectively 0) indicates that the corresponding statement is covered (respectively *not* covered) by each test case. The next five columns show, respectively, the computed values of the four SBFL variables and the suspiciousness score. The bottom row shows the test verdict of the six test cases $t_1 - t_6$, where "p" denotes a passed run and "f" denotes a failed run.

Consider s_3, which is exercised by two failed (t_5 and t_6) and three passed (t_1, t_2, and t_3) test cases. The values of a_{ef}, a_{nf}, a_{ep}, and a_{np} for s_3 are 2, 0, 3, and 1, respectively, and the suspiciousness score of s_3 is computed as 1.4. The suspiciousness scores of the other statements are computed in the same way as shown in Table I. All the entities in the program are then assigned *rank values* based on their suspiciousness scores by using a *ranking scheme* [40]. In this example, s_2 and s_5 have the same suspiciousness score of –0.4. The three rightmost columns in Table I show the rank values R_w, R_b, and R_a of $s_1 - s_6$ assigned by three ranking schemes, respectively, which differ only when assigning ranks to entities of equal scores, as explained below.

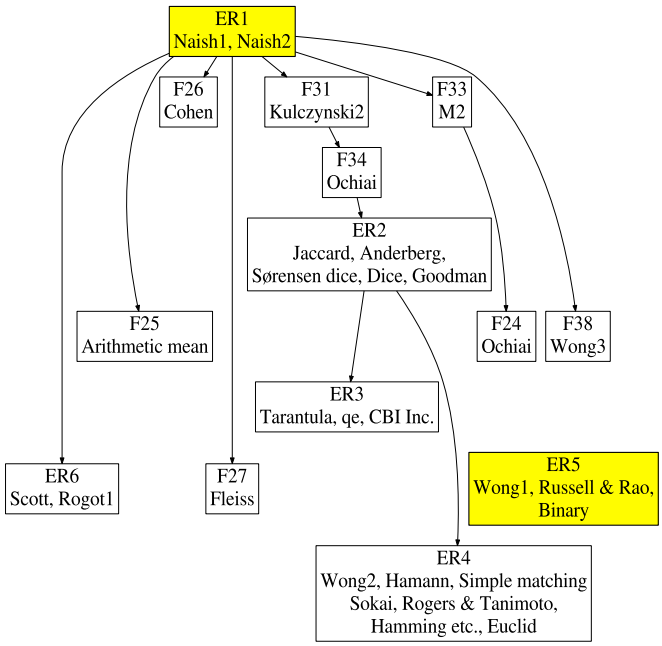


Fig. 2. Accuracy graph G_{theory} of relations discovered by Xie *et al*. [63].

In Table I, R_w, R_b, and R_a denote the rank values assigned by the *modified competition ranking scheme* (or the "1-3-3-4" scheme), the *standard competition ranking scheme* (or the "1-2-2-4" scheme), and the *fractional ranking scheme* (or the "1-2.5-2.5-4" scheme) [40], respectively. The three ranking schemes assign the largest possible, smallest possible, and mean rank value, respectively, to every element of the same value (causing a "tie") in the ordered list. In some SBFL empirical studies, these ranking schemes have been used for assigning ranks to program entities to represent, respectively, the *worst case, best case*, and *average case* scenarios [59] in computing the accuracy of formulas, hence the notations R_w, R_b, and R_a.

B. Accuracy Relations Proved by Theoretical Analysis

Xie *et al*. [63] reported a theoretical analysis on 30 formulas (the XCKX formulas) under certain assumptions. They proved the following two key results mathematically. For every possible program that contains a single fault, that is, a fault solely resided in a single program entity, 1) some formulas always assign either the same or a smaller rank value to the faulty program entity s than some other SBFL formulas, and 2) there exist groups (called *ER groups*) of SBFL formulas such that each group contains multiple formulas and, within each group, all formulas assign the same rank value to s. The first result establishes the relative ordering of the accuracy values of most of the XCKX formulas, and the second result partitions the set of XCKX formulas into equivalence classes (either ER groups or individual formulas) in terms of accuracy values. As mentioned in Section I, we refer to these ordering and equivalence relations together as *TRAO relations* for brevity.

Specifically, Xie *et al*. [63] identified six ER groups, referred to as ER1–ER6, respectively. All the TRAO relations they discovered can be depicted in the form of an *accuracy graph*, which we denote as G_{theory}, as shown in Fig. 2.

TABLE II
SBFL FORMULAS STUDIED IN OUR EXPERIMENTS

XCKX Formulas					
ID	**Name**	**Group**	**ID**	**Name**	**Group**
F_1	Naish1	$ER1_a$	F_{11}	Wong2	ER4
F_2	Naish2	$ER1_b$	F_{12}	Hamann	
F_3	Jaccard	ER2	F_{13}	Simple matching	
F_4	Anderberg		F_{14}	Sokal	
F_5	Sørensen dice		F_{15}	Rogers & Tanimoto	
F_6	Dice		F_{16}	Hamming etc.	
F_7	Goodman		F_{17}	Euclid	
F_8	Tarantula	ER3	F_{24}	Ample2	
F_9	q_e		F_{25}	Arithmetic mean	
F_{10}	CBI Inc.		F_{26}	Cohen	
F_{18}	Wong1	$ER5_a$	F_{27}	Fleiss	
F_{19}	Russell & Rao		F_{31}	Kulczynski2	
F_{20}	Binary	$ER5_b$	F_{33}	M2	
F_{21}	Scott	ER6	F_{34}	Ochiai	
F_{22}	Rogot1		F_{38}	Wong3	
***Non*-XCKX Formulas**					
ID	**Name**		**ID**	**Name**	
F_{23}	Ample		F_{36}	Overlap	
F_{28}	Geometric mean		F_{37}	Rogot2	
F_{29}	Harmonic Mean		F_{39}	Zoltar	
F_{30}	Kulczynski1		F_{40}	CBT	
F_{32}	M1		F_{41}	D^2	
F_{35}	Ochiai2		F_{42}	D^3	

Each node of G_{theory} contains either an individual XCKX formula or an ER group of two or more XCKX formulas. In Fig. 2, each node of an individual formula is labeled by its name and a unique identifier (e.g., "Cohen" and "F26," respectively), and each node of an ER group is labeled by its identifier (e.g., "ER1") and the names of its member formulas (e.g., "Naish1, Naish2"). For simplicity of language, hereafter when discussing accuracy graphs, we shall refer to the content of a node simply as a "formula," which may actually be a single SBFL formula or a group of SBFL formulas having equal or similar accuracy. A directed edge $\rightarrow$ from node N_1 to node N_2 (denoted as $N_1 \rightarrow N_2$) in G_{theory} means that for any formula F_i in N_1 and F_j in N_2, F_i is never less accurate than F_j. Table II includes a complete list of the 30 XCKX formulas, their identifiers, names, and ER groups.

The TRAO relations in G_{theory} shows the *relative accuracy* between any two XCKX formulas in a theoretical context. If there is no TRAO relation from node N_1 to node N_2, then the formulas in N_1 have not been proven to be more accurate than (or equally accurate as) those in N_2. For example, no XCKX formula is known to be always not less accurate than those in ER5, as seen from Fig. 2 that ER5 has no incoming edge.

C. SBFL Assumptions

To render their theoretical analysis tractable, Xie *et al*. [63] made a number of simplifying assumptions, many of which are fundamentally required for the SBFL technique or for many

SBFL formulas to be meaningful or feasible. But on the other hand, some of the assumptions may not hold in practice. One of the strongest assumptions is that all failures are due to the presence of one and only one fault in the program, and the fault resides in one and only one program entity that is being ranked via a formula. Another assumption requires at least one passed test case and one failed test case in the test suite. Some formulas will become undefined if there is no passed test case or failed test case. Nevertheless, previous work [67] has shown that even when only failed test cases are present, some formulas can still produce a similar accuracy compared with the presence of additional passed test cases. On the other hand, the absence of failed test case indicates that there is no detected failure and, thus, no debugging is needed. Chen *et al*. [7] further presented a comprehensive discussion on the various explicit and implicit assumptions as well as other practical issues for the theoretical analysis in [63] to be valid and consistent with empirical evaluation results. Furthermore, Abreu *et al*. [1] have shown in their experiment that the difference in accuracy of some SBFL formulas could be small if fewer test cases were used. A thorough treatment of the assumptions and issues is beyond the scope of this paper. Interested readers are referred to the papers [1], [7], [41], [62], and [63] for more details.

We have asked a question in Section I on what G_{theory} would look like when relaxing certain assumptions. In this work, we are interested in two specific assumptions. First, both the mathematical proofs in [63] and the model for analysis in [41] have only assumed the single-fault scenario. This assumption is popularly made in many prior empirical studies, while it has also been acknowledged to be unrealistic. Although some prior empirical work (such as [12]) reports that the issue of multiple faults in the same program can be addressed to a certain extent, a full picture on the relationships among nonsingle faults, ERAO relations, and the accuracy graph G_{theory} is still missing.

Second, the program spectra of crash runs collected are affected by the underlying platform. Some profiling tools can only collect the coverage information of a failed execution up to the last visited Ball–Larus path [6] – which is an acyclic path of a program under path profiling. These tools may miss the coverage on that path if the execution crashes. Some other tools can collect up to the last entity being executed. For instance, if the current entity is the only one following the last executed entity, then the faulty program entity can always be identified; but not otherwise. Thus, the presence of crash run may violate the requirement of $a_{nf} = 0$ for the faulty program entity in theoretical analyses [63] as well as the common SBFL assumption that "every failed test case executes at least one fault whose execution causes the failure" [53].

Besides assumptions, the theoretical analysis in [63] is also limited by its scope of having proved only the TRAO relations among the 30 XCKX formulas. In principle, G_{theory} can be expanded by analyzing more formulas mathematically to reveal more TRAO relations, yet speculating which potential TRAO relations might exist and which would be promising candidates for analytical proofs to be successfully constructed is not easy. We have also asked in Section I for a feasible and replicable methodology to both validate and expand G_{theory}, thereby opening up a pathway and paving a bridge to the more difficult task of theoretical analysis with relaxed assumptions and/or expanded scope to analyze more formulas. In any case, the results of this study alone can inform us the ERAO relations under less restrictive and more realistic assumptions and complement our knowledge of the TRAO relations. To the best of our knowledge, no prior empirical study has achieved these goals nor produced any graphs that are directly comparable to the one proved in [63].

III. Empirical Framework and Methodology

In this section, we present our empirical framework and methodology for accuracy graph construction.

A. Benchmark Suite

We selected the following 17 subject programs. Twelve subject C programs were downloaded from SIR [13]: seven programs $(P_1 - P_7)$ in the Siemens suite, *space* (P_8), four UNIX utility programs *flex*, *grep*, *gzip,* and *sed* ($P_9 - P_{12}$, respectively). The other five subject programs were Java packages downloaded from Defects4J [28]: *JFreeChart*, *Closure Compiler*, *Commons Math*, *Joda-Time*, and *Commons Lang* ($P_{13} - P_{17}$, respectively). The Siemens suite and UNIX utility programs are seeded-fault subjects, while the others contain real faults. The faulty statement(s) in each faulty version of a program were determined by comparing the difference between its source code and that of the bug-fixed version of the same program.

For subjects downloaded from SIR, we also downloaded the test pool and 1000 test suites of each of the Siemens suite programs and the *space* program from SIR. According to [13], these test suites were constructed by iteratively adding test cases (randomly from the test pool) to each individual test suite in attempting to exercise uncovered branches [13]. The UNIX utilities were downloaded with test pools but no test suite was available from SIR. We developed a tool to generate 1000 test suites for each UNIX utility subject program using the same procedure as described in SIR above except that it attempted to exercise uncovered statements rather than branches, as our study took statements as program entities. That is, our tool generated each test suite by randomly picking test cases one by one from the test pool until achieving the same statement coverage as the accumulated statement coverage of the entire test pool.

For subjects downloaded from Defects4J, they were real Java programs containing real faults. Each subject was packaged with multiple classes, and every class in the Java package was associated with its corresponding JUnit test class. Although the JUnit test cases were designed to test the individual units of the programs, when they were executed, not only the program units were covered, but many other parts of the programs beyond the units were also covered. In the benchmarks, most classes were not faulty. Nonetheless, for debugging purpose, we did not know which classes were responsible to fail a test case in a JUnit test class. Therefore, in SBFL assessment, we included the executable statements in all these application classes as potential candidates of faulty statements.

TABLE III
BENCHMARK SUBJECT PROGRAMS

Language	Subject Program Identifier (ID)	Subject Program Name	LOC[1]	SLOC[2]	No. of Faulty Versions Downloaded	Total No. of Faulty Versions Analyzed	No. of Test Cases	Real Fault	Seeded Fault
C	P_1	printtokens	563–564	341–343	7	7	4130		✓
C	P_2	printtokens2	504–511	350–355	10	10	4115		✓
C	P_3	replace	562–566	508–515	32	29	5542		✓
C	P_4	schedule	411–414	290–294	9	9	2650		✓
C	P_5	schedule2	307–308	261–263	10	9	2710		✓
C	P_6	tcas	172–179	133–137	41	40	1608		✓
C	P_7	totinfo	406	272–273	23	23	1052		✓
C	P_8	space	9114–9129	5882–5904	38	34	13585	✓	
C	P_9	flex	12423–14244	8518–10124	81	45	567		✓
C	P_{10}	grep	12653–13372	8530–9088	57	19	809		✓
C	P_{11}	gzip	6576–7996	4083–5159	59	17	214		✓
C	P_{12}	sed	6671–11990	4751–9287	32	24	360–370		✓
Java	P_{13}	JFreeChart	187775–230135	78564–96583	26	23	2205	✓	
Java	P_{14}	Closure Compiler	99385–170260	59026–104412	133	119	7927	✓	
Java	P_{15}	Commons Math	28033–186609	9471–84317	106	96	3602	✓	
Java	P_{16}	Joda-Time	66098–68712	26589–27795	27	24	4130	✓	
Java	P_{17}	Commons Lang	46813–61289	16593–21779	65	62	2245	✓	

[1] LOC refers to the number of Lines Of Code obtained using Linux command `find`, with arguments "`-name '*.java' | xargs wc -l`"
[2] SLOC refers to the number of Source Lines Of Code obtained using SLOCCount [58].

There were JUnit test classes that contained no failed test case when executed. Our experiment excluded these JUnit test classes and their induced statistics in the data analysis.

To investigate the influence of using different amounts of test cases on the reproduction of ERAO relations, we varied the size of the test suites by randomly picking 75%, 50%, and 25% of the test cases from the original test suites while ensuring that the resultant smaller test suites still satisfied the assumptions made in constructing the original test suites, with the only exception of their extents of statement/branch coverage.

To maintain compatibility with the assumptions made in G_{theory} and to avoid some SBFL formulas to become undefined, our experiment chose to ensure that every test suite included at least one passed test case and one failed test case. For the same reason, we also excluded every faulty version for which all its associated test suites contained no failed test cases. We further excluded some versions whose faulty entities could not be localized, such as omitting an entire method that could not be referred by the surrounding entities. In the end, the *full dataset* of our experiment included 127 faulty versions in the Siemens suite, 34 faulty versions of the *space* program, 105 faulty versions of the UNIX utility programs, and 324 faulty versions of the Java programs, together with all their corresponding applicable test suites (except those excluded due to the reasons explained above). Table III shows the descriptive statistics of the subject programs and test suites.

B. SBFL Formulas

It is impractical for an experiment to study an exhaustive list of formulas. Our empirical framework is generic enough to incorporate new formulas as needed. To demonstrate the potential of our framework, our study included all the formulas analyzed in the two recent theoretical studies [41], [63], as well as three recent formulas (CBT [61], D^2, and D^3 [60]) that were reported to have outstanding accuracy. In total, we studied 30 XCKX formulas and 12 *non*-XCKX formulas. We assigned an identifier to each formula (such as F_1 to stand for Naish1). A list of the identifiers (ID), names, and membership of ER group (where applicable) of all the 42 formulas studied are shown in Table II. The exact mathematical formulas and their sources of reference are listed in the online Appendix.

C. Environment and Tools

We performed data collection in a virtual machine configured with Intel Xeon CPU X5560 @ 2.8 GHz × 4, 16 GB physical memory and 1 TB disk storage. We used Ubuntu 12.04 LTS (64-bit) as the platform for executing the instrumented subject programs. To instrument C subject programs, we built a tool on *Pin* 4.13 [37] to probe the program entities before executing them to collect statement coverage up to program termination. We compiled the C subject programs by gcc 4.6.3 using the default (no optimization) option with an argument "`-gdwarf-2`" for injecting debugging information. We executed the Java subject programs under the configuration that came with the Defects4J framework, including Java 1.7.0_55, Perl 5.18.2, Git 1.9.1, and SVN 1.8.8.

We performed data analysis in a virtual machine with the same configuration as above but with MS Server installed. We implemented the programs for compiling the empirical data and various other automation codes in C# using VS 2012 R2. We

TABLE IV
DATASETS USED TO CONSTRUCT ACCURACY GRAPHS

Dataset	D_1	D_2	D_3	D_4
XCKX-formula graph	–	G_{base}	G_{crash}	$G_{\text{nonsingle}}$
Expanded graph	G_{typical}	G_{exp}	$G_{\text{exp\_crash}}$	$G_{\text{exp\_nonsingle}}$

Note: D_1 is the *full dataset* regardless of whether the required assumptions are satisfied. D_2 is the *base dataset* which is the subset of D_1 satisfying all the assumptions of G_{theory}. D_3 is the subset of D_1 satisfying all the assumptions except that execution of its test suites would cause *crash* runs. D_4 is the subset of D_1 satisfying all the assumptions except that its faults are *nonsingle*.

analyzed the data using MATLAB R2104b (8.4.0.150421) and rendered the accuracy graphs using Graphviz 2.38 [15].

D. Datasets for Construction of Accuracy Graphs

Recall that we intended to construct accuracy graphs that satisfy all or all but one of the assumptions, and then expand the graphs with *non*-XCKX formulas. We refer to an accuracy graph that contains XCKX formulas only as a *XCKX-formula graph* or an *unexpanded graph*, and one that includes both XCKX and *non*-XCKX formulas as an *expanded graph*.

To begin with, we should first find a way of empirical evaluation that assesses the accuracy of XCKX formulas, compares their relative accuracy to extract ERAO relations, and then compares the latter with the TRAO relations in G_{theory}. As a result, we constructed three XCKX-formula graphs and four expanded graphs that display the ERAO relations of formulas. Table IV summarizes the constituent datasets of these accuracy graphs.

The first graph G_{base} was constructed from the "base" dataset D_2 that satisfied all the assumptions of G_{theory}. Specifically, to align with the theoretical study [63] that created G_{theory}, all program versions in D_2 can be executed by each test case in their test suites in D_2 without crash runs and every fault in every faulty program version resided in *one single statement only*, bearing in mind that we used statement as the program entity.

The second XCKX-formula graph G_{crash} was constructed from the dataset D_3 that satisfied all the assumptions except that all the test suites in D_3 included crashing runs. The third XCKX-formula graph $G_{\text{nonsingle}}$ was constructed from the dataset D_4 that satisfied all the assumptions except that all the faults in D_4 were *nonsingle*. To clearly distinguish a *single* fault from a *nonsingle* fault, we define the following criteria for classifying a fault as *single fault*: 1) the fault resides in a single program entity (which in this study refers to a single line in the source code file), and 2) the entity itself does not contain compound entities (such as a line consisting of multiple statements which may cause short-circuit evaluation). A faulty entity that satisfies these two criteria is considered as a *single* fault; otherwise, it is a *nonsingle* fault.

We note that Xie *et al*. [63] explicitly assumed the absence of omission faults in their theoretical analysis. None of the datasets $D_2 - D_4$ contained faulty versions with omission faults.

The three expanded graphs G_{exp}, $G_{\text{exp\_crash}}$, and $G_{\text{exp\_nonsingle}}$, which correspond to G_{base}, G_{crash}, and $G_{\text{nonsingle}}$, were constructed by evaluating both XCKX and *non*-XCKX formulas based on the datasets D_2, D_3, and D_4, respectively. Finally, the expanded graph G_{typical} was constructed from the full dataset D_1 which contains all valid combinations of subjects (faulty program versions and test suites) regardless of whether the assumptions are satisfied or not.

E. Data Analysis and Accuracy Graph Construction

We executed each faulty version of each subject with each test case and extracted the coverage profile. For each test suite of each faulty version, we applied each SBFL formula to compute the suspiciousness scores of all the statements of the faulty version. We applied each of the three ranking schemes (see Section II-A) to compute a rank value for each faulty statement and then the accuracy of that SBFL formula.

To choose appropriate statistical tools for comparing the accuracy of formulas, we had performed a normality test to the accuracy values computed from the formulas and the datasets (D_1 to D_4). The hypothesis of a normal distribution was rejected at the 5% significance level. Hence, we chose to use the *Kruskal–Wallis test*, which is a nonparametric counterpart of the classical *one-way analysis of variance* (ANOVA), to determine whether the accuracy values computed by different formulas and datasets came from the same population. The *multiple comparison test* [22] was then used in conjunction with the Kruskal–Wallis test to compare all the studied formulas together to determine which groups of them were statistically similar or different. *Bonferroni correction* was used to compensate the familywise error of both tests. All such analysis tasks were accomplished with MATLAB.

To describe the groupings of formulas, we used a popular community function of MATLAB called *my_ph_letters* [3] that converted results of multiple comparison tests into *letter groups*, that is, a list (string) of English alphabets representing the relative order of the accuracy of the formulas. For instance, letter "A" refers to the group of the most accurate formulas, letter "B" refers to the next most accurate group, and so on. A formula may belong to a group *labeled* by a single letter (e.g., a group with label "A" containing the most accurate formulas) or a string of two or more consecutive letters (such as "DEF") in alphabetical order. We refer to such a label as *letter label*. Groups of formulas that share a letter in their labels are said to be *similar*, meaning that their differences of accuracy values are (statistically) insignificant, whereas the accuracy values of groups that do not share a letter in their labels are (statistically) significantly different (From hereon, to avoid verbosity, the word "statistically" will be omitted when it is clear from the context). For example, a formula with the letter label "CD" is significantly less accurate than another one with the letter label "B," but the former is similar in accuracy to the formula whose letter label is "DEF" which shares the letter "D" with "CD."

From the letter labels of all formulas, we determined the *accuracy relation* between two formulas $f1$ and $f2$ *with respect to a subject program and a ranking scheme* as follows.

1) $f1$ and $f2$ are in the same letter group, denoted as $f1 \equiv f2$, if the letter labels of $f1$ and $f2$ are exactly the same. In this case, $f1$ and $f2$ are *similar in accuracy*.

```
Function: Compare
   Input   f1, f2: two SBFL formulas
           p: a subject program
           r: a ranking scheme
   Output  a: the accuracy relation between f1 and f2 with respect to
             p and r
   Begin
 1 L1, L2 := GetLetterLabel(f1, f2, p, r)
 2 b1, b2 := the respective begin letter of L1 and L2
 3 e1, e2 := the respective ending letter of L1 and L2
 4 IF L1 = L2 THEN a := '≡'
 5 ELSE IF e1 < b2 THEN a := '>'
 6      ELSE IF b1 > e2 THEN a := '<'
 7           ELSE a := '≈'
   End
```

```
Function: Extract ERAO relations
   Input   F: the set of SBFL formulas
           P: the set of 17 subjects programs
           R: the set of 3 ranking schemes
   Output  E: the set of ERAO relations with respect to P and R
   Begin
 1 E := ∅
 2 FOREACH fx ∈ F DO BEGIN
 3   FOREACH fy ∈ F \ { fx } DO BEGIN
 4     A := ∅   // set of accuracy relations of fx and fy with
                   respect to P and R
 5     FOREACH p ∈ P, r ∈ R DO BEGIN
 6       a := Compare(fx, fy, p, r)
 7       A := A ∪ { a }
 8     ENDFOR
 9     IF ('>' ∉ A) AND ('<' ∉ A)
10     THEN E := E ∪ {(fx ~ fy)}
11     ELSE IF ('>' ∈ A) AND ('<' ∉ A)
12          THEN E := E ∪ {(fx → fy)}
13          ELSE IF ('<' ∈ A) AND ('>' ∉ A)
14               THEN E := E ∪ {(fx ← fy)}
15               ELSE E := E ∪ {(fx △ fy)}
16   ENDFOR
17 ENDFOR
   End
```

Fig. 3. Functions for extracting accuracy relations and ERAO relations. Note: The function **GetLetterLabel** retrieves the respective letter label of $f1$ and $f2$ with respect to p and r from MATLAB using the community function *my_ph_letters* [3] as explained in Section III-E of this paper.

2) $f1$ is *more accurate* than $f2$, denoted as $f1 > f2$, if all letters in the label of $f1$ precede all those of $f2$, or equivalently, the ending letter of the label of $f1$ precedes the begin letter of the label of $f2$, e.g., when the labels of $f1$ and $f2$ are "A" and "BC," respectively.
3) $f1$ is *less accurate* than $f2$, denoted as $f1 < f2$, if $f2$ is more accurate than $f1$.
4) Otherwise, if none of the above three cases apply, then $f1$ and $f2$ are not in the same letter group but are still *similar in accuracy* (as they share at least one letter in their labels and, hence, their difference in accuracy is statistically insignificant), denoted as $f1 \approx f2$, e.g., when their labels are "BC" and "CD," respectively.

Fig. 3 shows the function, **Compare**, to implement the above four cases for determining the accuracy relation between two formulas $f1$ and $f2$ with respect to a subject program p and a ranking scheme r, based on the letter labels returned by the function **GetLetterLabel** which retrieves them from MATLAB using the community function *my_ph_letters* [3] as explained earlier in this section.

There were 17 subject programs and 3 ranking schemes, yielding 51 accuracy relations for each formula pair and dataset. By comparing the accuracy relations of each formula pair in each dataset, we went on to analyze whether a formula in the pair was, *across all* 17 *subject programs and all three ranking schemes*, *consistently similar to* or *consistently more accurate than* the other, or otherwise.

Two formulas $f1$ and $f2$ are said to be *consistently similar* (in accuracy), denoted as $f1 \sim f2$, if they are *similar* in accuracy to each other (that is, $f1 \equiv f2$ or $f1 \approx f2$) with respect to *all* subject programs and *all* ranking schemes or, equivalently, $f1$ is *neither more* accurate *nor less* accurate than $f2$ (that is, $f1 \not> f2$ and $f1 \not< f2$) with respect to *any* subject program and *any* ranking scheme.

A formula $f1$ is said to be *consistently more accurate* than another formula $f2$, denoted as $f1 \rightarrow f2$, if (a) $f1$ is *more* accurate than $f2$ (that is, $f1 > f2$) with respect to at least one subject program and one ranking scheme, and (b) $f1$ is *not less* accurate than $f2$ (that is, $f1 \not< f2$) with respect to *any* subject program and *any* ranking scheme. In such a case, we also say that $f2$ is *consistently less accurate* than $f1$, denoted as $f2 \leftarrow f1$.

Two formulas $f1$ and $f2$ are said to have an ERAO relation with each other if either one of the following is true: ($f1 \rightarrow f2$) or ($f1 \leftarrow f2$) or ($f1 \sim f2$). On the other hand, if none of these is true, then $f1$ and $f2$ have no ERAO relation, denoted as $f1 \,\Delta\, f2$.

Fig. 3 also shows the function, **Extract ERAO relations**, for extracting all ERAO relations for further data analysis.

The ERAO relations are our empirical analog of the TRAO relations developed in the theoretical analysis of Xie *et al*. [63]. To ease comparison, we overload the same notations $\rightarrow$, $\leftarrow$, and $\sim$ to represent the corresponding TRAO relations even though the latter were not defined in statistical terms. When all the ERAO relations with respect to a dataset are extracted, an empirical accuracy graph can be constructed in a similar way that G_{theory} can be constructed from the TRAO relations discovered in [63], as elaborated toward the end of this section.

Fig. 4 illustrates the construction of an accuracy graph by means of a small example. Fig. 4(a) shows a table of the letter labels of seven formulas $(F_a - F_g)$ with respect to six subject programs $(P_u - P_z)$ and a particular ranking scheme (for simplicity, this example considers only one ranking scheme instead of all three). The two functions shown in Fig. 3 can be used to transform the table of letter labels in Fig. 4(a) to the table of ERAO relations in Fig. 4(b), from which the accuracy graph shown in Fig. 4(c) can then be constructed, as detailed below.

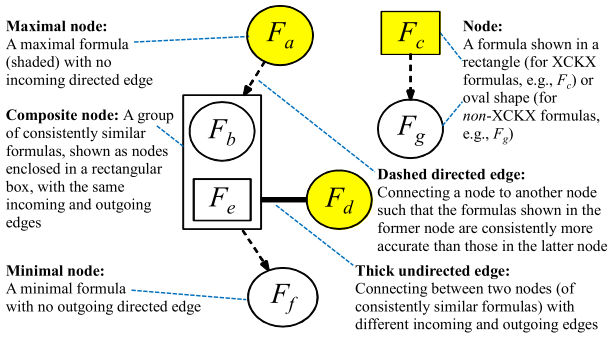
For instance, consider the formulas F_a and F_b. With respect to P_u, the letter labels of F_a and F_b are "A" and "BC," respectively. Now we have $F_a > F_b$ because "A" precedes "BC." Similarly, with respect to P_w and P_x, we also have $F_a > F_b$. On the other hand, with respect to P_v, we have $F_a \approx F_b$ because the letter labels of F_a and F_b are "BC" and "AB," respectively, which

Formula	Subject Program					
	P_u	P_v	P_w	P_x	P_y	P_z
F_a	A	BC	A	A	AB	B
F_b	BC	AB	B	BC	AB	B
F_c	B	A	B	B	C	A
F_d	C	A	B	BC	B	B
F_e	C	AB	B	C	AB	B
F_f	D	BC	C	D	A	C
F_g	B	B	B	B	C	A

(a)

Formula	F_a	F_b	F_c	F_d	F_e	F_f	F_g
F_a		→	Δ	Δ	→	→	Δ
F_b	←		Δ	~	~	→	Δ
F_c	Δ	Δ		Δ	Δ	Δ	→
F_d	Δ	~	Δ		~	Δ	Δ
F_e	←	~	Δ	~		→	Δ
F_f	←	←	Δ	Δ	←		Δ
F_g	Δ	Δ	←	Δ	Δ	Δ	

(b)



(c)

Fig. 4. Example for constructing an accuracy graph of formulas*. (a) Letter groups of formulas with respect to subject programs. (b) The ERAO relations extracted from the table in (a). (c) Legend for Fig. 5: the accuracy graph constructed from the table in (b).
Note: ∼ means that the two formulas are consistently similar. → means that the formula at the left (row label) is consistently more accurate than the formula at the top (column label). ← means that the formula at the left is consistently less accurate than the formula at the top. Δ means that no consistent ERAO relation can be established between the formulas at the left and the top.
*For simplicity of language, the term "formula" refers to either an individual SBFL formula (e.g., F_{33}) or an ER group of formulas (e.g., ER2).

share the letter "B" but are not identical. Finally, with respect to P_y and P_z, we have $F_a \equiv F_b$ because their letter labels are identical. Thus, the set of accuracy relations between F_a and F_b is $\{>, \approx, \equiv\}$, which contains $>$ but not $<$. Therefore, $F_a \rightarrow F_b$. Thus, in Fig. 4(b), the symbol "→" is filled in the cell that intersects the row F_a and the column F_b, and the symbol "←" is filled in the cell that intersects the row F_b and the column F_a.

As another instance, consider the formulas F_a and F_c. With respect to P_u, their letter labels are "A" and "B," respectively. Now we have $F_a > F_c$ because "A" precedes "B." Similarly, with respect to $P_w - P_y$, we also have $F_a > F_c$, but with respect to P_v and P_z, we have $F_a < F_c$. Thus, the set of accuracy relations between F_a and F_c is $\{<, >\}$. As there is no consistent accuracy relation, we have $F_a \,\Delta\, F_c$. Thus, in Fig. 4(b), the symbol "Δ" is filled in the cell that intersects the row F_a and the column F_c, as well as the cell that intersects the row F_c and the column F_a. Other cells in Fig. 4(b) can be filled in a similar way.

From the table in Fig. 4(b), we can construct the accuracy graph in Fig. 4(c) as follows. We first construct a *simple node* to represent each individual formula, using a rectangular shape for an XCKX formula node and an oval shape for a *non*-XCKX formula node. When a formula $f1$ is consistently more accurate than another formula $f2$ (that is, $f1 \rightarrow f2$), we construct a directed edge (shown as dashed arrow) from the node $f1$ to the node $f2$, such as the one from F_c to F_g. If a group of formulas (e.g., $\{F_b, F_e\}$) are consistently similar and all of their incoming and outgoing edges are the same, we enclose the formulas in a larger rectangle to form a *composite node*. If all the formulas in two nodes (such as the simple node F_d and the composite node containing $\{F_b, F_e\}$) are consistently similar but have different incoming or outgoing edge(s), we join the two nodes with a thick undirected edge. To avoid excessive edges that degrade graph readability, we perform transitive reduction [2] before rendering the graph. For instance, since $F_a \rightarrow F_b$ and $F_b \rightarrow F_f$, the edge that represents $F_a \rightarrow F_f$ is omitted in Fig. 4(c).

A formula in an accuracy graph [e.g., F_c in Fig. 4(c)] is called *maximal* if no other formula is consistently more accurate than it. A node is called *maximal* and shaded in yellow if it has no incoming directed edge. At the other extreme, a formula is called *minimal* [e.g., F_g in Fig. 4(c)] if it is *not* consistently more accurate than any other formula, and a node is *minimal* if it has no outgoing directed edge.

We extracted all ERAO relations among the XCKX formulas from datasets $D_2 - D_4$ to construct the XCKX-formula graphs G_{base}, G_{crash}, and $G_{\text{nonsingle}}$, respectively, and all ERAO relations among all the 42 formulas (30 XCKX and 12 *non*-XCKX formulas) from datasets $D_1 - D_4$ to construct the expanded graphs G_{typical}, G_{exp}, $G_{\text{exp\_crash}}$, and $G_{\text{exp\_nonsingle}}$, respectively.

IV. Analyzing the Accuracy Graphs

This section analyzes the accuracy graphs constructed from our experiment, summarizes our findings, and compares with prior SBFL empirical results.

Fig. 5 shows the seven accuracy graphs constructed from datasets $D_1 - D_4$: the three XCKX-formula graphs G_{base}, G_{crash}, and $G_{\text{nonsingle}}$ in subfigures (b'), (c'), and (d'), respectively, and the four expanded graphs G_{typical}, G_{exp}, $G_{\text{exp\_crash}}$, and $G_{\text{exp\_nonsingle}}$ in subfigures (a)–(d), respectively. The reader may refer to Fig. 4(c) which also serves as a legend for interpreting the shapes of nodes and edges of the graphs in Fig. 5.

A. Qualitative Analysis

1) Characteristics of XCKX-Formula Graphs: First, we examine the characteristics of XCKX-formula graphs. G_{base} was constructed from the dataset D_2. The ERAO relations in G_{base} depicted in Fig. 5(b') are highly consistent to G_{theory}. It shows

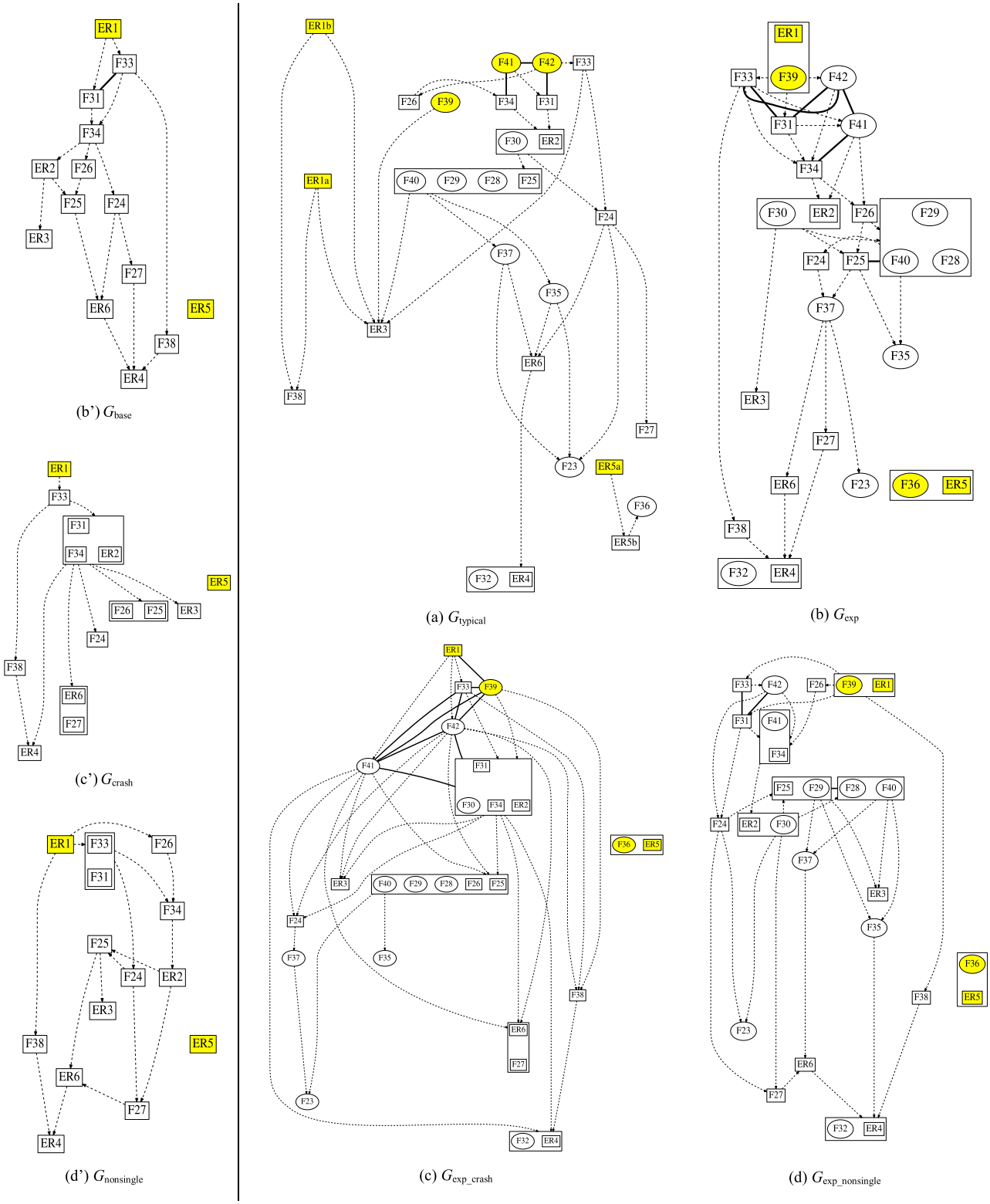


Fig. 5. Accuracy graphs constructed from this empirical study.
Note: Refer to Fig. 4(c) for the legend to the graphs in this figure. Graphs shown in (b), (c), and (d) were constructed from the same datasets as those for the graphs shown in (b'), (c'), and (d'), respectively, but the former were expanded to include *non*-XCKX formulas in the analysis while the latter included XCKX formulas only.

that through our methodology, we successfully constructed G_{base}, with G_{theory} as its proper subgraph. Like G_{theory}, G_{base} has two disjoint subgraphs with maximal nodes ER1 and ER5, respectively, and the subgraph of ER1 forms a *cluster* of connected nodes of all XCKX formulas except ER5. The least accurate formulas are in the group ER4. Furthermore, G_{base} contains 13 new edges that are not present in G_{theory}. The new edges are ER2 $\rightarrow F_{25}$, ER6 $\rightarrow$ ER4, $F_{24} \rightarrow$ ER6, $F_{24} \rightarrow F_{27}$, $F_{25} \rightarrow$ ER6, $F_{26} \rightarrow F_{25}$, $F_{27} \rightarrow$ ER4, $F_{31} \sim F_{33}$, $F_{33} \rightarrow F_{34}$, $F_{33} \rightarrow F_{38}$, $F_{34} \rightarrow F_{24}$, $F_{34} \rightarrow F_{26}$, and $F_{38} \rightarrow$ ER4. That is, the TRAO relations of G_{theory} have been enriched to G_{base} with previously unknown ERAO relations.

G_{crash} was constructed from the dataset D_3. Fig. 5(c') shows that, similar to G_{base}, G_{crash} has two subgraphs with maximal nodes ER1 and ER5, respectively. The least accurate formulas are also in the group ER4. Interestingly, while the layouts of G_{crash} and G_{base} look different, none of their edges is in the reverse direction of those in G_{theory}. In addition, G_{crash} contains 25 new edges that are not present in G_{theory}. They are ER2 $\rightarrow$ ER6, ER2 $\rightarrow F_{24}$, ER2 $\rightarrow F_{25}$, ER2 $\rightarrow F_{26}$, ER2 $\rightarrow F_{27}$, ER2 $\sim F_{31}$, ER2 $\sim F_{34}$, ER6 $\sim F_{27}$, $F_{25} \sim F_{26}$, $F_{31} \rightarrow$ ER6, $F_{31} \rightarrow F_{24}$, $F_{31} \rightarrow F_{25}$, $F_{31} \rightarrow F_{26}$, $F_{31} \rightarrow F_{27}$, $F_{31} \sim F_{34}$, $F_{33} \rightarrow$ ER2, $F_{33} \rightarrow F_{31}$, $F_{33} \rightarrow F_{34}$, $F_{33} \rightarrow F_{38}$, $F_{34} \rightarrow$ ER6, $F_{34} \rightarrow F_{24}$, $F_{34} \rightarrow F_{25}$, $F_{34} \rightarrow F_{26}$, $F_{34} \rightarrow F_{27}$, and $F_{38} \rightarrow$ ER4.

$G_{\text{nonsingle}}$ was constructed from the dataset D_4. Fig. 5(d') shows that $G_{\text{nonsingle}}$ has two subgraphs, one with maximal node ER1, and the other with maximal node ER5. Like G_{theory}, G_{base}, and G_{crash}, a cluster is formed with ER1 while ER5 forms a separate unconnected node, and the least accurate formulas are in group ER4. Again, G_{theory} is a proper subgraph of $G_{\text{nonsingle}}$. $G_{\text{nonsingle}}$ contains 13 new edges that are absent in G_{theory}, namely, ER2 $\rightarrow F_{25}$, ER2 $\rightarrow F_{27}$, ER6 $\rightarrow$ ER4, $F_{24} \rightarrow F_{25}$, $F_{24} \rightarrow F_{27}$, $F_{25} \rightarrow$ ER3, $F_{25} \rightarrow$ ER6, $F_{26} \rightarrow F_{34}$, $F_{27} \rightarrow$ ER6, $F_{31} \rightarrow F_{24}$, $F_{31} \sim F_{33}$, $F_{33} \rightarrow F_{34}$, and $F_{38} \rightarrow$ ER4.

> *Result 1a:* The three XCKX-formula graphs reveal new ERAO relations atop the TRAO relations of G_{theory}. Moreover, none of the ERAO relations is contradictory to the TRAO relations of G_{theory}. All these three graphs are indeed strikingly similar in revealing TRAO relations. They validate that the TRAO relations in G_{theory} appear to be robust with respect to the specific changes in assumption under study represented by these datasets, and the proven TRAO relations are empirically sound even when not all the assumptions hold.

2) Bridging Relative Accuracy Between Experiment and Theory: Next, we examine the extents of reproduction of TRAO relations (in G_{theory}) with ERAO relations in the empirical graphs. Two kinds of TRAO relations can be revealed from G_{theory}: 1) *direct* TRAO relations that are shown as directed edges (such as ER1 $\rightarrow$ ER6) in G_{theory}, and 2) *indirect* TRAO relations that can be inferred through the direct TRAO relations, such as ER1 $\rightarrow F_{24}$ inferred from ER1 $\rightarrow F_{33}$ and $F_{33} \rightarrow F_{24}$.

Table V shows whether the 14 direct and 10 indirect relations in G_{theory} are present in the empirical accuracy graphs. In the 2nd to 5th columns of the table, the symbol ○ indicates that the corresponding TRAO relation ($\rightarrow$ or $\sim$) in G_{theory} is observed in the empirical graph, ⊗ indicates that the TRAO relation ($\rightarrow$) in G_{theory} is observed as the "consistently similar" relation ($\sim$) in the empirical graph, and $\times$ indicates that the TRAO relation in G_{theory} "disappears" (i.e., is not found) in the empirical graph.

TABLE V
REPRODUCTION OF TRAO RELATIONS IN EMPIRICAL ACCURACY GRAPHS

Direct TRAO Relation	G_{typical}	G_{base} / G_{exp}	G_{crash} / $G_{\text{exp\_crash}}$	$G_{\text{nonsingle}}$ / $G_{\text{exp\_nonsingle}}$
$\text{ER1}_a \sim \text{ER1}_b$	×	○	○	○
$\text{ER5}_a \sim \text{ER5}_b$	×	○	○	○
ER1 → ER6	×	○	○	○
ER1 → F_{25}	×	○	○	○
ER1 → F_{26}	×	○	○	○
ER1 → F_{27}	×	○	○	○
ER1 → F_{31}	×	○	○	○
ER1 → F_{33}	×	○	○	○
ER1 → F_{38}	○	○	○	○
F_{31} → F_{34}	×	○	⊗	○
F_{34} → ER2	○	○	⊗	○
F_{33} → F_{24}	○	○	○	○
ER2 → ER3	○	○	○	○
ER2 → ER4	○	○	○	○
% of relations reproduced (○)	36%	100%	86%	100%
% of relations disappeared (×)	64%	0%	0%	0%

Indirect TRAO Relation	G_{typical}	G_{base} / G_{exp}	G_{crash} / $G_{\text{exp\_crash}}$	$G_{\text{nonsingle}}$ / $G_{\text{exp\_nonsingle}}$
ER1 → ER2	×	○	○	○
ER1 → ER3	○	○	○	○
ER1 → ER4	×	○	○	○
ER1 → F_{24}	×	○	○	○
ER1 → F_{34}	×	○	○	○
F_{31} → ER2	○	○	⊗	○
F_{31} → ER3	○	○	○	○
F_{31} → ER4	○	○	○	○
F_{34} → ER3	○	○	○	○
F_{34} → ER4	○	○	○	○
% of relations reproduced (○)	60%	100%	90%	100%
% of relations disappeared (×)	40%	0%	0%	0%

Note: ○ means that the TRAO relation (→ or ∼) is reproduced. ⊗ means that the TRAO relation (→) becomes the "consistently similar" relation (∼). × means that the TRAO relation disappears (is absent) in the empirical graphs.

G_{typical} was constructed from the full dataset D_1 that includes all valid combinations of faulty program versions and test suites regardless of whether the assumptions are satisfied or not. It was chosen to reflect the *typical* scenarios in practice where a myriad of realistic causes (such as the presence of omission faults and/or platform dependent failures) might arise violating one or more of the assumptions.

As seen from Table V, G_{typical} only reproduces 5 (=36% out of 14) direct TRAO relations and 6 (=60% out of 10) indirect TRAO relations while the remaining TRAO relations in G_{theory} are all absent in G_{typical}. The proportions of disappeared direct and indirect TRAO relations are 64% and 40%,

respectively. By contrast, both G_{base} and G_{exp} reproduce all the TRAO relations in G_{theory}. This finding provides encouraging supporting evidence that our framework and methodology of empirically replicating and expanding the TRAO relations as well as revealing new ERAO relations rests on a strong basis.

For the graphs constructed by relaxing a specific assumption, each of G_{crash}, $G_{\text{exp\_crash}}$, $G_{\text{nonsingle}}$, and $G_{\text{exp\_nonsingle}}$ reproduces 12–14 direct and 9–10 indirect TRAO relations, constituting 86–100% and 90–100% of the TRAO relations, respectively. The "disappeared" TRAO relations ($F_{31} \rightarrow F_{34}$, $F_{34} \rightarrow$ ER2, and $F_{31} \rightarrow$ ER2) actually become *consistently similar* relations ($F_{31} \sim F_{34}$, $F_{34} \sim$ ER2, and $F_{31} \sim$ ER2) in the empirical graphs. In other words, even though one formula in such a pair is theoretically proved to be never less accurate than the other, they are empirically found to be similar in statistical terms. This finding complements the theoretical analysis by providing a more precise picture of the relationships in practical scenarios.

Result 1b: The empirical accuracy graphs reproduce the TRAO relations of G_{theory} to different extents. G_{typical} *only* reproduces 36% direct and 60% indirect TRAO relations in G_{theory}; the others disappeared in our empirical accuracy graphs. G_{base} and G_{exp} reproduce *all* the TRAO relations of G_{theory}. The graphs constructed by relaxing a specific assumption, G_{crash} and $G_{\text{nonsingle}}$ (or $G_{\text{exp\_crash}}$ and $G_{\text{exp\_nonsingle}}$), reproduce 86–100% direct and 90–100% indirect TRAO relations, while the remaining TRAO relations become the "consistently similar" relation. The TRAO relations appear to be quite robust with respect to the violation of only one of the required assumptions (single fault or absence of crash run) of G_{theory}, which is interesting. However, most of the TRAO relations do not hold when multiple assumptions are violated.

Results 1a and 1b consolidate the underpinnings of our methodology on how additional formulas can be incorporated to expand accuracy graphs.

3) Characteristics of Expanded Graphs: We now examine the characteristics of the expanded graphs, beginning with G_{typical}.

First, we compare the ERAO relations of G_{typical} with the TRAO relations of G_{theory}. Fig. 5(a) shows that G_{typical} consists of two clusters of connected nodes with the maximal nodes ER1_{a}, ER1_{b}, ER5_{a}, F_{39}, F_{41}, and F_{42}. We observe that even if some formulas (in the same ER group) have been proved to be equivalent in accuracy, they may exhibit very different relative accuracy in typical scenarios, which is interesting.

ER1 in G_{typical} is one such instance. Its two member formulas Naish1 and Naish2 have significantly different accuracy values and they bear no ERAO relation. They are two separate nodes labeled as ER1_{a} and ER1_{b}, respectively, in G_{typical}. ER5 is another instance, which is split into two connected nodes ER5_{a} and ER5_{b} in G_{typical}. It again illustrates that the TRAO relations do not necessarily hold in typical scenarios. Furthermore, G_{typical} also reveals new ERAO relations that do not appear in G_{theory}.

We next examine the characteristics of the expanded graphs that supplement the existing theoretical framework with ERAO relations derived from both XCKX and *non*-XCKX formulas.

Fig. 5(b)–(d) shows G_{exp}, $G_{\text{exp\_crash}}$, and $G_{\text{exp\_nonsingle}}$, respectively. After incorporating 12 *non*-XCKX formulas, as expected, the graphs G_{base}, G_{crash}, and $G_{\text{nonsingle}}$ were expanded to include more nodes and edges in G_{exp}, $G_{\text{exp\_crash}}$, and $G_{\text{exp\_nonsingle}}$, respectively, rather than merging them into the existing nodes and edges. In comparing the unexpanded graphs with their expanded ones, we observe several common characteristics. In each of the unexpanded graphs, both ER1 and ER5 are the maximal nodes, while ER4 contains the least accurate formulas. In all the expanded graphs except G_{typical}, both ER1 and ER5 remain to be maximal and ER4 remains to be the least accurate group of formulas, but now ER1 is consistently similar to the *non*-XCKX formula F_{39}, ER5 is consistently similar to the *non*-XCKX formula F_{36}, and ER4 is consistently similar to the *non*-XCKX formula F_{32}. Besides, ER2 is consistently similar to the *non*-XCKX formula F_{30}.

4) Potential Theoretical Relations (TRAO or Beyond): We found 21 new ERAO relations that consistently appear in all the expanded graphs. Twelve of them involve pairs each containing an XCKX formula and a *non*-XCKX formula: ER2 $\sim F_{30}$, ER4 $\sim F_{32}$, $F_{25} \sim F_{28}$, $F_{25} \sim F_{29}$, $F_{25} \sim F_{40}$, $F_{31} \sim F_{42}$, $F_{34} \sim F_{41}$, ER2 $\rightarrow F_{28}$, ER2 $\rightarrow F_{29}$, ER2 $\rightarrow F_{40}$, $F_{25} \rightarrow F_{35}$, and $F_{30} \rightarrow F_{25}$. The other nine ERAO relations are between pairs of *non*-XCKX formulas: $F_{28} \sim F_{29}$, $F_{28} \sim F_{40}$, $F_{29} \sim F_{40}$, $F_{28} \rightarrow F_{35}$, $F_{29} \rightarrow F_{35}$, $F_{30} \rightarrow F_{28}$, $F_{30} \rightarrow F_{29}$, $F_{30} \rightarrow F_{40}$, and $F_{40} \rightarrow F_{35}$. These relations seem robust enough to "withstand" the change of the underlying datasets from closely satisfying the required assumptions in G_{exp} to violating some of them in G_{typical}.

The ERAO relations of each graph were grounded on hypothesis testing to ensure the pairs being statistically similar or different in accuracy. Such relations are important as they were found to *consistently* appear in all the expanded graphs. We conjecture that such formula pairs potentially have yet-to-be-discovered theoretical relations. It is thus worthy to set a higher priority to investigate their potential theoretical dependency, for instance, whether F_{30} and F_{32} are theoretically equivalent in accuracy to ER2 and ER4, respectively, or whether F_{39}, a maximal node of all graphs, is provably never less accurate than ER3 given that all graphs contain $F_{39} \rightarrow$ ER3.

In reality, knowing in advance whether the fault under debugging is single or not is out of the question. Moreover, a given test suite for SBFL may not be chosen by a tester to ensure no crash run (e.g., the only failed test is a crash run). The prior theoretical study [63] conjectures that if G_{theory} and G_{typical} are not very different, then researchers may focus on the comparison of *non*-XCKX formulas against formulas in ER1 and ER5 *without* bothering the assumptions of G_{theory}. Our results not only provide empirical evidence to support this conjecture, but also quantify which parts of the two graphs

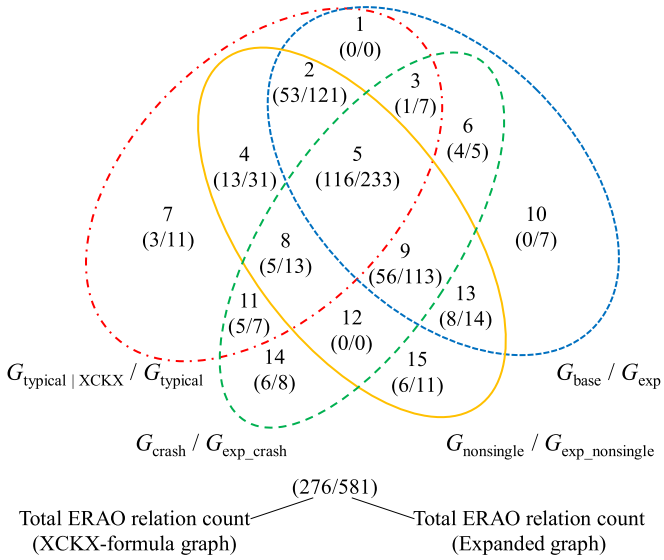
are consistently similar and which parts of them consistently differ.

Result 2: We have identified new ERAO relations which consistently appear in all expanded graphs. As these relations were obtained through statistical hypothesis tests of whether any two formulas have significantly different accuracy, we conjecture that these relations are statistically reliable. We recommend focusing on further investigation of these relationships, particularly those in connection with the existing theoretical framework.

Result 3: Given the Results 1a, 1b, and 2, we observe that $G_{\text{exp\_crash}}$ and $G_{\text{exp\_nonsingle}}$ (both constructed by relaxing one specific assumption) are strikingly similar to G_{exp} in retaining all or almost all of the TRAO relations of G_{theory}. Nevertheless, G_{typical} (built on datasets that might violate several assumptions) is very different from G_{exp} in its failure to preserve most of the TRAO relations. Since the relaxation of each of our two chosen specific assumptions does not cause such a big difference, it is worthy to further investigate whether the difference is caused by other factors such as omission fault or platform-dependent issue.

The TRAO relations in [41], [63] were established based on a number of required assumptions, some of which may not be satisfied in practice. G_{typical} has shown that the maximal formulas within ER1 actually have different accuracy values in practice. G_{typical} shows that ER1 may be split into two groups of consistently similar formulas, ER1_a and ER1_b, where ER1_a actually bears no ERAO relation with ER1_b. G_{typical} has also shown that, in practice, the group ER5 should be split into ER5_a and ER5_b, where ER5_a is consistently more accurate than ER5_b. These empirical observations demonstrate that the current proving techniques presented in existing theoretical analysis are still not powerful enough to either prove or disprove the TRAO relations in practical scenarios when the assumptions do not hold. New mathematical tools need to be developed to prove or disprove the TRAO relations in practice.

On the other hand, although empirical studies suffer from certain threats to validity, where the ERAO relations obtained are often found inconsistent over different settings, such researches can help to provoke debates of the actual accuracy in practice over the theoretical work. Our work has broken the barrier between theoretical and empirical studies by developing a state-of-the-art empirical framework to validate or disprove the theoretical accuracy relations of formulas. Our framework is robust enough to include any new SBFL formulas and to establish new ERAO relations via accuracy graphs. For instance, G_{typical} has provided counter-examples for disproving the conjectures of theoretical accuracy relations, while G_{base} and G_{exp} help to validate or discover new TRAO relations for further theoretical analysis.



Accuracy Graphs	Region ID														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
G_{base} / G_{exp}	✓	✓	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓	✗	✗
G_{crash} / $G_{\text{exp\_crash}}$	✗	✗	✓	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓	✗
$G_{\text{nonsingle}}$ / $G_{\text{exp\_nonsingle}}$	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✓	✓	✗	✓
$G_{\text{typical}\,|\,\text{XCKX}}$ / G_{typical}	✓	✓	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗

Fig. 6. Venn diagram of the distribution of ERAO relations in accuracy graphs. Note: (1) Each oval shape represents two graphs: an XCKX-formula graph (e.g., G_{base}) and its corresponding expanded graph (e.g., G_{exp}). (2) Since G_{typical} is an expanded graph, its XCKX-formula subset (denoted as $G_{\text{typical}\,|\,\text{XCKX}}$) will be compared with another XCKX-formula graph. (3) The four oval shapes overlap to form 15 disjoint regions, indexed by the IDs 1–15, respectively, in the above diagram. Each region represents the set of the ERAO relations that appear in *all* of the graphs indicated by a tick (✓) but in *none* of the graphs indicated by a cross (✗) in the column of the region ID in the table above. For instance, Region 6 represents the subset $(G_2 \cap G_3) \setminus (G_4 \cup G_1)$ where G_2, G_3, G_4, and G_1 are G_{base}, G_{crash}, $G_{\text{nonsingle}}$, and $G_{\text{typical}\,|\,\text{XCKX}}$ for XCKX-formula graphs, and G_{exp}, $G_{\text{exp\_crash}}$, $G_{\text{exp\_nonsingle}}$, and G_{typical} for expanded graphs, respectively. (4) Below the ID of each region in the above Venn diagram is a pair of numbers shown in the form of (*x*/*y*), where *x* (respectively *y*) is the number of ERAO relations of the XCKX-formula graphs (respectively the expanded graphs) in that region. For instance, the numbers (4/5) under region ID 6 mean that in Region 6, there are 4 ERAO relations in G_{base} and G_{crash} but in neither $G_{\text{nonsingle}}$ nor $G_{\text{typical}\,|\,\text{XCKX}}$, and 5 ERAO relations in G_{exp} and $G_{\text{exp\_crash}}$ but in neither $G_{\text{exp\_nonsingle}}$ nor G_{typical}.

B. Quantitative Analysis

This section reports a quantitative analysis of the accuracy graphs comparing between *satisfying* versus *not satisfying* one or all of the assumptions. In doing so, we observe interesting results as elaborated in the two sections below.

Since G_{typical} is an expanded graph constructed from the full dataset without regard to the satisfaction of assumptions, we will use its XCKX-formula subset, defined as $G_{\text{typical}\,|\,\text{XCKX}} = \{e \in G_{\text{typical}} : e \text{ is an ERAO relation between two XCKX formulas}\}$, for comparing with other XCKX-formula graphs.

1) Graph Dissimilarities Associated With Assumption Violation: Fig. 6 shows a Venn diagram populated with data that

TABLE VI
EXTENTS OF OVERLAP AMONG ACCURACY GRAPHS

XCKX-Formula / Expanded Graph	Extent of overlap with graphs constructed from dataset		
	D_4 (violating the single-fault assumption only)	D_3 (with the presence of crash runs only)	D_1 (violating multiple assumptions)
G_{base}	88.9% ($G_{\text{nonsingle}}$)	69.7% (G_{crash})	64.4% ($G_{\text{typical}\mid\text{XCKX}}$)
G_{exp}	86.7% ($G_{\text{exp\_nonsingle}}$)	67.8% ($G_{\text{exp\_crash}}$)	64.2% (G_{typical})
Difference	2.2%	1.9%	0.2%

Note: The extent of overlap between graphs G_x and G_y, overlap(G_x, G_y), can be computed from the data shown in Fig. 6. For instance, $G_{\text{base}} \cap G_{\text{nonsingle}}$ spans the regions with IDs 2, 5, 9, and 13, which contain 53, 116, 56, and 8 elements, respectively, whose sum is 233. So, $G_{\text{base}} \cap G_{\text{nonsingle}}$ contains 233 elements. $G_{\text{base}} \setminus G_{\text{nonsingle}}$ contains 5 elements as it spans the regions with IDs 1, 3, 6, and 10, which contain 0, 1, 4, and 0 elements, respectively, whose sum is 5. $G_{\text{nonsingle}} \setminus G_{\text{base}}$ contains 24 elements as it spans the regions with IDs 4, 8, 12, and 15 which contain 13, 5, 0, and 6 element(s), respectively, whose sum is 24. Hence, $G_{\text{base}} \cup G_{\text{nonsingle}}$ has 262 (= 233 + 5 + 24) elements. Finally, overlap(G_{base}, $G_{\text{nonsingle}}$) = 233 ÷ 262 = 88.9%.

overview the distribution of ERAO relations among accuracy graphs. Mathematically, an accuracy graph is simply a set of accuracy relations. So the *extent of overlap* (*similarity*) between two graphs G_x and G_y, denoted as $\text{overlap}(G_x, G_y)$, may be quantified by the proportion of common elements in the sets, that is

$$\text{overlap}(G_x, G_y) = \frac{\text{Number of elements in } G_x \cap G_y}{\text{Number of elements in } G_x \cup G_y}.$$

Table VI shows the extents of overlap between G_{base} and other XCKX-formula graphs, as well as those between G_{exp} and other expanded graphs, respectively. We have the following observations.

First, all percentages of overlap are moderate to high, ranging between 64.2% and 88.9%.

Second, for XCKX-formula graphs, G_{base} is most similar to $G_{\text{nonsingle}}$ but less similar to G_{crash} and least similar to $G_{\text{typical}\mid\text{XCKX}}$ (the extents of overlap being 88.9%, 69.7%, and 64.4%, respectively). For expanded graphs, G_{exp} is also most similar to $G_{\text{exp\_nonsingle}}$ but less similar to $G_{\text{exp\_crash}}$ and least similar to G_{typical} (the extents of overlap being 86.7%, 67.8%, and 64.2%, respectively).

By comparing G_{base} with $G_{\text{nonsingle}}$ and G_{exp} with $G_{\text{exp\_nonsingle}}$, violating the single-fault assumption accounts for 11.1% (= 1 – 88.9%), and 13.3% (= 1 – 86.7%) in difference, respectively. However, the differences when using crash runs are larger, up to 32.2% (= 1 – 67.8%) for the expanded graph $G_{\text{exp\_crash}}$. When violating multiple assumptions, the differences are even larger, up to 35.8% (= 1 – 64.2%) for the expanded graph G_{typical}.

Third, the graph dissimilarities associated with violation of assumptions are generally not substantially affected when expanding from the XCKX-formula graphs to include *non*-XCKX formulas. As seen in the bottom row of Table VI, the effects of expanding from $G_{\text{nonsingle}}$ to $G_{\text{exp\_nonsingle}}$, from G_{crash} to $G_{\text{exp\_crash}}$, and from $G_{\text{typical}\mid\text{XCKX}}$ to G_{typical} are only 2.2% (out of 88.9%), 1.9% (out of 69.7%), and 0.2% (out of 64.4%), respectively, which are rather small.

Result 4: Violating the single-fault assumption alone accounts for 11.1−13.3% difference in ERAO relations, whereas the use of test suites having crash runs alone results in a difference of 30.3−32.2% in ERAO relations. Furthermore, violating multiple assumptions leads to even larger differences (35.6−35.8%). In comparison, the changes in expanding the graphs with more formulas are rather small (only 2.2%, 1.9%, and 0.2%, respectively) in all these scenarios.

2) Regions to Which the Majority of ERAO Relations Belongs: In Fig. 6, the numbers (276/581) at the bottom of the Venn diagram are the total numbers of ERAO relations identified from the XCKX-formula graphs and expanded graphs, respectively. We examine the specific regions to which the majority of ERAO relations belong and have the following observations.

Graphs G_{base} and G_{exp} contain 238 and 500 ERAO relations, respectively, which constitute 86.2% and 86.1% of all the ERAO relations found in this study. Outside these two graphs, the majority of the ERAO relations reside in $G_{\text{nonsingle}}$ and $G_{\text{exp\_nonsingle}}$. Specifically, if our analysis considers nonsingle faults, the combined sets $G_{\text{base}} \cup G_{\text{nonsingle}}$ and $G_{\text{exp}} \cup G_{\text{exp\_nonsingle}}$ already constitute 262 (94.9%) and 555 (95.5%) of all the ERAO relations, respectively.

Furthermore, if crash runs are also included, the combined sets $G_{\text{base}} \cup G_{\text{nonsingle}} \cup G_{\text{crash}}$ and $G_{\text{exp}} \cup G_{\text{exp\_nonsingle}} \cup G_{\text{exp\_crash}}$ constitute 273 (98.9%) and 570 (98.1%) of all the ERAO relations, respectively. Only the 3 and 11 ERAO relations in Region 7 of G_{typical} are not in the combined sets $G_{\text{base}} \cup G_{\text{nonsingle}} \cup G_{\text{crash}}$ and $G_{\text{exp}} \cup G_{\text{exp\_nonsingle}} \cup G_{\text{exp\_crash}}$, respectively.

Hence, more than 98% of all the ERAO relations in this study can be found by constructing accuracy graphs from the datasets D_2 (that satisfy all assumptions), D_3 (that include crash runs), and D_4 (that include nonsingle faults). Apparently, other violations of assumptions are associated with only a small proportion of the ERAO relations.

Result 5: Utilizing G_{base}, G_{crash}, and $G_{\text{nonsingle}}$ is sufficient to identify 98.9% of the ERAO relations of XCKX formulas, while utilizing G_{exp}, $G_{\text{exp\_crash}}$, and $G_{\text{exp\_nonsingle}}$ is sufficient to identify 98.1% of the ERAO relations of both XCKX and *non*-XCKX formulas. It shows that to identify ERAO relations in empirical studies, considering only nonsingle faults as well as faults that lead to crash runs might be sufficiently close to the typical or practical scenarios.

C. Benchmarking Prior SBFL Empirical Results

In this section, we further apply our accuracy graphs to benchmark the results reported in the publications from four top-tier software engineering venues [46]: FSE, ICSE, TOSEM,

TABLE VII
CROSS VALIDATION OF RESULTS OF RELATED WORK

	Summary of Validation	Venue [Ref.]
1	Zhang *et al.* reported the relations Jaccard (F_3 in ER2) → Tarantula (F_8 in ER3) ~ CBI Inc. (F_{10} in ER3), which are consistent with our results in G_{base} and G_{typical}.	FSE 2009 [68]
2	Abreu *et al.* reported the relations Ochiai (F_{34}) → Jaccard (F_3 in ER2) → Tarantula (F_8 in ER3), which are consistent with our results in G_{base} and G_{typical}.	JSS 2009 [1]
3	Naish *et al.* reported findings which are consistent with ours observed in G_{exp} except the following: ER2 → F_{25}, ER2 → F_{28}, ER2 → F_{29}, F_{26} → F_{25}, F_{26} → F_{28}, F_{26} → F_{29}, F_{31} ~ F_{33}, and F_{33} → F_{38}. Moreover, they used a similar experimental setup as what we did for G_{typical}, and obtained consistent results with ours in G_{typical}, except ER2 → F_{24}, ER2 → F_{25}, ER2 → F_{28}, ER2 → F_{29}, and F_{26} → F_{34}.	TOSEM 2011 [41]
4	Artzi *et al.* reported the relations Ochiai (F_{34}) ~ Jaccard (F_3 in ER2) → Tarantula (F_8 in ER3), but our study found Ochiai → Jaccard → Tarantula in both G_{base} and G_{typical}.	TSE 2012 [5]
5	Xie *et al.* constructed the theoretical accuracy graph G_{theory} which we have shown to be highly consistent with our empirical accuracy graphs except G_{typical}.	TOSEM 2013 [63]
6	Xuan *et al.* reported the relations Ochiai (F_{34}) → Ochiai2 (F_{35}) → Jaccard (F_3 in ER2) → Kulczynski2 (F_{31}) → Tarantula (F_8 in ER3) ~ CBI Inc. (F_{10} in ER3) whereas our study found Kulczynski2 → Ochiai → Jaccard → Tarantula ~ CBI Inc., Jaccard → Ochiai2 in G_{base}, and also Ochiai (F_{34}) → Ochiai2 (F_{35}), Kulczynski2 (F_{31}) → Tarantula (F_8 in ER3), but not other "→" relations.	FSE 2014 [65]
7	Wong *et al.* reported that D^2 (F_{41}) is more effective than Kulczynski1 (F_{30}) in both the best and worst cases. Regarding multi-fault scenarios, they found D^3 (F_{42})→ D^2 → Ochiai (F_{34}) → Tarantula (F_8 in ER3). Their findings are consistent with both G_{exp} and G_{typical}.	TRel 2014 [60]
8	Jin and Orso reported the relations Naish1 ($ER1_a$) ~ Naish2 ($ER1_b$) ~ Ochiai (F_{34}) whereas our study found Naish1 ~ Naish2 → Ochiai in G_{base}, and that Naish1, Naish2, and Ochiai are separate nodes and unconnected in G_{typical}.	TOSEM 2015 [26]

and TSE, together with notable publications from JSS and the present journal (TRel). We have selected eight representative publications for which the SBFL formulas and/or empirical data under study are closely relevant to this work. For brevity, we overload the same notations $\rightarrow$, $\leftarrow$, and $\sim$ to represent the relative effectiveness among formulas observed in their experiments. The results of validation with these publications are summarized in Table VII. In the table, we highlight their major results on SBFL formulas and indirectly connect them to the theoretical results in [63] through comparing with our accuracy graphs. Consistency between our results and the highlighted results helps to triangulate both results. Inconsistency between the results suggests areas that require further investigation through replication of experiments.

D. ERAO Relations With Different Test Suite Sizes

Empirical studies [1], [44] have identified that the accuracy of SBFL formulas can be affected to different extents by varying test suite sizes. Sometimes, the difference can be small.

TABLE VIII
REPRODUCTION OF ERAO RELATIONS WHEN THE TEST SUITE SIZE VARIES

Accuracy Graph	Number (Percentage) of ERAO Relations Reproduced When the Size of Test Suite Is a Percentage of the Original Test Suite Size						
	100%	75%		50%		25%	
G_{typical}	175	161	(92%)	157	(90%)	151	(86%)
G_{exp}	246	240	(98%)	231	(94%)	221	(90%)
$G_{\text{exp\_crash}}$	220	214	(97%)	208	(95%)	195	(89%)
$G_{\text{exp\_nonsingle}}$	249	229	(92%)	223	(90%)	210	(84%)

We would like to verify the potential impact on the observed ERAO relations if the test suite size is varied. We randomly selected 75%, 50%, and 25% of the test cases from the original test suites in this empirical study while preserving the assumptions made in constructing the original test suites except for the extents of statement/branch coverage. The experimental procedures are the same as those reported in Section III.

Table VIII summarizes the number and percentage of ERAO relations reproduced (compared with the original test suites) in the expanded accuracy graphs, G_{typical}, G_{exp}, $G_{\text{exp\_crash}}$, and $G_{\text{exp\_nonsingle}}$, by test suites constructed from 100%, 75%, 50%, and 25% of the test cases in the original test suites.

As shown in Table VIII, a total of 175 ERAO relations are revealed in G_{typical}. When the test suite size drops to 75%, 50%, and 25%, the percentages of ERAO relations in G_{typical} that can be obtained by the reduced test suites are 92%, 90%, and 86%, respectively. G_{exp} contains 246 ERAO relations. When the test suite size drops to 75%, 50%, and 25%, the percentages of ERAO relations in G_{exp} that can be obtained by the reduced test suites are 98%, 94%, and 90%, respectively. $G_{\text{exp\_crash}}$ and $G_{\text{exp\_nonsingle}}$ contain 220 and 249 ERAO relations, respectively. When the test suite size drops as above, the percentages of ERAO relations in $G_{\text{exp\_crash}}$ that are preserved by the reduced test suites are 97%, 95%, and 89%, respectively, whereas for $G_{\text{exp\_nonsingle}}$, the corresponding percentages are 92%, 90%, and 84%, respectively. Overall, the percentages of preserved ERAO relations are very high, ranging from 84% to 98%. The ERAO relations are not substantially affected even when the size of the test suite drops to only 25%. Thus, the ERAO relations appear to be robust with respect to variations in test suite sizes.

> *Result 6:* For all accuracy graphs, when the number of test cases is reduced to 75% from the original test suite, 92–98% of ERAO relations can still be reproduced. When the number of test cases drops to 50% and 25%, 90–95% and 84–90% of ERAO relations are preserved, respectively. Hence, the ERAO relations appear to be robust with respect to variations in test suite sizes.

E. Threats to Validity

The results of empirical studies invariably depend on the subjects used. They may or may not hold for other programs,

test data, execution environments or platforms, and so on. To facilitate cross-validation of results, our experiment studied 17 subject programs which have been widely used in existing software testing and SBFL research. Building accuracy graphs using other experimental setups can strengthen the generalizability of our result.

Our experimental subjects include different sized programs as well as spanning over two programming languages (C and Java). Each C subject is a complete program executed as a whole when running the test cases. Although the test suites of the Java subjects were JUnit test cases, which were by design unit tests, yet because the subjects are real and complete programs with no artificial stubs or drivers, execution of the JUnit test cases invariably covers many parts of the programs other than the units under test in such a way that the test executions were actually extended to integration-level coverage beyond the pure unit-level coverage. We are not aware of how test stubs or drivers could be constructed while maintaining the integrity of the subjects. If code development and debugging are conducted at the class level, the code coverage achieved by test cases obtained through the current experimental setup may likely be different. Considering the above factors, readers should interpret the results with caution. It is also unclear whether or how the variation of program sizes, different programming languages and different types and levels of tests contribute to any effects on the experimental results. Any such effects could be examined by further experimentations with these varying characteristics being separated for analysis.

In the data analysis of the experiment, we excluded the execution profiles obtained from executing the JUnit test classes whose corresponding test cases resulted in no failure. Inclusion of these execution profiles will change the values of the set of coverage variables of each exercised statement in the experiment. It is interesting to know the extent of changes compared to our current findings.

Program spectra can only be obtained on exercised program entities (statements in this experiment). For some faulty versions in which the faults reside in nonexecutable locations, we marked the related executable statements proximate to the actual fault positions. Other ways in deciding the faulty lines may produce different results.

Our approach to building the accuracy graph was to ensure statistical consistency at the subject program level. Using other notions of consistency may produce different accuracy graphs.

We adopted the Kruskal–Wallis test, which is a nonparametric test that does not require normality of the distribution of the data, to compare the accuracy values of formulas. Nonparametric tests are known to be less powerful than their parametric counterparts such as ANOVA. Using other statistical tests to compare the accuracy of formulas may result in different accuracy graphs.

The ERAO relations of formulas were obtained by simply aggregating the comparison results of letter groups in every subject and ranking scheme. Other ways of summarizing the accuracy values or aggregating the comparison results may produce different results. For example, if a formula F_x has more instances or higher frequencies of being more accurate than another formula F_y, then there is arguably stronger evidence that F_x is statistically more accurate than F_y when compared to the situation where F_x is more accurate than F_y in only one instance while in a large number of other instances they have similar accuracy values. Hence, aggregating the accuracy comparisons by using a scheme that takes into account the "strength" of evidence may result in different accuracy graphs.

The test verdicts were downloaded with the subject programs from their repositories. Similar to the findings of Zhang *et al*. [69], some of the verdicts may not correctly describe the faults. We had tried our best to ensure the correctness of the verdicts including appropriate corrections. Also, as a test oracle may not truly describe the absence of program faults due to coincidental correctness, there may exist incorrect test results. We aim to study single-fault programs in some accuracy graphs, and yet when executing the subject programs with their test suites, some not-yet-identified faults may be unknowingly or unintentionally triggered. If this is the case, the reproduction of the TRAO relations would become more interesting.

V. Related Work

In this section, we review some of the closely related work.

SBFL utilizes program spectra for distinguishing faulty program entities. Harrold *et al*. [21] found that execution trace spectrum was one of the most applicable spectra for debugging. Santelices *et al*. [51] found that different entities had different accuracy when localizing different kinds of faults. Tang *et al*. [54] found that performing SBFL at the object code level may be more accurate than at the statement level.

Tarantula [27] is one of the earliest techniques designed exclusively for the purpose of SBFL. Later, various classical metrics such as Jaccard [25], Ochiai [42], and Ample [1], [10] were imported to SBFL, and some of them were widely adopted in various empirical studies [1], [10], [68]. Wong *et al*. proposed many SBFL techniques, including Wong 1–3 [62], cross-tab (CBT) [61], and DStar [60]. Naish *et al*. [41] comprehensively evaluated the relative accuracy of 33 popular formulas and proposed two new formulas Naish1 and Naish2, which were also included in our experiment. Empirical studies usually suffer from their limited ability to generalize the results beyond the experimental context. Zhang *et al*. [69] proposed to study the effect of evaluation sequences. Xie *et al*. [63] performed a theoretical analysis on a set of formulas comparing their accuracy, which has been discussed in this paper.

SBFL in general utilizes the coverage information of both passed and failed runs. Some studies [1], [67] revealed that failed runs were more dominant to the accuracy of SBFL formulas than passed runs. The formulas Wong1 [62] and binary [35], which were proved [63] to be theoretical maximal formulas, solely utilize failed runs. Zhang *et al*. [67] proposed an improved technique by contrasting only the failed runs.

Empirical accuracy of SBFL formulas varies among different programs and faults. Studies have shown that using a combination of multiple formulas for fault localization can provide more stable accuracy and reduce error rates. Xuan and Monperrus [64] proposed a learning-based framework called MULTRIC that combined multiple formulas with learned weights from the spectra of faulty and non-faulty entities. Le *et al*. [31] proposed an information-retrieval-based technique that performs fault localization with multiple formulas together with analytical results from bug reports. Wang *et al*. [56] proposed using genetic algorithm (GA) and simulated annealing (SA) to compute the weights of multiple formulas.

Lucia *et al*. [36] proposed ways to construct fusion localizers that fuse the adaptively selected formulas for locating faults. They showed that certain variants of fusion localizers were more effective than individual formulas. Tang *et al*. [55] proposed a dual-service fault localization (DFL) model based on dynamic construction of an ensemble SBFL formula that contrasts the program spectra of a forthcoming version with the spectra of an existing live version of program service, and demonstrated that the DFL model could improve the accuracy of localizing faults in the forthcoming version of the service.

SBFL techniques can also be used to locate faults in domains other than conventional programs. Isolating faulty agents in a multiagent system (MAS) is known to be error-prone and time consuming and requires prior knowledge of the unexpected behavior of the agents. Passos *et al*. [44], [45] proposed a SBFL alternative, coined extended SBFL for MAS, that locates faulty agents by computing their fault suspiciousness in the MAS. Their technique reduced the diagnosis effort with only some minimal information from the system. Hofer *et al*. [23], [24] used SBFL techniques to identify buggy cells in spreadsheet applications. While program spectra were used to locate buggy source code, hit-spectra matrix was used to compute the failure probability of each spreadsheet cell. Liu *et al*. [34] adopted SBFL to aid the debugging of Simulink models, using an iterative approach to locate multiple faults by eliminating them one by one.

A recent survey [62] provides a comprehensive overview of SBFL techniques and discusses key issues and concerns that are pertinent to software fault localization as a whole.

VI. Conclusion

In this paper, we have revisited the accuracy graph G_{theory} of XCKX formulas. We have proposed a novel framework and methodology to construct accuracy graphs with the notion of monotonic ordering of accuracy among formulas over a series of subjects empirically. We have validated that our framework is robust through both experiments and systematic comparison with existing results.

Our methodology has demonstrated its potential to simulate *all* theoretical relations successfully in the context assumed by the prior theoretical study [63] on XCKX formulas, and uses the results of [63] as the basis to expand the graph to include a set of *non*-XCKX formulas. We have applied this methodology to produce other graphs with relaxed specific assumptions to reveal relations between *non*-XCKX formulas and those between XCKX and *non*-XCKX formulas found in practical scenarios, as summarized in Results 1–6 in Section IV. We are not aware of any prior research proposing empirical frameworks close to what we have developed in this work.

Unlike theoretical analysis that relies on the investigator's own insights and inspirations to discover and prove accuracy relations, our methodology has provided a highly applicable and practical mechanism for an investigator to systematically reveal empirical accuracy relations. Different from mathematical proofs that heavily rely on human intellectual ability, our methodology can be highly automated, rendering mega-scale graph construction a potential actionable item.

Previous theoretical results have only identified two maximal SBFL formulas (namely, ER1 and ER5 in Fig. 1). By extending to non-XCKX formulas, our experiments have shown that a set of other formulas may also be maximal, e.g., F_{39}, F_{41}, and F_{42} in G_{typical} in Fig. 5(a) under the "typical" setting which makes no assumption of the fault. That is, in our experiment, these formulas, together with ER1 and $\mathrm{ER5_a}$, were ranked ahead of other formulas while they were incomparable among themselves in the sense that none is statistically ranked ahead of another. Thus, whenever formulas in ER1 and ER5 are included in an experiment, we suggest that these other potentially maximal formulas should be included, too.

Moreover, the formulas of $\mathrm{ER5_a}$ were statistically ranked ahead of $\mathrm{ER5_b}$, which is contrary to the results of previous theoretical analysis that they are equally accurate. Thus, for practical use when the tester usually has no prior knowledge of any faults in the program or whether the program under test will crash or not, our results indicate that the tester may apply the maximal formulas in G_{typical} for performing SBFL in preference of other formulas.

Our methodology and ERAO relations serve as a starting point to align theoretical results with empirical data. In G_{base} and G_{exp}, their ERAO relations provide clues of potential TRAO relations for facilitating new mathematical proofs. As reported in Result 1b, G_{crash}, $G_{\mathrm{exp\_crash}}$, $G_{\mathrm{nonsingle}}$, and G_{exp} can align a majority of TRAO relations by considering off-by-one assumption and crash runs only. On the other hand, G_{typical} reveals counter-examples that reject the theoretical conjectures of some TRAO relations. Further work should be done to fine-tune the methodology, definition of ERAO relations as well as a more generalized TRAO relation toward the development of a theory that can better explain the empirical data without contradiction.

APPENDIX
SUBJECT PROGRAMS (IN C LANGUAGE) AND THEIR FAULTY PROGRAM VERSIONS STUDIED IN THIS PAPER

Subject Program ID	Version	Single / Nonsingle / Other Fault	Contain test cases that crash the program
P_1	1	Nonsingle	None
P_1	2	Single	None
P_1	3	Other	None
P_1	4	Nonsingle	None
P_1	5	Other	None
P_1	6	Nonsingle	None
P_1	7	Nonsingle	None
P_2	1	Other	None
P_2	2	Other	None
P_2	3	Other	None
P_2	4	Nonsingle	None
P_2	5	Single	None
P_2	6	Nonsingle	None
P_2	7	Nonsingle	None
P_2	8	Nonsingle	None
P_2	9	Nonsingle	None
P_2	10	Single	Some
P_3	1	Nonsingle	None
P_3	2	Other	None
P_3	3	Nonsingle	None
P_3	4	Nonsingle	None
P_3	5	Nonsingle	None
P_3	6	Nonsingle	None
P_3	7	Nonsingle	None
P_3	9	Nonsingle	None
P_3	10	Nonsingle	None
P_3	11	Nonsingle	None
P_3	12	Nonsingle	None
P_3	13	Nonsingle	None
P_3	15	Single	None
P_3	16	Nonsingle	None
P_3	17	Single	None
P_3	18	Nonsingle	None
P_3	19	Nonsingle	None
P_3	20	Single	None
P_3	21	Nonsingle	None
P_3	22	Nonsingle	None
P_3	23	Nonsingle	None
P_3	24	Other	None
P_3	25	Single	None
P_3	26	Nonsingle	None
P_3	27	Nonsingle	Some
P_3	28	Nonsingle	None
P_3	29	Nonsingle	None
P_3	30	Nonsingle	None
P_3	31	Nonsingle	None
P_4	1	Nonsingle	Some
P_4	2	Nonsingle	None
P_4	3	Nonsingle	None
P_4	4	Single	None
P_4	5	Other	Some
P_4	6	Nonsingle	Some
P_4	7	Nonsingle	None
P_4	8	Other	None
P_4	9	Single	Some
P_5	1	Other	None
P_5	2	Other	None
P_5	3	Other	None
P_5	4	Other	None
P_5	5	Nonsingle	None
P_5	6	Nonsingle	None
P_5	7	Nonsingle	None
P_5	8	Other	Some
P_5	10	Nonsingle	None
P_6	1	Nonsingle	None
P_6	2	Nonsingle	None
P_6	3	Nonsingle	None
P_6	4	Nonsingle	None
P_6	5	Nonsingle	None
P_6	6	Single	None
P_6	7	Nonsingle	None
P_6	8	Nonsingle	None
P_6	9	Single	None
P_6	10	Nonsingle	None
P_6	11	Nonsingle	None
P_6	12	Nonsingle	None
P_6	13	Nonsingle	None
P_6	14	Nonsingle	None
P_6	15	Nonsingle	None
P_6	16	Nonsingle	None
P_6	17	Nonsingle	None
P_6	18	Nonsingle	None
P_6	19	Nonsingle	None
P_6	20	Single	None
P_6	21	Single	None
P_6	22	Single	None
P_6	23	Single	None
P_6	24	Single	None
P_6	25	Nonsingle	None
P_6	26	Nonsingle	None
P_6	27	Nonsingle	None
P_6	28	Nonsingle	None
P_6	29	Nonsingle	None
P_6	30	Nonsingle	None
P_6	31	Nonsingle	None
P_6	32	Nonsingle	None
P_6	33	Nonsingle	None
P_6	34	Nonsingle	None
P_6	35	Nonsingle	None
P_6	36	Nonsingle	None
P_6	37	Single	None
P_6	39	Nonsingle	None
P_6	40	Nonsingle	None
P_6	41	Nonsingle	None
P_7	1	Other	None
P_7	2	Nonsingle	None
P_7	3	Nonsingle	None
P_7	4	Nonsingle	None
P_7	5	Single	None
P_7	6	Nonsingle	None
P_7	7	Single	None
P_7	8	Nonsingle	None
P_7	9	Single	None
P_7	10	Nonsingle	None
P_7	11	Single	None
P_7	12	Nonsingle	None
P_7	13	Single	None
P_7	14	Nonsingle	None
P_7	15	Single	None
P_7	16	Single	None
P_7	17	Single	None
P_7	18	Nonsingle	None
P_7	19	Nonsingle	None
P_7	20	Nonsingle	None
P_7	21	Nonsingle	None
P_7	22	Single	None
P_7	23	Single	None
P_8	3	Single	None
P_8	4	Single	None
P_8	5	Single	None
P_8	6	Single	None
P_8	7	Nonsingle	None
P_8	8	Nonsingle	None
P_8	9	Nonsingle	None
P_8	10	Nonsingle	None
P_8	11	Nonsingle	None
P_8	12	Nonsingle	None
P_8	13	Nonsingle	None
P_8	14	Single	None
P_8	15	Single	None
P_8	16	Nonsingle	None
P_8	17	Other	None
P_8	18	Single	None
P_8	19	Nonsingle	None
P_8	20	Single	None
P_8	21	Single	None
P_8	22	Nonsingle	None
P_8	23	Single	None
P_8	24	Nonsingle	None
P_8	25	Nonsingle	None
P_8	26	Single	None
P_8	27	Nonsingle	None
P_8	28	Single	None
P_8	29	Nonsingle	None
P_8	30	Single	None
P_8	31	Single	None
P_8	33	Single	None
P_8	35	Other	None
P_8	36	Nonsingle	None
P_8	37	Nonsingle	None
P_8	38	Other	None
P_9	1.1	Single	None
P_9	1.10	Nonsingle	None
P_9	1.11	Nonsingle	None
P_9	1.14	Single	None
P_9	1.15	Single	None
P_9	1.17	Single	None
P_9	1.18	Nonsingle	None
P_9	1.19	Single	None
P_9	1.3	Single	None
P_9	1.4	Single	None
P_9	1.5	Nonsingle	None
P_9	1.6	Single	All
P_9	1.7	Single	None
P_9	1.9	Single	None
P_9	2.11	Other	None
P_9	2.12	Nonsingle	None
P_9	2.14	Single	None
P_9	2.15	Nonsingle	None
P_9	2.16	Single	None
P_9	2.18	Other	None
P_9	2.19	Single	None
P_9	2.2	Single	None
P_9	2.3	Nonsingle	None
P_9	2.5	Other	None
P_9	2.6	Single	None
P_9	2.7	Single	None
P_9	2.9	Other	None
P_9	3.10	Nonsingle	None
P_9	3.11	Other	None
P_9	3.14	Single	None
P_9	3.15	Single	None
P_9	3.17	Single	None
P_9	3.6	Single	None
P_9	3.9	Nonsingle	None
P_9	4.12	Other	None
P_9	4.13	Nonsingle	None
P_9	4.14	Nonsingle	None
P_9	4.15	Single	None
P_9	4.16	Single	None
P_9	4.5	Single	None
P_9	4.6	Nonsingle	None
P_9	4.7	Nonsingle	None
P_9	4.8	Nonsingle	None
P_9	5.6	Single	All
P_9	5.9	Single	None
P_{10}	1.11	Single	None
P_{10}	1.14	Single	Some
P_{10}	1.3	Nonsingle	Some
P_{10}	1.7	Single	None
P_{10}	1.8	Nonsingle	None
P_{10}	2.1	Nonsingle	None
P_{10}	2.2	Other	None
P_{10}	2.6	Single	None
P_{10}	2.7	Single	None
P_{10}	3.12	Other	None
P_{10}	3.13	Other	None
P_{10}	3.16	Single	None
P_{10}	3.18	Single	None
P_{10}	3.2	Single	None
P_{10}	3.3	Nonsingle	None
P_{10}	3.8	Single	None
P_{10}	4.10	Single	None
P_{10}	4.12	Nonsingle	None
P_{10}	4.2	Nonsingle	None
P_{11}	1.14	Single	None
P_{11}	1.15	Nonsingle	None
P_{11}	1.16	Nonsingle	None
P_{11}	1.2	Single	Some
P_{11}	1.4	Nonsingle	None
P_{11}	1.5	Nonsingle	None
P_{11}	2.1	Nonsingle	None
P_{11}	2.3	Nonsingle	None
P_{11}	2.6	Single	None
P_{11}	4.1	Nonsingle	None
P_{11}	4.10	Single	None
P_{11}	4.6	Nonsingle	None
P_{11}	5.1	Nonsingle	None
P_{11}	5.13	Nonsingle	None
P_{11}	5.6	Nonsingle	Some
P_{11}	5.7	Single	None
P_{11}	5.9	Single	None
P_{12}	2.1	Single	None
P_{12}	2.2	Other	None
P_{12}	2.3	Other	None
P_{12}	2.4	Other	None
P_{12}	2.5	Nonsingle	None
P_{12}	3.1	Single	None
P_{12}	3.2	Single	None
P_{12}	3.3	Nonsingle	None
P_{12}	3.4	Single	None
P_{12}	3.5	Nonsingle	None
P_{12}	3.6	Single	None
P_{12}	5.1	Single	None
P_{12}	5.2	Nonsingle	None
P_{12}	5.3	Single	None
P_{12}	5.4	Single	None
P_{12}	6.1	Nonsingle	None
P_{12}	6.3	Nonsingle	None
P_{12}	6.4	Single	None
P_{12}	6.5	Single	None
P_{12}	6.6	Nonsingle	None
P_{12}	7.1	Single	None
P_{12}	7.2	Nonsingle	Some
P_{12}	7.3	Nonsingle	None
P_{12}	7.4	Single	None

Note: The table lists each faulty version of the C subject programs analyzed in this paper, the type of the fault in the faulty version, and whether the associated test suites contain test cases that lead to crash runs.

(CONTINUED)
SUBJECT PROGRAMS (IN JAVA LANGUAGE) AND THEIR FAULTY PROGRAM VERSIONS STUDIED IN THIS PAPER

Subject Program ID	Version	Single / Nonsingle / Other Fault	Contain test cases that crash the program
P_{13}	1	Single	None
P_{13}	2	Nonsingle	None
P_{13}	3	Other	None
P_{13}	4	Other	None
P_{13}	5	Nonsingle	None
P_{13}	6	Nonsingle	None
P_{13}	7	Nonsingle	None
P_{13}	8	Single	None
P_{13}	9	Nonsingle	None
P_{13}	10	Nonsingle	None
P_{13}	11	Single	None
P_{13}	12	Nonsingle	None
P_{13}	13	Single	None
P_{13}	15	Nonsingle	None
P_{13}	16	Nonsingle	None
P_{13}	17	Other	None
P_{13}	19	Nonsingle	None
P_{13}	20	Single	None
P_{13}	21	Nonsingle	None
P_{13}	22	Nonsingle	None
P_{13}	24	Single	None
P_{13}	25	Nonsingle	None
P_{13}	26	Other	None
P_{14}	1	Nonsingle	None
P_{14}	2	Nonsingle	None
P_{14}	3	Nonsingle	None
P_{14}	4	Nonsingle	None
P_{14}	5	Nonsingle	None
P_{14}	6	Nonsingle	None
P_{14}	7	Nonsingle	None
P_{14}	8	Nonsingle	None
P_{14}	9	Nonsingle	None
P_{14}	10	Single	None
P_{14}	11	Nonsingle	None
P_{14}	12	Nonsingle	None
P_{14}	13	Single	None
P_{14}	14	Single	None
P_{14}	15	Nonsingle	None
P_{14}	16	Nonsingle	None
P_{14}	17	Nonsingle	None
P_{14}	18	Nonsingle	None
P_{14}	19	Nonsingle	None
P_{14}	20	Nonsingle	None
P_{14}	21	Nonsingle	None
P_{14}	22	Nonsingle	None
P_{14}	23	Nonsingle	None
P_{14}	24	Nonsingle	None
P_{14}	25	Nonsingle	None
P_{14}	26	Nonsingle	None
P_{14}	27	Nonsingle	None
P_{14}	29	Nonsingle	None
P_{14}	31	Nonsingle	None
P_{14}	32	Nonsingle	None
P_{14}	33	Nonsingle	None
P_{14}	36	Nonsingle	None
P_{14}	38	Nonsingle	None
P_{14}	39	Nonsingle	None
P_{14}	40	Nonsingle	None
P_{14}	41	Nonsingle	None
P_{14}	42	Nonsingle	None
P_{14}	43	Nonsingle	None
P_{14}	44	Nonsingle	None
P_{14}	45	Nonsingle	None
P_{14}	48	Nonsingle	None
P_{14}	49	Nonsingle	None
P_{14}	50	Nonsingle	None
P_{14}	51	Single	None
P_{14}	52	Nonsingle	None
P_{14}	53	Nonsingle	None
P_{14}	55	Nonsingle	None
P_{14}	56	Nonsingle	None
P_{14}	57	Nonsingle	None
P_{14}	58	Nonsingle	None
P_{14}	59	Nonsingle	None
P_{14}	60	Nonsingle	None
P_{14}	61	Nonsingle	None
P_{14}	62	Nonsingle	None
P_{14}	63	Nonsingle	None
P_{14}	64	Nonsingle	None
P_{14}	65	Single	None
P_{14}	66	Nonsingle	None
P_{14}	67	Nonsingle	None
P_{14}	68	Nonsingle	None
P_{14}	69	Nonsingle	None
P_{14}	70	Single	None
P_{14}	71	Nonsingle	None
P_{14}	73	Nonsingle	None
P_{14}	74	Nonsingle	None
P_{14}	75	Nonsingle	None
P_{14}	76	Nonsingle	None
P_{14}	77	Other	None
P_{14}	78	Nonsingle	None
P_{14}	80	Nonsingle	None
P_{14}	81	Nonsingle	None
P_{14}	82	Nonsingle	None
P_{14}	83	Nonsingle	None
P_{14}	84	Nonsingle	None
P_{14}	85	Nonsingle	None
P_{14}	86	Single	None
P_{14}	87	Nonsingle	None
P_{14}	88	Nonsingle	None
P_{14}	91	Nonsingle	None
P_{14}	92	Single	None
P_{14}	93	Single	None
P_{14}	94	Nonsingle	None
P_{14}	95	Nonsingle	None
P_{14}	96	Nonsingle	None
P_{14}	97	Single	None
P_{14}	98	Nonsingle	None
P_{14}	99	Nonsingle	None
P_{14}	100	Nonsingle	None
P_{14}	101	Nonsingle	None
P_{14}	102	Nonsingle	None
P_{14}	104	Single	None
P_{14}	105	Nonsingle	None
P_{14}	107	Other	None
P_{14}	108	Nonsingle	None
P_{14}	109	Nonsingle	None
P_{14}	111	Nonsingle	None
P_{14}	112	Nonsingle	None
P_{14}	113	Nonsingle	None
P_{14}	114	Nonsingle	None
P_{14}	115	Nonsingle	None
P_{14}	116	Nonsingle	None
P_{14}	117	Nonsingle	None
P_{14}	118	Nonsingle	None
P_{14}	119	Single	None
P_{14}	120	Nonsingle	None
P_{14}	121	Nonsingle	None
P_{14}	122	Nonsingle	None
P_{14}	123	Single	None
P_{14}	124	Nonsingle	None
P_{14}	125	Nonsingle	None
P_{14}	126	Nonsingle	None
P_{14}	127	Nonsingle	None
P_{14}	128	Nonsingle	None
P_{14}	129	Nonsingle	None
P_{14}	130	Nonsingle	None
P_{14}	131	Nonsingle	None
P_{14}	132	Nonsingle	None
P_{14}	133	Other	None
P_{15}	2	Single	None
P_{15}	3	Other	None
P_{15}	5	Single	None
P_{15}	7	Nonsingle	None
P_{15}	8	Nonsingle	None
P_{15}	9	Single	None
P_{15}	10	Other	None
P_{15}	11	Single	None
P_{15}	13	Nonsingle	None
P_{15}	15	Nonsingle	None
P_{15}	16	Nonsingle	None
P_{15}	17	Nonsingle	None
P_{15}	18	Nonsingle	None
P_{15}	19	Other	None
P_{15}	20	Nonsingle	None
P_{15}	21	Nonsingle	None
P_{15}	23	Nonsingle	None
P_{15}	24	Nonsingle	None
P_{15}	25	Other	None
P_{15}	26	Nonsingle	None
P_{15}	27	Single	None
P_{15}	28	Nonsingle	None
P_{15}	29	Nonsingle	None
P_{15}	30	Single	None
P_{15}	31	Nonsingle	None
P_{15}	32	Single	None
P_{15}	33	Single	None
P_{15}	34	Single	None
P_{15}	35	Nonsingle	None
P_{15}	36	Nonsingle	None
P_{15}	37	Nonsingle	None
P_{15}	38	Nonsingle	None
P_{15}	39	Nonsingle	None
P_{15}	40	Nonsingle	None
P_{15}	41	Nonsingle	None
P_{15}	42	Nonsingle	None
P_{15}	44	Nonsingle	None
P_{15}	45	Nonsingle	None
P_{15}	46	Nonsingle	None
P_{15}	47	Nonsingle	None
P_{15}	48	Single	None
P_{15}	49	Nonsingle	None
P_{15}	50	Nonsingle	None
P_{15}	51	Nonsingle	None
P_{15}	52	Nonsingle	None
P_{15}	53	Nonsingle	None
i_{15}	54	Nonsingle	None
P_{15}	55	Nonsingle	None
P_{15}	56	Nonsingle	None
P_{15}	57	Single	None
P_{15}	58	Single	None
P_{15}	59	Nonsingle	None
P_{15}	60	Nonsingle	None
P_{15}	61	Single	None
P_{15}	62	Nonsingle	None
P_{15}	63	Nonsingle	None
P_{15}	64	Nonsingle	None
P_{15}	65	Nonsingle	None
P_{15}	66	Nonsingle	None
P_{15}	67	Nonsingle	None
P_{15}	68	Nonsingle	None
P_{15}	69	Single	None
P_{15}	70	Single	None
P_{15}	72	Nonsingle	None
P_{15}	73	Nonsingle	None
P_{15}	74	Nonsingle	None
P_{15}	75	Nonsingle	None
P_{15}	76	Nonsingle	None
P_{15}	78	Nonsingle	None
P_{15}	79	Nonsingle	None
P_{15}	80	Single	None
P_{15}	81	Nonsingle	None
P_{15}	82	Nonsingle	None
P_{15}	83	Nonsingle	None
P_{15}	84	Nonsingle	None
P_{15}	85	Single	None
P_{15}	86	Nonsingle	None
P_{15}	87	Nonsingle	None
P_{15}	88	Nonsingle	None
P_{15}	89	Nonsingle	None
P_{15}	90	Nonsingle	None
P_{15}	91	Nonsingle	None
P_{15}	92	Nonsingle	None
P_{15}	93	Nonsingle	None
P_{15}	94	Single	None
P_{15}	95	Other	None
P_{15}	96	Nonsingle	None
P_{15}	97	Nonsingle	None
P_{15}	99	Nonsingle	None
P_{15}	100	Nonsingle	None
P_{15}	101	Nonsingle	None
P_{15}	102	Nonsingle	None
P_{15}	103	Nonsingle	None
P_{15}	104	Nonsingle	None
P_{15}	105	Nonsingle	None
P_{15}	106	Nonsingle	None
P_{16}	3	Nonsingle	None
P_{16}	4	Single	None
P_{16}	5	Nonsingle	None
P_{16}	6	Nonsingle	None
P_{16}	7	Nonsingle	None
P_{16}	8	Nonsingle	None
P_{16}	9	Nonsingle	None
P_{16}	10	Nonsingle	None
P_{16}	11	Nonsingle	None
P_{16}	13	Nonsingle	None
P_{16}	14	Other	None
P_{16}	15	Other	None
P_{16}	16	Single	None
P_{16}	17	Nonsingle	None
P_{16}	18	Nonsingle	None
P_{16}	19	Single	None
P_{16}	20	Nonsingle	None
P_{16}	21	Nonsingle	None
P_{16}	22	Single	None
P_{16}	23	Nonsingle	None
P_{16}	23	Nonsingle	None
P_{16}	24	Nonsingle	None
P_{16}	25	Nonsingle	None
P_{16}	26	Nonsingle	None
P_{16}	27	Nonsingle	None
P_{17}	1	Other	None
P_{17}	2	Nonsingle	None
P_{17}	3	Nonsingle	None
P_{17}	4	Nonsingle	None
P_{17}	5	Nonsingle	None
P_{17}	6	Single	None
P_{17}	7	Nonsingle	None
P_{17}	8	Single	None
P_{17}	9	Other	None
P_{17}	10	Nonsingle	None
P_{17}	11	Other	None
P_{17}	12	Other	None
P_{17}	13	Other	None
P_{17}	14	Nonsingle	None
P_{17}	15	Nonsingle	None
P_{17}	16	Nonsingle	None
P_{17}	17	Nonsingle	None
P_{17}	18	Nonsingle	None
P_{17}	19	Nonsingle	None
P_{17}	20	Nonsingle	None
P_{17}	21	Nonsingle	None
P_{17}	22	Nonsingle	None
P_{17}	24	Nonsingle	None
P_{17}	25	Nonsingle	None
P_{17}	26	Single	None
P_{17}	27	Nonsingle	None
P_{17}	28	Other	None
P_{17}	29	Nonsingle	None
P_{17}	30	Nonsingle	None
P_{17}	31	Nonsingle	None
P_{17}	32	Nonsingle	None
P_{17}	33	Nonsingle	None
P_{17}	34	Nonsingle	None
P_{17}	35	Nonsingle	None
P_{17}	36	Nonsingle	None
P_{17}	37	Nonsingle	None
P_{17}	38	Other	None
P_{17}	39	Other	None
P_{17}	40	Nonsingle	None
P_{17}	41	Nonsingle	None
P_{17}	42	Other	None
P_{17}	43	Other	None
P_{17}	44	Other	None
P_{17}	45	Other	None
P_{17}	46	Nonsingle	None
P_{17}	47	Nonsingle	None
P_{17}	48	Nonsingle	None
P_{17}	49	Nonsingle	None
P_{17}	50	Nonsingle	None
P_{17}	51	Other	None
P_{17}	52	Other	None
P_{17}	53	Nonsingle	None
P_{17}	54	Other	None
P_{17}	55	Other	None
P_{17}	57	Single	None
P_{17}	58	Nonsingle	None
P_{17}	59	Single	None
P_{17}	60	Nonsingle	None
P_{17}	61	Single	None
P_{17}	62	Nonsingle	None
P_{17}	63	Nonsingle	None
P_{17}	65	Nonsingle	None

Note: The table lists each faulty version of the Java subject programs analyzed in this paper, the type of the fault in the faulty version, and whether the associated test suites contain test cases that lead to crash runs.

(CONTINUED)

XCKX FORMULAS

ID	Name	Group	Formula	Ref.
F_1	Naish1	ER1$_a$	$\begin{cases} -1, \text{ if } a_{ef} < F \\ P - a_{ep}, \text{ if } a_{ef} = F \end{cases}$	[41]
F_2	Naish2	ER1$_b$	$a_{ef} - \frac{a_{ep}}{a_{ep} + a_{np} + 1}$	[41]
F_3	Jaccard	ER2	$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep}}$	[25]
F_4	Anderberg	ER2	$\frac{a_{ef}}{a_{ef} + 2(a_{nf} + a_{ep})}$	[4]
F_5	Sørensen dice	ER2	$\frac{2a_{ef}}{2a_{ef} + a_{nf} + a_{ep}}$	[11]
F_6	Dice	ER2	$\frac{2a_{ef}}{a_{ef} + a_{nf} + a_{ep}}$	[11]
F_7	Goodman	ER2	$\frac{2a_{ef} - a_{nf} - a_{ep}}{2a_{ef} + a_{nf} + a_{ep}}$	[19]
F_8	Tarantula	ER3	$\frac{\frac{a_{ef}}{a_{ef} + a_{nf}}}{\frac{a_{ef}}{a_{ef} + a_{nf}} + \frac{a_{ep}}{a_{ep} + a_{np}}}$	[27]
F_9	q_e	ER3	$\frac{a_{ef}}{a_{ef} + a_{ep}}$	[32]
F_{10}	CBI Inc.	ER3	$\frac{a_{ef}}{a_{ef} + a_{ep}} - \frac{a_{ef} + a_{nf}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$	[33]
F_{11}	Wong2	ER4	$a_{ef} - a_{ep}$	[59]
F_{12}	Hamann	ER4	$\frac{a_{ef} + a_{np} - a_{nf} - a_{ep}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$	[35]
F_{13}	Simple matching	ER4	$\frac{a_{ef} + a_{np}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$	[39]
F_{14}	Sokal	ER4	$\frac{2(a_{ef} + a_{np})}{2(a_{ef} + a_{np}) + a_{nf} + a_{ep}}$	[35]
F_{15}	Rogers & Tanimoto	ER4	$\frac{a_{ef} + a_{np}}{a_{ef} + a_{np} + 2(a_{nf} + a_{ep})}$	[48]
F_{16}	Hamming etc.	ER4	$a_{ef} + a_{np}$	[20]
F_{17}	Euclid	ER4	$\sqrt{a_{ef} + a_{np}}$	[30]
F_{18}	Wong1	ER5$_a$	a_{ef}	[59]
F_{19}	Russell & Rao	ER5$_a$	$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$	[50]
F_{20}	Binary	ER5$_b$	$\begin{cases} 0, \text{ if } a_{ef} < F \\ 1, \text{ if } a_{ef} = F \end{cases}$	[35]
F_{21}	Scott	ER6	$\frac{4a_{ef}a_{np} - 4a_{nf}a_{ep} - (a_{nf} - a_{ep})^2}{(2a_{ef} + a_{nf} + a_{ep})(2a_{np} + a_{nf} + a_{ep})}$	[52]
F_{22}	Rogot1	ER6	$\frac{1}{2}\left(\frac{a_{ef}}{2a_{ef} + a_{nf} + a_{ep}} + \frac{a_{np}}{2a_{np} + a_{nf} + a_{ep}}\right)$	[49]
F_{24}	Ample2		$\frac{a_{ef}}{a_{ef} + a_{nf}} - \frac{a_{ep}}{a_{ep} + a_{np}}$	[10]
F_{25}	Arithmetic mean		$\frac{2a_{ef}a_{np} - 2a_{nf}a_{ep}}{(a_{ef} + a_{ep})(a_{np} + a_{nf}) + (a_{ef} + a_{nf})(a_{ep} + a_{np})}$	[49]
F_{26}	Cohen		$\frac{2a_{ef}a_{np} - 2a_{nf}a_{ep}}{(a_{ef} + a_{ep})(a_{np} + a_{ep}) + (a_{ef} + a_{nf})(a_{nf} + a_{np})}$	[9]
F_{27}	Fleiss		$\frac{4a_{ef}a_{np} - 4a_{nf}a_{ep} - (a_{nf} - a_{ep})^2}{(2a_{ef} + a_{nf} + a_{ep}) + (2a_{np} + a_{nf} + a_e)}$	[17]
F_{31}	Kulczynski2		$\frac{1}{2}\left(\frac{a_{ef}}{a_{ef} + a_{nf}} + \frac{a_{ef}}{a_{ef} + a_{ep}}\right)$	[35]
F_{33}	M2		$\frac{a_{ef}}{a_{ef} + a_{np} + 2(a_{nf} + a_{ep})}$	[35]
F_{34}	Ochiai		$\frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}$	[42]
F_{38}	Wong3		$a_{ef} - h$, where $h = \begin{cases} a_{ep}, \text{ if } a_{ep} \le 2 \\ 2 + 0.1(a_{ep} - 2), \text{ if } 2 < a_{ep} \le 10 \\ 2.8 + 0.001(a_{ep} - 10), \text{ if } a_{ep} > 10 \end{cases}$	[59]

(CONTINUED)

NON-XCKX FORMULAS

ID	Name	Formula	Ref.
F_{23}	Ample	$\left\lvert \frac{a_{ef}}{a_{ef} + a_{nf}} - \frac{a_{ep}}{a_{ep} + a_{np}} \right\rvert$	[10]
F_{28}	Geometric mean	$\frac{a_{ef}a_{np} - a_{nf}a_{ep}}{\sqrt{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}}$	[38]
F_{29}	Harmonic Mean	$\frac{(a_{ef}a_{np} - a_{nf}a_{ep})\left((a_{ef} + a_{ep})(a_{np} + a_{nf}) + (a_{ef} + a_{nf})(a_{ep} + a_{np})\right)}{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}$	[49]
F_{30}	Kulczynski1	$\frac{a_{ef}}{a_{nf} + a_{ep}}$	[35]
F_{32}	M1	$\frac{a_{ef} + a_{np}}{a_{nf} + a_{ep}}$	[16]
F_{35}	Ochiai2	$\frac{a_{ef}a_{np}}{\sqrt{(a_{ef} + a_{ep})(a_{np} + a_{nf})(a_{ef} + a_{nf})(a_{ep} + a_{np})}}$	[42]
F_{36}	Overlap	$\frac{a_{ef}}{\min(a_{ef}, a_{nf}, a_{ep})}$	[14]
F_{37}	Rogot2	$\frac{1}{4}\left(\frac{a_e}{a_{ef} + a_{ep}} + \frac{a_{ef}}{a_{ef} + a_{nf}} + \frac{a_{np}}{a_{np} + a_{ep}} + \frac{a_{np}}{a_{np} + a_{nf}}\right)$	[49]
F_{39}	Zoltar	$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + \frac{10000a_{nf}a_{ep}}{a_e}}$	[18]
F_{40}	CBT	*Not listed here due to numerous computation steps. Refer to* [61] *for details.*	[61]
F_{41}	D^2	$\frac{{a_{ef}}^2}{a_{nf} + a_{ep}}$	[60]
F_{42}	D^3	$\frac{{a_{ef}}^3}{a_{nf} + a_{ep}}$	[60]

REFERENCES

[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1780–1792, 2009.

[2] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM J. Comput.*, vol. 1, no. 2, pp. 131–137, 1972.

[3] G. Altieri, "MATLAB community function: Means with letters after a multiple comparison test," [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/49696. Accessed on: Apr. 22, 2016.

[4] M. R. Anderberg, *Cluster Analysis for Applications*. New York, NY, USA: Academic, 1973.

[5] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, "Fault localization for dynamic web applications," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 314–335, Mar./Apr. 2012.

[6] T. Ball and J. R. Larus, "Efficient path profiling," in *Proc. Annu. ACM/IEEE Int. Symp. Microarchitecture*, 1996, pp. 46–57.

[7] T. Y. Chen, X. Xie, F. C. Kuo, and B. Xu, "A revisit of a theoretical analysis on spectrum-based fault localization," in *Proc. Annu. Comput. Softw. Appl. Conf.*, 2015, vol. 1, pp. 17–22.

[8] T. M. Chilimbi, B. Liblit, K. Mehra, A. V. Nori, and K. Vaswani, "HOLMES: Effective statistical debugging via efficient path profiling," in *Proc. Int. Conf. Softw. Eng.*, 2009, pp. 34–44.

[9] J. Cohen, "A coefficient of agreement for nominal scales," *Educ. Psychol. Meas.*, vol. 20, no. 1, pp. 37–46, 1960.

[10] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight defect localization for Java," in *Proc. Eur. Conf. Object-Oriented Program.*, 2005, pp. 528–550.

[11] L. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[12] N. DiGiuseppe and J. A. Jones, "On the influence of multiple faults on coverage-based fault localization," in *Proc. Int. Symp. Softw. Testing Anal.*, 2011, pp. 210–220.

[13] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng.*, vol. 10, no. 4, pp. 405–435, 2005.

[14] M. Dunham, *Data Mining: Introductory and Advanced Topics*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2002.

[15] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz—Open source graph drawing tools," in *Graph Drawing*. Berlin, Germany: Springer, 2001, pp. 483–484.

[16] B. Everitt, *Graphical Techniques for Multivariate Data*. Amsterdam, The Netherlands: North Holland, 1978.

[17] J. Fleiss, "Estimating the accuracy of dichotomous judgments," *Psychometrika*, vol. 30, no. 4, pp. 469–479, 1965.

[18] A. Gonzalez, "Automatic error detection techniques based on dynamic invariants," M.S. thesis, Delft Univ. Technol., Delft, The Netherlands, 2007.

[19] L. Goodman and W. Kruskal, "Measures of association for cross-classifications (Part I)," *J. Amer. Statist. Assoc.*, vol. 49, no. 268, pp. 732–764, 1954.

[20] R. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, no. 2, pp. 147–160, 1950.

[21] M. J. Harrold, G. Rothermel, K. Sayre, R. Wu, and L. Yi, "An empirical investigation of the relationship between spectra differences and regression faults," *Softw. Testing, Verification Rel.*, vol. 10, no. 3, pp. 171–194, 2000.

[22] Y. Hochberg and A. C. Tamhane, *Multiple Comparison Procedures*. Hoboken, NJ, USA: Wiley, 1987.

[23] B. Hofer, A. Perez, R. Abreu, and F. Wotawa, "On the empirical evaluation of similarity coefficients for spreadsheets fault localization," *Autom. Softw. Eng.*, vol. 22, no. 1, pp. 47–74, 2015.

[24] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, and E. Getzner, "On the empirical evaluation of fault localization techniques for spreadsheets," in *Proc. Int. Conf. Fundam. Approaches Softw. Eng.*, 2013, pp. 68–82.

[25] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et du Jura," *Bulletin de la Société Vaudoise des Sciences Naturelles*, Impr. Corbaz, vol. 37, no. 142, pp. 547–579, 1901.

[26] W. Jin and A. Orso, "Automated support for reproducing and debugging field failures," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 4, 2015, Art. no. 26.

[27] J. A. Jones and M. J. Harrold, "Empirical evaluation of the Tarantula automatic fault-localization technique," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2005, pp. 273–282.

[28] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proc. Int. Symp. Softw. Testing Anal.*, 2014, pp. 437–440.

[29] B. Korel and J. Laski, "Dynamic slicing of computer programs," *J. Syst. Softw.*, vol. 13, no. 3, pp. 187–195, 1990.

[30] E. Krause, "Taxicab geometry," *Math. Teacher*, vol. 66, no. 8, pp. 695–706, 1973.

[31] T.-D. B. Le, R. J. Oentaryo, and D. Lo, "Information retrieval and spectrum based bug localization: Better together," in *Proc . Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2015, pp. 579–590.

[32] H. Lee, L. Naish, and K. Ramamohanarao, "Study of the relationship of bug consistency with respect to performance of spectra metrics," in *Proc. Int. Conf. Comput. Sci. Inf. Technol.*, 2009, pp. 501–508.

[33] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implementation*, 2005, pp. 15–26.

[34] B. Liu, S. Nejati, L. C. Briand, and T. Bruckmann, "Simulink fault localization: An iterative statistical debugging approach," *Softw. Test., Verif. Rel.*, vol. 26, no. 6, pp. 431–459, 2016.

[35] F. Lourenco, V. Lobo, and F. Bação, "Binary-based similarity measures for categorical data and their application in self-organizing maps," in *Proc. Conf. Classification Anal. Data*, 2004, pp. 121–138.

[36] Lucia, D. Lo, and X. Xia, "Fusion fault localizers," in *Proc. ACM/IEEE Int Conf. Autom. Softw. Eng.*, 2014, pp. 127–138.

[37] C. K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2005, pp. 190–200.

[38] A. Maxwell and A. Pilliner, "Deriving coefficients of reliability and agreement for ratings," *Brit. J. Math. Statist. Psychol.*, vol. 21, no. 1, pp. 105–16, 1968.

[39] A. Meyer, A. Garcia, A. Souza, and C. Souza, Jr., "Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (Zea mays L)," *Genetics Mol. Biol.*, vol. 27, no. 1, pp. 83–91, 2004.

[40] S. K. Mishra, "The most representative composite rank ordering of multi-attribute objects by the particle swarm optimization," Jan. 2009. [Online]. Available: http://dx.doi.org/10.2139/ssrn.1326386. Accessed on: Oct. 14, 2016.

[41] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, 2011, Art. no. 11.

[42] A. Ochiai, "Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions," *Bull. Jpn. Soc. Sci. Fish*, vol. 22, no. 9, pp. 522–525, 1975.

[43] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proc. Int. Symp. Softw. Test. Anal.*, 2001, pp. 199–209.

[44] L. S. Passos, R. Abreu, and R. J. Rossetti, "Empirical evaluation of similarity coefficients for multiagent fault localization," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 5, pp. 767–782, May 2017.

[45] L. S. Passos, R. Abreu, and R. J. F. Rossetti, "Sensitivity analysis of spectrum-based fault localisation for multi-agent systems," presented at the *Int. Workshop Principles of Diagn.*, 2014.

[46] J. Ren and R. N. Taylor, "Automatic and versatile publications ranking for research institutions and scholars," *Commun. ACM*, vol. 50, no. 6, pp. 81–85, 2007.

[47] M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2003, pp. 30–39.

[48] D. Roger and T. Tanimoto, "A computer program for classifying plants," *Science*, vol. 132, no. 3434, pp. 1115–1118, 1960.

[49] E. Rogot and I. D. Goldberg, "A proposed index for measuring agreement in test-retest studies," *J. Chronic Disease*, vol. 19, no. 9, pp. 991–1006, 1966.

[50] P. Russel and T. Rao, "On habitat and association of species of anopheline larvae in South-Eastern Madras," *J. Malaria Inst. India*, vol. 3, no. 1, pp. 153–178, 1940.

[51] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold, "Lightweight fault-localization using multiple coverage types," in *Proc. Int. Conf. Softw. Eng.*, 2009, pp. 56–66.

[52] W. A. Scott, "Reliability of content analysis: The case of nominal scale coding," *Public Opin. Q.*, vol. 19, no. 3, pp. 321–325, 1955.

[53] F. Steimann, M. Frenkel, and R. Abreu, "Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators," in *Proc. Int. Symp. Softw. Test. Anal.*, 2013, pp. 314–324.

[54] C. M. Tang, W. K. Chan, and Y. T. Yu, "Extending the theoretical fault localization effectiveness hierarchy with empirical results at different code abstraction levels," in *Proc. Annu. Comput. Softw. Appl. Conf.*, 2014, pp. 161–170.

[55] C. M. Tang, J. Keung, W. K. Chan, and Y. T. Yu, "DFL: Dual-service fault localization," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur.*, 2016, pp. 412–422.

[56] S. Wang, D. Lo, L. Jiang, Lucia, and H. C. Lau, "Search-based fault localization," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2011, pp. 556–559.

[57] M. Weiser, "Program slicing," in *Proc. Int. Conf. Softw. Eng.*, 1981, pp. 439–449.

[58] D. A. Wheeler, "Counting source lines of code (SLOC)," [Online]. Available: http://www.dwheeler.com/sloc. Accessed on: Oct. 14, 2016.

[59] W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *J. Syst. Softw.*, vol. 83, pp. 188–208, 2010.

[60] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Trans. Rel.*, vol. 63, no. 1, pp. 290–308, Mar. 2014.

[61] W. E. Wong, V. Debroy, and D. Xu, "Towards better fault localization: A crosstab-based statistical approach," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 3, pp. 378–396, May 2012.

[62] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.

[63] X. Xie, T. Y. Chen, F. C. Kuo, and B. Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, 2013, Art. no. 31.

[64] J. Xuan and M. Monperrus, "Learning to combine multiple ranking metrics for fault localization," in *Proc. IEEE Int. Conf. Softw. Maint. Evol.*, 2014, pp. 191–200.

[65] J. Xuan and M. Monperrus, "Test case purification for improving fault localization," in *Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 52–63.

[66] A. Zeller, "Isolating cause-effect chains from computer programs," in *Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2002, pp. 1–10.

[67] Z. Zhang, W. K. Chan, and T. H. Tse, "Fault localization based only on failed runs," *IEEE Comput.*, vol. 45, no. 6, pp. 64–71, Jun. 2012.

[68] Z. Zhang, W. K. Chan, T. H. Tse, B. Jiang, and X. Wang, "Capturing propagation of infected program states," in *Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2009, pp. 43–52.
[69] Z. Zhang, B. Jiang, W. K. Chan, T. H. Tse, and X. Wang, "Fault localization through evaluation sequences," *J. Syst. Softw.*, vol. 83, no. 2, pp. 174–187, 2010.

Chung Man Tang received the B.Sc. degree in computer studies and the M.Phil. degree in computer science from City University of Hong Kong, Hong Kong.

He is currently working toward the Ph.D. degree in the Computer Science Department, City University of Hong Kong, Hong Kong.

His research interests include software engineering and computers in education.

W. K. Chan (M'05) received the B.Eng., M.Phil., and Ph.D. degrees from The University of Hong Kong, Hong Kong.

He is an Associate Professor in the Department of Computer Science, City University of Hong Kong, Hong Kong. His current main research interests include program analysis and testing for concurrent software and systems. He has published extensively in venues such as TOSEM, TSE, TPDS, TSC, TRel, JSS, IST, STVR, CACM, Computer, ICSE, FSE, ISSTA, ASE, WWW, ISSRE, ICWS, QRS, ICDCS, and others.

Dr. Chan is the Special Issues Editor of the *Journal of Systems and Software.*

Yuen Tak Yu (M'98) received the Ph.D. degree from The University of Melbourne, Australia.

He is an Associate Professor in the Department of Computer Science, City University of Hong Kong, Hong Kong. His research interests include software testing, e-commerce, and computers in education. His publications have appeared in scholarly journals and leading international conferences, such as TOSEM, TSE, TRel, TSC, JSS, IST, Information Research, Computers and Education, ICSE, FSE, ISSRE, COMPSAC, QRS, ICCE, and others.

Dr. Yu is a past chair of the IEEE Hong Kong Section Computer Society Chapter.

Zhenyu Zhang (M'10) received the Ph.D. degree from The University of Hong Kong, Hong Kong.

He is an Associate Professor at the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China. His current research interests are program debugging and testing for software and systems, and the reliability issues of web-based services and cloud-based systems. He has published research results in venues such as TSE, Computer, ICSE, FSE, ASE, and WWW.