The Journal of Systems and Software 84 (2011) 885–905

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Non-parametric statistical fault localization☆

Zhenyu Zhang[a], W.K. Chan[b,*], T.H. Tse[c], Y.T. Yu[b], Peifeng Hu[d]

[a] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[b] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon Tong, Hong Kong
[c] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
[d] China Merchants Bank, Central, Hong Kong

ARTICLE INFO

Article history:
Received 26 February 2010
Received in revised form
21 December 2010
Accepted 30 December 2010
Available online 18 January 2011

Keywords:
Fault localization
Hypothesis testing
Parametric method
Non-parametric method

ABSTRACT

Fault localization is a major activity in program debugging. To automate this time-consuming task, many existing fault-localization techniques compare passed executions and failed executions, and suggest suspicious program elements, such as predicates or statements, to facilitate the identification of faults. To do that, these techniques propose statistical models and use hypothesis testing methods to test the similarity or dissimilarity of proposed program features between passed and failed executions. Furthermore, when applying their models, these techniques presume that the feature spectra come from populations with specific distributions. The accuracy of using a model to describe feature spectra is related to and may be affected by the underlying distribution of the feature spectra, and the use of a (sound) model on inapplicable circumstances to describe real-life feature spectra may lower the effectiveness of these fault-localization techniques. In this paper, we make use of hypothesis testing methods as the core concept in developing a predicate-based fault-localization framework. We report a controlled experiment to compare, within our framework, the efficacy, scalability, and efficiency of applying three categories of hypothesis testing methods, namely, standard non-parametric hypothesis testing methods, standard parametric hypothesis testing methods, and debugging-specific parametric testing methods. We also conduct a case study to compare the effectiveness of the winner of these three categories with the effectiveness of 33 existing statement-level fault-localization techniques. The experimental results show that the use of non-parametric hypothesis testing methods in our proposed predicate-based fault-localization model is the most promising.

1. Introduction

1.1. Background

Program debugging is a process to locate faults in faulty programs, repair the programs, and confirm that the repairs effectively remove the identified faults (Vessey, 1986). It cannot be avoided in any typical software development project. In practice, program debugging (including fault localization as one of the three activities) often takes a lengthy and manual procedure. Automated or semi-automated techniques that accurately locate the faults help improve the debugging process. Traditionally, a developer may iteratively and repetitively set up breakpoints through an integrated development environment, execute a faulty program over inputs, monitor how different slices of the program states change with the executions, and identify suspicious program elements. To help identify the suspicious program elements more effectively, a class of statistical fault-localization techniques has been proposed. Examples include Tarantula (Jones et al., 2002; Jones and Harrold, 2005), SOBER (Liu et al., 2005, 2006), CBI (Liblit et al., 2003, 2005), DES (Zhang et al., 2008, 2010), CP (Zhang et al., 2009b), and Ochiai (Abreu et al., 2009).

The basic intuition behind this class of techniques is that, if certain static or dynamic program features correlate the root cause (that is, the fault) with the observed failures, a technique may use a statistical analysis approach to reveal the correlations. The strengths of correlations between such program features and the presence (as well as absence) of the observed failures can be used as indicators of the degree to which some (suspicious) program features may explain the observed failures. Furthermore, since the selected program features can be mapped back to certain program elements, various indicators essentially provide different ways to assess the fault suspiciousness of various program elements. In DES (Zhang et al., 2008, 2010), for instance, we use an evalu-

☆ The research is supported in part by grants of the National Natural Science Foundations of China (project nos. 61003027 and 61073006), the Research Grants Council of Hong Kong (project nos. 111410, 123206, 123207 and 716507), and City University of Hong Kong (project no. 7002464).

* Corresponding author. Tel.: +852 2788 9684; fax: +852 2788 8614.
E-mail address: wkchan@cs.cityu.edu.hk (W.K. Chan).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.12.048

ation sequence of a predicate (Liblit et al., 2003) as a program element and its distribution (Liu et al., 2006) of decision outcomes (Lau and Yu, 2005) as a program feature. Moreover, using the correlation formula of SOBER (Liu et al., 2005), DES estimates the strengths of correlations between these program features and the observed failures in the dataset. DES further maps each evaluation sequence to the corresponding program predicate, and hence the fault-suspiciousness estimate of each program feature is translated linearly back to a fault-suspiciousness estimate of the mapped program predicate.

Many previous fault localization studies propose their own ways to determine the strengths of the above correlations. Typically, such techniques compare the program features obtained from a set of failed executions with the same program features obtained from a set of passed (or passed and failed) executions, and measure the correlation strengths accordingly. Some use this basic information to derive other heuristics to locate faults. In other words, in the core part, a technique typically looks for high contrasts between the program features obtained from the former set and those from the latter set (Jones et al., 2002; Jones and Harrold, 2005; Liu et al., 2005, 2006; Liblit et al., 2003, 2005; Zhang et al., 2008, 2010; Masri, 2009). In this contrast step, the model of the technique assumes that there are plenty of samples available. Thus, a technique of this class characterizes each sample set approximately by a pre-supposed distribution (say, a normal distribution by applying the law of large numbers). Such a distribution can be characterized by certain parameters (such as the mean and standard deviation for a normal distribution). For example, Tarantula compares the mean percentages of passed executions and failed executions that go through a particular statement as building blocks of its ranking formula to estimate the fault suspiciousness of the statement. Another example is that DES uses the distributions of evaluation biases (Liu et al., 2005) of evaluation sequences, and in its experiment, DES takes both the means and standard deviations as parameters in the ranking formula when integrated with SOBER.

The model of an existing technique in this class thus depends on a number of parameters, including the selection of sensitive program features to gauge the presence and absence of observed failures, the characterization of the distribution of each program feature, and the maximum number of executions to conduct the contrast step above. For ease of presentation, we will collectively refer to such existing techniques in this class as *parametric* statistical fault localization techniques.

Many previous studies, including Jones and Harrold (2005), Liu et al. (2005) and Zhang et al. (2009b), have asserted that the dynamic program features related to statements and predicates can be sensitive to the presence and absence of observed failures. Moreover, the maximum number of executions may noticeably affect the effectiveness (Liu et al., 2006). In the experiment presented in Zhang et al. (2009a), we have discovered that the distributions of evaluation biases for many program predicates are *far from* normal. In the subject programs of that experiment, for instance, the distributions of evaluation biases of almost 45% of the predicates on or closest to faulty statements, which are usually referred to as the most fault-relevant predicates, do not exhibit normal distributions with adequate confidence even though we may wish to lower our standard to accept it to be a normal distribution at a significance level of 50%. Our previous result thus implies that a technique which uses the parameters of pre-supposed distributions of program features in the contrast step to assess the fault suspiciousness of program subjects can be non-scientific.

1.2. Our work

The above discussion thus poses a series of interesting research questions. For instance, are some techniques of this class independent of the distributions of the selected program features so that they can be more reliably applied to a larger class of programs in general and scenarios encountered in the above contrast step in particular? Is such a distribution-independent technique effective? Is it efficient with respect to the state-of-the-art debugging-specific techniques, such as those mentioned above? We will answer these questions in the present paper.

We have proposed on the preliminary work (Hu et al., 2008) of this paper that a predicate-based statistical fault-localization technique can adopt a non-parametric hypothesis testing method as the procedure to determine the extent of differences in the contrast step mentioned above. For ease of presentation, we refer to such a procedure as the *core* of the fault-localization technique, and refer to a predicate-based fault-localization technique that uses a non-parametric hypothesis testing method as its core as a *non-parametric predicate-based fault-localization technique*.

In particular, in Hu et al. (2008), we use SOBER (Liu et al., 2005, 2006) as an example fault-localization technique but replaces SOBER's core by a standard non-parametric hypothesis testing method—the Mann–Whitney test (Mann and Whitney, 1947). The preliminary results of Hu et al. (2008) show that this non-parametric version of SOBER is more effective than the original SOBER in terms of *T*-score (Renieris and Reiss, 2003).

In this paper, we extend our preliminary work on studying whether non-parametric techniques can be superior to their parametric counterparts and propose a *predicate-based fault-localization framework*, which formulates the use of various hypothesis testing methods to compare the differences of program spectra in passed executions and failed executions. In our framework, we include three categories of hypothesis testing methods, namely, two non-parametric hypothesis testing methods—the Wilcoxon signed-rank test (Wilcoxon, 1945) and the Mann–Whitney test (Mann and Whitney, 1947), two parametric hypothesis testing methods—the Student's *t*-test (Devore, 2008) and the *F*-test (Devore, 2008), and two debugging-specific hypothesis testing methods taken from existing predicate-based fault-localization techniques—CBI (Liblit et al., 2003, 2005) and SOBER (Liu et al., 2006). We use the names "TC1", "TC2", and "TC3" to refer to these three categories of techniques. We apply the three categories of hypothesis testing methods in our framework to synthesize six predicate-based fault-localization techniques, and investigate their efficacy, scalability, and efficiency issues. Further in the paper, we will also include an additional category "TC4" to represent 33 statement-level fault-localization techniques, for comparisons with the winner among TC1, TC2, and TC3 on fault-localization effectiveness.

We further introduce our motivation of the above comparison setting as follows. The Wilcoxon signed-rank test is frequently cited in pair with the Mann–Whitney test in statistics. Moreover, the Wilcoxon signed-rank test is popularly used in statistics as an alternative to Student's *t*-test (Devore, 2008) when the population cannot be assumed to be normally distributed (Lomax, 2007), and we therefore include also Student's *t*-test in our investigation. We further note that *F*-test (Devore, 2008) is a parametric alternative when the variances are equal, while Student's *t*-test does not have this restriction (Lomax, 2007). Hence, we also include *F*-test in our investigation. Lastly, we include two representative debugging-specific techniques, namely, CBI and SOBER, so that we can determine how well the standard parametric and non-parametric fault-localization techniques perform. A set of 33 statement-level techniques have first been summarized in Naish et al. (in press). We choose to compare with them because of inadequate previous research reported in comparisons between predicate-level fault-localization techniques and statement-level fault-localization techniques.

We first set up a controlled experiment to evaluate and compare the *six* techniques in TC1, TC2, and TC3 in multiple dimensions,

including effectiveness, scalability, and efficiency. Following previous studies including SOBER (Liu et al., 2006) and DES (Zhang et al., 2009a), we use the Siemens suite as subject programs. Originally, we planned to include more subjects. However, because we have included six techniques and target to evaluate these techniques in much wider aspects than most previous experiments, we estimate that our experiment would require much time and effort to conduct and the resulting data would be hard to analyze. To balance between our resource constraints and the scale of the experiment, we finally settle with the use of the Siemens suite as subjects. After that, we use a median-sized program space that has a set of real-life faults as an additional program subject to evaluate the effectiveness of the winner techniques among TC1, TC2, and TC3 with *33* statement-level fault-localization techniques (TC4). We include many TC4 techniques in this case study because latest research in statistical fault localization almost exclusively uses statement-level techniques. However, since we include so many TC4 techniques in the case study, we are unable to compare all the techniques in terms of scalability, efficiency, or other practical scenarios owing to effort and resource constraints.

We have four conclusions from the experimental results:

(i) A predicate-based fault-localization technique using a non-parametric core is more effective than one that uses a parametric or debugging-specific core.
(ii) When increasing the maximum number of executions in fault-localization techniques, the advantage of using a non-parametric core becomes more significant.
(iii) A predicate-based fault-localization technique using a non-parametric core is more efficient than one that uses a debugging-specific core.
(iv) A predicate-based fault-localization technique using a non-parametric core is more effective than existing statement-level fault-localization techniques.

The main contribution of this paper is fourfold. First, it proposes a framework for statistical predicate-based fault-localization techniques, which uses hypothesis testing as the core concept. This is the first time that such a framework is proposed. It also differs from our previous work. Second, it proposes to characterize a predicate-based such technique (under our framework) based on whether their hypothesis testing methods are parametric, non-parametric, or debugging-specific. Our empirical study shows that the techniques among various categories differ very much from one another in efficacy and scalability. Third, it reports the first empirical study to validate whether a predicate-based fault-localization technique using a non-parametric core is more effective and scalable than a technique using a parametric or debugging-specific core. The empirical results show that the use of a non-parametric core for predicate-based statistical fault localization can be promising and outperforms the latter two kinds of cores in terms of effectiveness. In addition, if we deem existing debugging-specific methods to be efficient, the use of a non-parametric core is found to be comparable in efficiency. Fourth, it reports the first case study to compare the effectiveness of predicate-based fault-localization technique using a non-parametric core with statement-level fault-localization techniques. The empirical results interestingly show that the use of a non-parametric core for predicate-based statistical fault localization can be promising and outperforms the evaluated statement-level fault-localization techniques.

The remainder of the paper is organized as follows: We first review related work in Section . Then, in Section 3, we discuss our approach and formally present the research questions to be studied. Next, we report our empirical evaluation in Section 4, followed by a conclusion in Section 5.

2. Related work

In this section, we review related work on fault localization research.

2.1. Program slicing

Program slicing (Weiser, 1984) is a code-based technique widely used in program debugging research (Tip, 1995). A slice refers to a set of statements in a program that may affect the computed values at some location, such as particular occurrences of a variable. Slicing techniques can be static (Weiser, 1984) or dynamic (Agrawal and Horgan, 1990; Korel and Laski, 1988). Statistical fault localization provides assessments of individual program features, which may annotate the slices to make debugging more effective. Ottenstein and Ottenstein (1984) develop a program dependence graph (PDG) to reduce the computation of static slices of a sequential program to a reachability problem in PDG. Horwitz et al. (1990) extend the technique to inter-procedural slicing. However, the typical size of a static slice for a program can be one-third of the program (Binkley et al., 2007). It may not be useful to present such a large piece of code for developers to look for faulty statements. To address this problem, researchers study dynamic analysis techniques to reduce the size of a slice. Chen and Cheung (1993) propose dynamic dicing and the related strategies to construct dynamic dices. Gupta et al. (2005) propose to use forward dynamic slicing to narrow down slices. They further integrate forward dynamic slicing with backward one (Zhang et al., 2006) to prune irrelevant statements. Slicing techniques can also be integrated with statistical fault localization so that only those program statements that exist in a set of slices will be examined to locate faults. The main difference between our non-parametric predicate-based fault-localization techniques and program slicing techniques is that the former is based on coverage information of program executions while the latter may need additional program execution context information about any possible statement.

2.2. Predicate-based statistical fault-localization techniques

Another approach to debugging is to use the statistics collected from test case executions. Collofello and Cousins (1987) pioneer the use of test cases for fault localization. Earlier research (Agrawal et al., 1991; Korel, 1988; Korel and Laski, 1988), however, only utilizes failed test cases. Later research such as Jones et al. (2002) evaluates this approach as ineffective. Subsequent research such as Liu et al. (2005) switches to use both the passed and failed test cases in localizing faults. Harrold et al. (2000) evaluate nine kinds of program features, including path count, data-dependency count, and execution trace. Among them, the execution trace spectrum is most widely used in debugging. Their study surprisingly shows that the use of data-dependency count is less effective than many other program features. A later study (Yu et al., 2008) shows, however, that by applying a proper contrast step, the use of data-dependency counts can be more effective than that of control-dependency counts. CBI (Liblit et al., 2003, 2005) and SOBER (Liu et al., 2005, 2006) are two representative techniques that relate to control-dependency information. More specifically, they make use of the execution spectra information of program predicates set in branch statements and so on, and hence we call them "predicate-based statistical fault-localization techniques". CBI compares the probability that a program fails when a predicate is ever evaluated to be true with the probability that the program fails when the predicate is ever evaluated. The technique uses this difference as the primary program feature to identify the positions of the predicates related to faults. SOBER further proposes to use the actual probability that a predicate is evaluated to be true, which they call the evaluation

bias, as the program feature. It contrasts the evaluation biases of each predicate in passed and failed executions to locate predicates that are related to faults. After locating suspicious predicates, these methods recommend programmers to search for faults around the located suspicious predicates in the program.

Based on the proposed framework of predicate-based fault-localization techniques in this paper, both parametric/non-parametric hypothesis testing methods and the debugging-specific hypothesis testing methods in CBI and SOBER can be described. The main difference between parametric/non-parametric hypothesis testing methods and debugging-specific hypothesis testing methods is that the former uses mature standard mathematical methods while the latter uses self-proposed methods that are specifically designed for fault localization.

2.3. Statement-level statistical fault-localization techniques

Rather than locating suspicious predicates, some techniques directly assess the suspiciousness of statements and target at locating faulty statements. They are so called "statement-level fault-localization techniques". Jones et al. (2002) and Jones and Harrold (2005) propose *Tarantula* to rank every statement according to its fault suspiciousness. As mentioned in Section 1, Tarantula uses the mean values of the execution count statistics as building blocks. It then uses these building blocks to compose a formula to assess fault suspiciousness. Baudry et al. (2006) observe that some statements are always executed by the same set of executions. They use an evolutionary approach to select a subset of the given execution set, aiming to achieve better diversity in terms of dynamic basic blocks. Applying the algorithm developed in Jones et al. (2002) to rank statements, Baudry et al. (2006) empirically show that their approaches require fewer test cases to achieve the same fault-localization effectiveness. Abreu et al. (2009) further show empirically that a technique can achieve almost the same fault-localization accuracy by using a few failed executions. Several more advanced approaches use statistical measures for behaviors related to program failures. Jones et al. (2007) further extend Tarantula so that it can be applied when multiple developers are available to debug the program independently. Our present work provides another dimension to optimize the use of test executions. Observing that individual executions of the same statements may have different contributions to indicate faulty statements when they are used together, Wong et al. (2007) propose to use a utility function to calibrate the contribution of each passed execution when computing the fault relevance of executed statements. Their techniques are shown empirically to outperform Tarantula. They further define a series of heuristics based on different marginal contributions of additional failed executions and passed executions (Wong et al., 2010).

Naish et al. (in press) have given a summary of the following statement-level fault-localization techniques: Tarantula (Jones et al., 2002) has been introduced in the last paragraph. Duarte et al. (1999), Dice (1945), Ochiai (1957), Russell & Rao (1940), Anderberg (1973), Simple Matching and Ochiai2 (da Silva Meyer et al., 2004), and Rogers & Tanimoto (1960) are originally used for classifications in the botany domain. Overlap (Krause, 1973) is a general version of Ochiai (1957). Others include Euclid (Krause, 1973), M1 and M2 (Everitt, 1978) (where the names M1 and M2 are proposed by Naish et al., in press), Kulczynski1 and Kulczynski2 (Lourenço et al., 2004), Hamann and Sokal (Lourenço et al., 2004) (which are used in data clustering), Jaccard (1901) (originally used in the botany domain and later used in data clustering), Hamming (1950) (originally used for error detection codes), Goodman & Kruskal (1954), Scott (1955), Cohen (1960), Fleiss (1965), Geometric mean (Maxwell and Pilliner, 1968), Rogot1, Rogot2, harmonic mean, and arithmetic mean (Rogot and Goldberg, 1966) (originally used as biometrics metrics), Ample (Dallmeier et al., 2005) (originally used for fault localization; following Naish et al., in press, the Ample technique used in our paper is taken from their modified version Abreu et al., 2007), Zoltar (González-Sánchez, 2007), and Wong1, Wong2, and Wong3 (Wong et al., 2007) (originally used for fault localization). Interested readers may follow Table 1 to obtain the exact formulas. A similarity among all these techniques is that they share the same input format and generate outputs of the same format. An essential difference among them lies in the ranking formulas they use to assess the suspiciousness of a statement related to faults. The study on whether their models have any non-parametric property is outside the scope of this paper. In this paper, we design non-parametric predicate-based fault-localization techniques and compare them empirically with these statement-level techniques to gauge whether predicate-based techniques may have comparable effectiveness with statement-level techniques in terms of fault localization.

2.4. Other fault-localization techniques

Renieris and Reiss (2003) observe that it may be more useful to compare failed executions with "similar" ones, where the "similarity" of a pair of execution is measured by the edit distance between the two execution sequences. Their approach, however, does not use statistical methods to pinpoint faulty positions from the results of a pair of similar executions. Apart from using statistics, some proposals adopt an iterative elimination approach. For instance, *delta debugging* simplifies the failed test cases and yet preserves the failures (Zeller and Hildebrandt, 2002), producing cause–effect chains (Zeller, 2002) and linking them to suspicious statements (Cleve and Zeller, 2005). Other heuristics have also been studied, such as the use of Jaccard distance (Abreu et al., 2009). Debroy and Wong (2009) and Wong et al. (2008) propose a crosstab method to compute the fault suspiciousness of statements and focus on programs having multiple faults.

Arumuga Nainar et al. (2007) further extend CBI to address compound Boolean expressions. They show that the accuracy of CBI changes significantly when compound Boolean expressions are involved. Zhang et al. (2008) conduct an empirical study to show that the short-circuit rules in evaluating Boolean expressions in predicates affect the effectiveness of fault-localization techniques, and that the results of CBI can be improved using short-circuit information in the form of evaluation sequences. Chilimbi et al. (2009) propose Holmes, which uses fragments of paths rather than individual predicates to locate faults iteratively. Our previous paper (Zhang et al., 2009a) finds empirically that the evaluation biases of many predicates are not distributed normally. Our preliminary work (Hu et al., 2008) of the present paper proposes to use the Mann–Whitney non-parametric hypothesis testing method to replace the debugging-specific hypothesis testing method in SOBER, and conducts experiments to compare its effectiveness with SOBER and CBI. The empirical results show that such a technique is promising. In this paper, we propose a generic framework of predicate-based fault-localization techniques, apply many non-parametric, parametric, or debugging-specific hypothesis testing methods to it to generate predicate-based fault-localization techniques, and empirically evaluate them. Our framework is general, and its application is not limited to the techniques presented in this paper.

Based on the suspiciousness estimation obtained from a contrast step, CP (Zhang et al., 2009b) constructs a probabilistic control flow graph and a propagation model for the faulty program with a view to capturing the propagation of infected and abstract states extracted from the given set of program executions to locate faults. Besides, a few studies (Hao et al., 2010, 2006; Jiang et al., 2009) focus on optimizing the sizes of input test data to facilitate effec-

Table 1
The 33 statement-level fault-localization techniques listed in Naish et al. (in press).

Name	Formula
Jaccard (1901)	$a_{ef}/(a_{ef}+a_{nf}+a_{ep})$
Anderberg (1973)	$a_{ef}/(a_{ef}+2(a_{nf}+a_{ep}))$
Sørensen-Dice (Duarte et al., 1999)	$a_{ef}/(a_{ef}+a_{nf}+a_{ep})$
Dice (1945)	$2a_{ef}/(a_{ef}+a_{nf}+a_{ep})$
Kulczynski1 (Lourenço et al., 2004)	$a_{ef}/(a_{nf}+a_{ep})$
Kulczynski2 (Lourenço et al., 2004)	$\frac{1}{2}\left(\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ef}}{a_{ef}+a_{ep}}\right)$
Russell & Rao (1940)	$a_{ef}/(a_{ef}+a_{nf}+a_{ep}+a_{np})$
Hamann (Lourenço et al., 2004)	$(a_{ef}+a_{np}-a_{nf}-a_{ep})/(a_{ef}+a_{nf}+a_{ep}+a_{np})$
Simple Matching (da Silva Meyer et al., 2004)	$(a_{ef}+a_{np})/(a_{ef}+a_{nf}+a_{ep}+a_{np})$
Sokal (Lourenço et al., 2004)	$2(a_{ef}+a_{np})/(2a_{ef}+2a_{np}+a_{nf}+a_{ep})$
M1 Everitt, 1978	$(a_{ef}+a_{np})/(a_{nf}+a_{ep})$
M2 Everitt, 1978	$a_{ef}/(a_{ef}+a_{np}+2a_{nf}+2a_{ep})$
Rogers & Tanimoto (1960)	$(a_{ef}+a_{np})/(a_{ef}+a_{nf}+2a_{nf}+2a_{ep})$
Goodman & Kruskal (1954)	$(2a_{ef}-a_{nf}-a_{ep})/(2a_{ef}+a_{nf}+a_{ep})$
Hamming (1950)	$a_{ef}+a_{np}$
Euclid (Krause, 1973)	$\sqrt{a_{ef}+a_{np}}$
Ochiai (1957)	$\frac{a_{ef}}{sqrt(a_{ef}+a_{nf})(a_{ef}+a_{ep})}$
Overlap (Krause, 1973)	$a_{ef}/min(a_{ef}, a_{nf}, a_{ep})$
Tarantula (Jones et al., 2002)	$\frac{a_{ef}/(a_{ef}+a_{nf})}{a_{ef}/(a_{ef}+a_{nf})+a_{ep}/(a_{ep}+a_{np})}$
Zoltar (González-Sánchez, 2007)	$\frac{a_{ef}}{a_{ef}+a_{nf}+a_{ep}+\frac{10000a_{nf}a_{ep}}{a_{ef}}}$
Ample (Dallmeier et al., 2005)	$\lvert\frac{a_{ef}}{a_{ef}+a_{nf}}-\frac{a_{ep}}{a_{ep}+a_{np}}\rvert$
Wong1 (Wong et al., 2007)	a_{ef}
Wong2 (Wong et al., 2007)	$a_{ef}-a_{ep}$
Wong3 (Wong et al., 2007)	$a_{ef}-\begin{cases}a_{ep} & a_{ep}\le 2\\ 2+0.1(a_{ep}-2) & 2\le a_{ep}\le 10\\ 2.8+0.001(a_{ep}-10) & a_{ep}\ge 10\end{cases}$
Ochiai2 (da Silva Meyer et al., 2004)	$\frac{a_{ef}a_{np}}{\sqrt{(a_{ef}+a_{ep})(a_{np}+a_{nf})(a_{ef}+a_{nf})(a_{ep}+a_{np})}}$
Geometric mean (Maxwell and Pilliner, 1968)	$\frac{a_{ef}a_{np}-a_{nf}a_{ep}}{\sqrt{(a_{ef}+a_{ep})(a_{np}+a_{nf})(a_{ef}+a_{nf})(a_{ep}+a_{np})}}$
Harmonic mean (Rogot and Goldberg, 1966)	$\frac{(a_{ef}a_{np}-a_{nf}a_{ep})((a_{ef}+a_{ep})(a_{np}+a_{nf})+(a_{ef}+a_{nf})(a_{ep}+a_{np}))}{(a_{ef}+a_{ep})(a_{np}+a_{nf})(a_{ef}+a_{nf})(a_{ep}+a_{np})}$
Arithmetic mean (Rogot and Goldberg, 1966)	$\frac{2a_{ef}a_{np}-2a_{nf}a_{ep}}{(a_{ef}+a_{ep})(a_{np}+a_{nf})+(a_{ef}+a_{nf})(a_{ep}+a_{np})}$
Cohen (1960)	$\frac{2a_{ef}a_{np}-2a_{nf}a_{ep}}{(a_{ef}+a_{ep})(a_{np}+a_{nf})+(a_{ef}+a_{nf})(a_{ep}+a_{np})}$
Scott (1955)	$\frac{4a_{ef}a_{np}-4a_{nf}a_{ep}-(a_{nf}-a_{ep})^2}{(2a_{ef}+a_{nf}+a_{ep})(2a_{np}+a_{nf}+a_{ep})}$
Fleiss (1965)	$\frac{4a_{ef}a_{np}-4a_{nf}a_{ep}-(a_{nf}-a_{ep})^2}{(2a_{ef}+a_{nf}+a_{ep})+(2a_{np}+a_{nf}+a_{ep})}$
Rogot1 (Rogot and Goldberg, 1966)	$\frac{1}{2}\left(\frac{a_{ef}}{2a_{ef}+a_{nf}+a_{ep}}+\frac{a_{np}}{2a_{np}+a_{nf}+a_{ep}}\right)$
Rogot2 (Rogot and Goldberg, 1966)	$\frac{1}{4}\left(\frac{a_{ef}}{a_{ef}+a_{ep}}+\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{np}}{a_{np}+a_{ep}}+\frac{a_{np}}{a_{np}+a_{nf}}\right)$

tive fault localization. For example, Jiang et al. (2009) investigate the effect of test case prioritization techniques on effectiveness of fault-localization techniques. Many other methodologies, such as training a neural network (Wong and Qi, 2009) and integrating with model checking (Griesmayer et al., 2009), have also been proposed to improve the effectiveness of fault localization.

3. Our study

In this section, we first review the basic terminology used in our study and then present a model to assess the fault relevance of predicates to facilitate the localization of faults, before posing the research questions to be addressed by this paper.

3.1. Preliminaries

Here, we revisit the notion of program predicates (Liblit et al., 2003, 2005) and evaluation biases (Liu et al., 2005, 2006). In predicate-based statistical fault localization, *predicates* are the program element in focus. Liblit et al. (2003, 2005) address three types of program locations and the associated set of predicates as follows:

1. *Branches*: At each decision statement (such as an "if" or "while" statement), CBI tracks, via a pair of program predicates, whether the conditional "true" and "false" branches have been taken. SOBER (Liu et al., 2006) further collects the execution frequency of the branches.
2. *Returns*: At each return statement (of a function module), six predicates are tracked to find whether the returned value r satisfies $r<0$, $r<=0$, $r>0$, $r>=0$, $r==0$, or $r!=0$. Both CBI and SOBER collect evaluations of these predicates.
3. *Scalar-pairs*: To monitor the relationship of a variable to another variable or to a constant in each assignment statement, six predicates (similar to those for return statements above) are tracked by CBI. For example, for an assignment statement $x=y$, the following six predicates are tracked: $x>y$, $x>=y$, $x<y$, $x<=y$, $x==y$, and $x!=y$.

SOBER, however, experimentally verifies and concludes that *not* tracking these predicates will not degrade the quality of fault localization when using the Siemens suite. We will exclude scalar-pairs from our experiment with a view to a fair comparison.

Given a predicate, a technique may sample the outcomes of its component conditions at each execution. One may further use the differences among the sampled outcomes to facilitate fault localization. Liu et al. (2005) define the concept of *evaluation bias* to support the description of such differences.

Definition 1 *(Evaluation bias (Liu et al., 2005)).* Let n_t be the number of times that a predicate P has been evaluated to be "true" in an execution, and n_f the number of times that it has been evaluated to be "false" in the same execution. $\pi(P)=n_t/(n_t+n_f)$ is called the *evaluation bias* of predicate P in this particular execution.

3.2. Predicate-based statistical fault-localization framework

First, we present the conceptual model behind our statistical fault-localization framework. In this way, the framework can be initialized by using different hypothesis testing methods to synthesize different predicate-based fault-localization techniques.

We first model a faulty program by a set of predicates $\{P_1, P_2, \ldots, P_m\}$, and use T_p and T_f to denote the set of successful test cases and the set of failed test cases, respectively. We use $E^i(r)$ to denote the evaluation bias of predicate p_i in a program execution over test case r. Based on this information, our target is to assess the fault suspiciousness of predicate P_i being related to fault by analyzing $E^i(r)$. Once we obtain these fault-suspiciousness values for different predicates, debuggers may use such information to assist them to locate faults. Different executions may give different evaluation bias values for the same predicate. Hence, we use a random variable X^i to represent the evaluation bias $E^i(r)$, and use $f(X^i \mid T_p)$ to represent the probability density function of X^i for the set T_p of successful test cases and $f(X^i \mid T_f)$ to represent that for the set T_f of failed test cases.

Many previous studies (Hu et al., 2008; Liblit et al., 2003, 2005; Liu et al., 2005, 2006; Zhang et al., 2008, 2009a, 2010) observe that the stronger the correlation between a predicate P_i and a fault, the larger will be the difference between $f(X^i \mid T_p)$ and $f(X^i \mid T_f)$. Our methodology will, therefore, use the difference between $f(X^i \mid T_p)$ and $f(X^i \mid T_f)$ to assess the suspiciousness of predicate P_i, expressed as follows:

$$R(P_i) = \mathrm{Diff}(f(X^i|T_p), f(X^i|T_f))$$

We then review our predicate-based statistical fault-localization model. Following our previous work (Hu et al., 2008; Zhang et al., 2009a), we measure the difference between the two probability density functions $f(X^i \mid T_p)$ and $f(X^i \mid T_f)$ by conducting a hypothesis testing method to test the following null hypothesis:

H_0: The evaluation biases of predicate P_i for the set T_p of successful test cases and those for the set T_f of failed test cases come from the same population.

The p-value of a hypothesis testing method is the probability of that the population is at least as extreme as the observed result. Thus,

$$\mathrm{Diff}(f(X^i|T_p), f(X^i|T_f)) = p\text{-value of hypothesis } H_0 \quad (1)$$

For the program feature spectra in failed executions and passed executions, there is unfortunately no scientific support for the mapping between the similarity of their distributions and the magnitudes of the p-values for the hypothesis testing of their distributions. However, the smaller the p-value, the less confident are we that the two sets of samples come from the same population. We choose to use our confidence on whether the two sets of sample come from the same population as a measurement of fault suspiciousness. A fault-localization technique may sort the predicates in ascending order of p-values. Such an ordered predicate list is helpful for developers to locate faults in programs (Liblit et al., 2003, 2005; Liu et al., 2005, 2006; Renieris and Reiss, 2003; Zhang et al., 2008, 2009a).

3.3. Research questions

To measure the difference between two sample sets H and H', parametric hypothesis testing can be meaningfully applied only if the following assumptions (Lowry, 2006) hold:

$N1$: The two sample sets are randomly and independently drawn from the source population.
$N2$: The measurements in both sample sets have the same interval scales.
$N3$: The source population(s) can reasonably be assumed to have a known distribution.

When any of these three assumptions does not hold, a non-parametric testing method should be used instead. As shown by our previous work in Zhang et al. (2009a), the source populations of evaluation bias of predicates are indeed far from being normally distributed (as often implicitly assumed). The property of non-parametric hypothesis testing also frees us from the need to use artificial configuration parameters and allows a technique to use fewer samples than their parametric counterpart to assess the difference.

We now present the details of our framework to facilitate further elucidation. First, we classify statistical fault-localization techniques into several categories, as shown in Table 2. The first category TC1 refers to predicate-based techniques that use debugging-specific parametric testing methods (such as those used in CBI and SOBER). The second category TC2 refers to predicate-based techniques that use standard parametric hypothesis testing methods, such as F-test and Student's t-test (or t-test for short). The third category TC3 refers to predicate-based techniques that use standard non-parametric hypothesis testing methods, such as the Mann–Whitney test and Wilcoxon Signed-rank test. The fourth

Table 2
Classification of statistical fault-localization techniques.

Category	Type (predicate-based)		Example tests (References)
	Parametric or non-parametric	Standard or debugging-specific	
TC1	Parametric*	Debugging-specific	Tests used in CBI (Liblit et al., 2003, 2005) and SOBER (Liu et al., 2005, 2006)
TC2	Parametric	Standard	*F*-Test (Devore, 2008) and *t*-test (Devore, 2008)
TC3	Non-parametric	Standard	Mann–Whitney test (Mann and Whitney, 1947) and Wilcoxon Signed-rank test (Wilcoxon, 1945)

Category	Type	Examples (References)
TC4	Statement-level	Tarantula (Jones et al., 2002) and Jaccard (Jaccard, 1901)

*It is not clear whether the tests used in CBI and SOBER should be categorized as *parametric*. However, we tend to consider the tests in CBI and SOBER as parametric because they use parametric numerical methods without knowing the distribution of the execution spectra of the predicates. For example,
(i) CBI uses harmonic means in its computation.
(ii) CBI approximates the frequency of a predicate being exercised in a program run in terms of a 0/1 status (depending on whether it has been exercised in a program run).
(iii) SOBER sets the evaluation bias of a predicate to a theoretical mean value 0.5 (by taking the mean value of 0 and 1) if it is never evaluated.
(iv) SOBER assumes a normal distribution in one of its computation steps.

category TC4 refers to statment-level techniques such as Tarantula and Jaccard.

We design the following research questions to find the properties of TC1, TC2, and TC3, compare the effects of using them in our framework, and compare with TC4.

*Q*1: Compared with TC1, is TC2 more effective?
*Q*2: Compared with TC2, is TC3 more effective?
*Q*3: Compared with TC1, is TC3 more effective?

Research question *Q*1 essentially asks, when parametric testing methods are used, whether a standard one (TC2) is better than a debugging-specific one (TC1). Research question *Q*2 asks, when standard statistical testing methods are used, is a non-parametric one (TC3) better than a parametric one (TC2)? Research question *Q*3 is similarly posed to directly compare TC3 techniques with TC1.

We further study the research question *Q*4 to compare the effectiveness of the winner among TC1, TC2, and TC3 with existing statement-level fault-localization techniques.

*Q*4: How does the winner among TC1, TC2, and TC3 compare with TC4 in terms of effectiveness?

Next, we study the scalability issue. There seems to be a common belief that, with increasingly more data, the population can be more closely approximated by a normal distribution (Liu et al., 2005), and hence the adoption of parametric tests is better justified. Should this be the case, according to statistics theories, parametric tests (if applicable) will provide more accurate results than non-parametric ones. Hence, we are also interested in the effect of a larger number of test cases on the relative effectiveness of the techniques. We state this as research question *Q*5 below. Finally, we study the efficiency of the three classes of techniques, as stated in research question *Q*6.

*Q*5: When the number of test cases increases, are TC2 and TC3 techniques more effective than TC1 in localizing faults?
*Q*6: Are TC1, TC2, and TC3 techniques comparable in efficiency?

4. Experiment

This experiment consists of three parts: (a) An effectiveness comparison among TC1, TC2, and TC3. This part is to study which of parametric, non-parametric, and debugging-specific hypothesis testing methods gives predicate-based fault-localization techniques the best effectiveness in locating faults in programs. (b) An effectiveness comparison between TC4 and the winner among TC1, TC2, and TC3. This part is to study whether a good predicate-based fault-localization technique can be as effective as or more effective than statement-level fault-localization techniques in locating faults in programs. (c) Scalability and efficiency analysis of TC1, TC2, and TC3. This part is to investigate the scalability and efficiency issues of predicate-based fault-localization techniques. We do not include TC4 techniques in this part because previous work has studied these issues on TC4 and we are limited by our resources to repeat them.

4.1. Subject programs

Our experiment uses the seven programs in the Siemens suite, namely, tcas, tot_info, replace, print_tokens, print_tokens2, schedule, and schedule2, as well as one real-life program space, all downloaded from the Software-artifact Infrastructure Repository (SIR) (Do et al., 2005).

Each subject program in the Siemens suite has 7–41 faulty versions, each version being hand-seeded with one fault. Table 3 shows the descriptive statistics of each subject program, including the number of faulty versions, number of executable lines of code (LOC), number of test cases in the pool, and percentage of failed test cases among all test cases. The table also shows the minimum, maximum, and median perceived failure rates of the faulty versions of each subject program over the test pool, together with their fault types and code excerpts. For example, tcas has 41 faulty versions, each version consisting of 133–137 LOCs. For this program, 1608 test cases are available, of which 2.4% are failed test cases. Faulty version v12, which contains a "*wrong logic or relational operators*" fault, has a minimum perceived failure rate of (0.001).[1] In other words, among all the faults in the 41 faulty versions, the one in version v12 has the minimum failure rate of 0.001. According to *orthogonal defect classification* (Durães and Madeira, 2006), this fault occurs frequently in real-life programs. It belongs to the *Check* class (see Table 4), which constitutes 36.1% of occurrences among all classes of faults in the subject programs. The fault can further be classified under the fault type *C2* (see Table 4), which represents 52.9% of all fault occurrences within the *Check* class of faults in the Siemens suite.

The program space is an interpreter for an array definition language (ADL). It reads an ADL file, parses it, checks the consistency according to ADL grammar, and outputs a list of array elements (or error messages). According to the original version in the SIR repository (Do et al., 2005), the program consists of 6218 executable lines of code. It is attached with a test suite containing 13,496 test cases and 38 faulty versions, each of which contains a real fault. The explanations for the corresponding data section in Table 3 is similar to those of the last paragraph.

[1] A failing rate is defined as the number of failed test cases in a test pool over the total number of test cases in the same pool.

Table 3
Descriptive statistics of subject programs.

Programs	No. of Executable Faulty Versions	No. of LOC	No. of Test Cases	Percentage of Failed Test Cases
print_tokens & print_tokens2	17	341–354	4115 - 4130	1.7%, 5.4%
minimum failure rate = 0.001 print_tokens v1 /* *Wrong branching around statements* */ /* *case 16 : ch=get_char(...); case 25 : case 32 : token_ptr->token_id=special(next_st); */* *224: case 16 : case 32 : ch=get_char(...); case 25 : token_ptr->token_id=special(next_st);*				
maximum failure rate = 0.125 print_tokens2 v6 /* *Wrong logic or relational operands* */ *358: if(isdigit(*(str+i+1))) /* i+1 should be i */*				
median failure rate = 0.042 print_tokens2 v10 /* *Wrong logic or relational operands* */ *380: { while (*(str /* str should be str+i */)!='\0')*				
replace	32	508–515	5542	2.0%
minimum failure rate = 0.0001 replace v15 /* *Wrong logic or relational operands* */ *241: result = i + 1; /* i+1 should be i */*				
maximum failure rate = 0.035 replace v19 /* *Missing assignment* */ *514: /* result = */ getline(line, MAXSTR, &result);*				
median failure rate = 0.006 replace v14 /* *Missing OR-term/AND-term* */ *370: if ((lin[*i] != NEWLINE) /* && (!locate(lin[*i], pat, j+1)) */)*				
schedule & schedule2	19	261–294	2650 - 2710	2.4%, 3.2%
minimum failure rate = 0.001 schedule2 v5 /* Missing the whole if statement */ *111: /* if(prio < 1) return(BADPRIO); */*				
maximum failure rate = 0.116 schedule v7 /* Missing the whole if statement */ *210: /* if(ratio == 1.0) n--; */*				
median failure rate = 0.011 schedule v4 /* Wrong logic or relational operands */ *207: if (count > 1) /* 1 should be 0 */ {*				
tcas	41	133–137	1608	2.4%
minimum failure rate = 0.001 tcas v12 /* *Wrong logic or relational operators* */ *118: enabled = High_Confidence || /* || should be && */ (Own_Tracked_Alt_Rate <= OLEV) && (Cur_Vertical_Sep > MAXALTDIFF);*				
maximum failure rate = 0.182 tcas v27 /* *Missing OR-term/AND-term* */ *118: enabled = High_Confidence && (Own_Tracked_Alt_Rate <= OLEV) /* && (Cur_Vertical_Sep > MAXALTDIFF) */;*				
median failure rate = 0.021 tcas v10 /* *Wrong logic or relational operators* */ *105: return (Own_Tracked_Alt <= /* <= should be < */ Other_Tracked_Alt);*				
tot_info	23	272–274	1052	5.6%
minimum failure rate = 0.001 tot_info v23 /* *Wrong logic or relational operands* */ *215: for (n = 0 /* 0 should be 1 */; n <= ITMAX; ++n)*				
maximum failure rate = 0.087 tot_info v7 /* *Wrong logic or relational operators* */ *378: if (pi >= /* >= should be > */ 0.0)*				
median failure rate = 0.017 tot_info v2 /* *Wrong logic or relational operators* */ *85: if (scanf(" %ld", &x(i,j)) == /* == should be != */ 0)*				
space	28	6218	13496	14.8%
minimum failure rate = 0.0005 space v22 /* *Missing "if" statement* */ *7368: /* a block may cause infinite loop */*				
maximum failure rate = 0.937 space v29 /* *Wrong assignment* */ *4879: strcpy(Keywords[84], "P1 VAL"); /* should be P1 ET*				
median failure rate = 0.052 space v14 /* *Missing assignment* */ *8805: /*error = */ (GetKeyword(Keywords[88], curr_ptr));*				

Table 4
Important fault types in subject programs.

Class	Fault type (Durães and Madeira, 2006)
Assignment (43.9%)	A1: Missing assignment (9.6%)
	A2: Wrong/extraneous assignment (37.0%)
	A3: Wrong assigned variable (46.7%)
	A4: Wrong data types or conversion (6.4%)
Check (36.1%)	C1: Missing OR-term/AND-term (45.0%)
	C2: Wrong logic or relational operators (52.9%)
	C3: Missing branching around statements (1.9%)
Interface (4.2%)	I1: Wrong return expression (16.6%)
	I2: Missing return (83.3%)
Algorithm (15.6%)	G1: Missing the whole "if" statement (68.1%)
	G2: Missing function call (13.6%)
	G3: Wrong function call (18.1%)

Table 4 shows the frequencies of various classes of faults for these subject programs according to the *orthogonal defect classification* (Durães and Madeira, 2006).

4.2. Alternative hypothesis testing methods for TC1, TC2, and TC3 techniques

Six statistical testing methods are studied in this paper, two from each category, shown under "Example Tests" in Table 2. They are:

(TC1) Methods used in CBI (Liblit et al., 2003) and SOBER (Liu et al., 2006), which will simply be referred to as *CBI* and *SOBER*, respectively,
(TC2) *F*-test and Student's *t*-test (which will simply be called *t-test*), and
(TC3) Mann–Whitney test and Wilcoxon signed-rank test (Lowry, 2006), which will be abbreviated as *Mann–Whitney* and *Wilcoxon*, respectively.

The procedures of CBI and SOBER have been briefly discussed in Section 1 and are not repeated here. Details can be found in Liblit et al. (2003, 2005) and Liu et al. (2005, 2006).

Mann–Whitney (Lomax, 2007) is a non-parametric testing method, widely used to compare the medians of two non-normal distributions. In our preliminary experiment in Hu et al. (2008), we employ Mann–Whitney to conduct non-parametric statistical fault localization. Its p-value, which measures the probability that the evaluation bias for all passed executions and those for all failed executions for a given predicate come from the same population, is used to indicate the extent of fault suspiciousness of a program feature, and hence is used to sort the list of predicates (see Eq. (1)).

Given two samples, F-test and t-test are statistical tests commonly used to decide whether the means and dispersions, respectively, of the distributions are equal (Lowry, 2006). Again, their p-values are used to sort the predicate lists in the experiment.

4.3. Effectiveness metric for TC1, TC2, TC3, and TC4 techniques

The metric T-score has been first proposed in Renieris and Reiss (2003) and then used in other studies (Cleve and Zeller, 2005; Liu et al., 2005, 2006; Renieris and Reiss, 2003) to evaluate the effectiveness of predicate-based fault-localization techniques. The idea is that debugging can start from some highly prioritized predicate statements, search the whole space of statements in a breadth-first manner, and then measure the result of effectiveness by the percentage of statements examined before reaching any faulty statement inclusively. Several previous studies, such as Cleve and Zeller (2005) and Renieris and Reiss (2003), have reported the limitations of T-score. One such limitation is that its rationale heavily relies on an assumption in the debugging process. In reality, it is not easy to take for granted the behavior of "an ideal programmer who is able to distinguish defects from non-defects at each location, and can do so at the same cost for each location considered" (Cleve and Zeller, 2005). As such, we adopt another metric in this paper — the P-score (Zhang et al., 2009a) — which makes no such assumption, and is more intrinsic to a given list of predicates generated by a fault-localization technique than T-score.

P-score uses the appearance position of the *most fault-relevant predicate* in the generated predicate list as the effectiveness of that fault-localization technique. It is similar to the concepts of *Expense* (Jones and Harrold, 2005; Zhang et al., 2009b) (which has been popularly used to evaluate statement-level fault-localization techniques) and F-measure (Chen and Merkel, 2008) (which measures the number of test cases to reveal the first failure in a faulty program). P-score is given by

$$\text{P-score} = \frac{\sum_{i=1}^{m} \text{1-based index of } \tilde{P}_i \text{ in } L}{m \times \text{number of predicates in } L} \times 100\%$$

where m is the number of faults in the program, a 1-based index is an index that starts from 1 (rather than 0), and $\tilde{P}_i$ is the *most fault-relevant predicate*, that is, the predicate closest to the position i of the fault in the program (in terms of the number of non-empty non-comment lines). Let us illustrate P-score through an example. Suppose (i) the program has only one fault, (ii) there are 10 predicates in a program, prioritized as $\langle P_2, P_3, \ldots\rangle$, and (iii) the most fault-relevant predicate is P_3 (that is, $\tilde{P} = P_3$). One would examine the first two predicates out of the total of ten before locating the most fault-relevant predicate, and hence P-score $= 2/10 \times 100\% = 20\%$. The smaller the value, the more effective will be the fault-localization technique.

P-score is used in all parts of our experiment to evaluate the effectiveness of TC1, TC2, and TC3, except when comparing the effectiveness of TC3 and TC4 techniques as discussed in Section 4.6. In that Section , we need to compare the effectiveness of both predicate-based techniques and statement-level techniques together. Predicate-based techniques are based on the execution spectra of predicates, and predicates are set on three kinds of statements (namely, branch statements, return statements, and assignment statements) according to the settings of CBI or SOBER. On the other hand, statement-level techniques are based on the execution spectra of all statements. Since the number of statements are greater than that of predicates in most cases, it is not fair to compare them directly. In the comparative experiment discussed in Section 4.6, we collect the execution spectra of predicates on branch statements only, and adapt TC3 (predicate-based fault-localization) techniques to work solely on branch predicates. Further, we collect the execution spectra of branch statements to drive TC4 (statement-level fault-localization) techniques. Thus, the inputs to predicate-based and statement-level techniques are approximately equal in scale. After that, we adjust P-score to search for the "*most fault-relevant branch statement* $\tilde{P}_i$", and use this revised version of P-score to evaluate both predicate-based techniques and statement-level techniques.

4.4. Experimental setup

Recall that there are a total of 132 faulty versions for all the seven Siemens programs and a total of 38 faulty versions for the space program. Two of the 132 faulty Siemens versions (namely, version v27 of program replace and version v9 of program schedule2) come with no failed test cases, as reported in Liu et al. (2005, 2006). These two versions are excluded because all the methods in our experiment require both passed and failed test cases. According to execution statistics on our platform, the faults in 10 out of 38 faulty versions of the space program cannot be revealed by any test case. Since both passed executions and failed executions are needed to conduct statistical fault localization, we exclude these 10 versions from our experiment.[2] They are versions v1, v2, v25, v26, v30, v32, v34, v35, v36, and v38.

Following Liu et al. (2005, 2006), we use the whole test suite as inputs to the testing methods (except when studying the effect of test suite size on the efficacy of TC1, TC2, and TC3 techniques as discussed in Section 4.7 and when conducting the efficiency analysis of TC1, TC2, and TC3 techniques as discussed in Section 4.8). We also use branches and returns (see Section 3.1) as program locations for predicates in all parts of the experiment except when comparing the effectiveness of TC3 and TC4 techniques as described in Section.

We identify faulty statements by comparing each faulty version with the original (supposedly correct) version. If a fault lies in a global definition statement, we mark the directly affected executable statement as faulty. If a statement is omitted, we mark the next executable statement as the faulty statement. Next, we manually mark the most fault-relevant predicate (or branch statement) in each faulty version. For 111 faulty Siemens versions, the position of the most fault-relevant predicate is always no more than 3 lines from a faulty statement. There is no ambiguity in identifying the most fault-relevant predicate and we use all of them in our experiment. For the remaining 19 faulty Siemens versions, the most fault-relevant predicate in each case is hard to be

[2] We use the UNIX tool gcov to collect the statistics of program executions. However, gcov cannot work with crashed program runs, and we therefore exclude from our experiment the test cases that cause a program to crash. This strategy is also used in other studies such as Jones and Harrold (2005) and Zhang et al. (2009b). Owing to different experimental settings (including the platforms and gcc versions), the number of faulty versions excluded from every study can be different. For example, Jones and Harrold (2005) excluded 8 faulty versions from their experiment, while Debroy and Wong (2009) excluded 3 faulty versions.

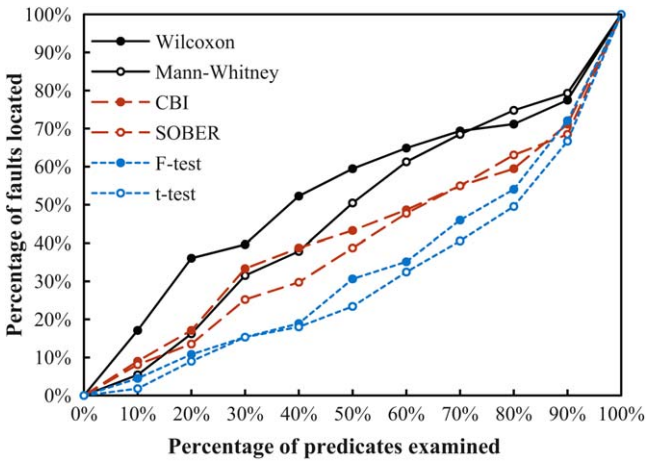

Fig. 1. Overall effectiveness comparison.

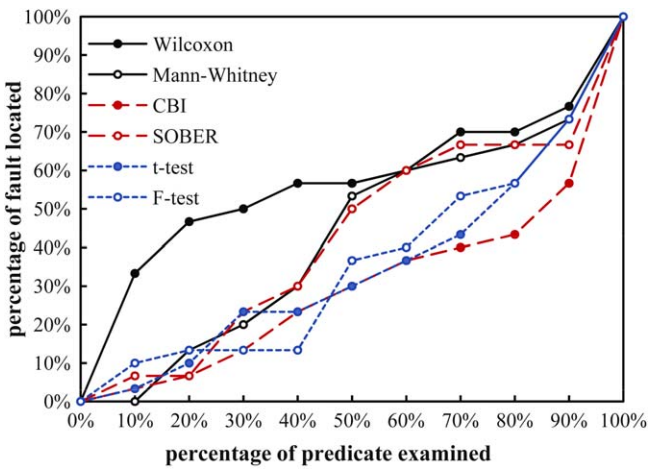

Fig. 2. Individual effectiveness comparison on replace.

uniquely determined. We therefore exclude these 19 versions from the experiment. For the 28 faulty versions of space, the positions of the most fault-relevant branch statements are easy to identify.

We conduct our experiment using a Dell PowerEdge 1950 server running Solaris UNIX with kernel version Generic 120012-14. The tools used to build up our experimental platform include flex++ 2.5.31, bison++ 1.21.9-1, CC 5.8, and gcov 3.4.3. The implementation of the two standard non-parametric tests (namely, Mann–Whitney and Wilcoxon) and the two standard parametric tests (namely, *F*-test and *t*-test) have been downloaded from the ALGLIB website (available at http://www.alglib.net/).

4.5. Effectiveness comparison of TC1, TC2, and TC3 techniques

Fig. 1 shows the results of applying *P*-score to evaluate the effectiveness of the six methods (Wilcoxon, Mann–Whitney, CBI, SOBER, *F*-test, and *t*-test). It depicts the percentage of faulty versions whose most fault-relevant predicates can be located when a certain percentage of predicates in each of the faulty version have been examined.

As an illustration, we consider the behavior of the tests when 10% of the predicates have been examined, and have the following observation:

TC1: CBI and SOBER can only reach the most fault-relevant predicate in 9.01% and 8.11% of the 111 faulty versions, respectively.
TC2: *F*-test and *t*-test can reach the most fault-relevant predicate in 1.80% and 4.50% of the 111 faulty versions, respectively.
TC3: Wilcoxon and Mann–Whitney can reach the most fault-relevant predicate in 17.12% and 5.41% of the 111 faulty versions, respectively.

Similarly, when examining up to 20% of all the predicates in the generated predicate list,

TC1: CBI and SOBER can only reach the most fault-relevant predicate in 17.12% and 13.51% of the 111 faulty versions, respectively.
TC2: *F*-test and *t*-test can reach the most fault-relevant predicate in 9.01% and 10.81% of the 111 faulty versions, respectively.
TC3: Wilcoxon and Mann–Whitney can reach the most fault-relevant predicate in 36.04% and 16.22% of the 111 faulty versions, respectively.

Moreover, in the range of [10–80%], both CBI and SOBER outperform *F*-test and *t*-test. In the range of [10–90%], Wilcoxon always outperforms CBI and SOBER, while the effectiveness of Mann–Whitney is (in the range of [10–40%]) comparable to, or (in the range of [50–90%]) better than CBI and SOBER. From this plot, we observe that Wilcoxon performs better than CBI and SOBER, Mann–Whitney performs comparably to CBI and SOBER, while CBI and SOBER perform better than *F*-test and *t*-test.

Table 5 further summarizes the statistics of the effectiveness of each test. Take Wilcoxon as an example. It has to examine (in the best case) 0.89% and (in the worst case) 100.00% of the all predicates, respectively, in order to locate the most fault-relevant predicate in a faulty version. The median and mean statistics are 39.82% and 46.91%, respectively, and the standard deviation is 35.71%.

Among all six tests, Wilcoxon always scores the best in the rows that correspond to the minimum, median, and mean statistics, but its standard deviation is also the highest. Thus, while Wilcoxon generally performs very well, its performance also varies more widely than other tests. Note also that all the tests may reach the worst case of assigning the lowest rank to the most fault-relevant predicate.

We also include Figs. 2–6 to give readers a better understanding of the effectiveness of every method on each individual subject program. Note that we merge the results of print_tokens and print_tokens2 and show them in one figure (Fig. 6) because they have very similar structures and the number of faulty versions for each of them is too limited to form meaningful individual statistics. For the same reason, we merge the results of schedule and schedule2 and show them in one figure (Fig. 3).

From the plots for replace in Fig. 2, we find that the result of Wilcoxon is always the best among the six, the results of

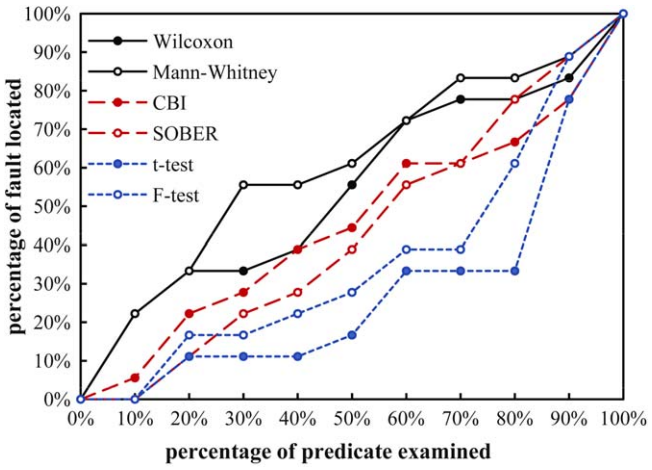

Fig. 3. Individual effectiveness comparison on schedule and schedule2.

Table 5
Statistics of effectiveness of individual tests.

	Wilcoxon (%)	Mann–Whitney (%)	CBI (%)	SOBER (%)	*F*-Test (%)	*t*-Test (%)
Min	0.89	3.45	0.91	3.70	4.44	4.50
Max	100.00	100.00	100.00	100.00	100.00	100.00
Median	39.82	50.00	63.64	63.06	75.86	80.30
Mean	46.91	53.38	58.58	60.64	67.01	70.74
Stdev	35.71	30.56	34.34	32.06	29.29	28.23

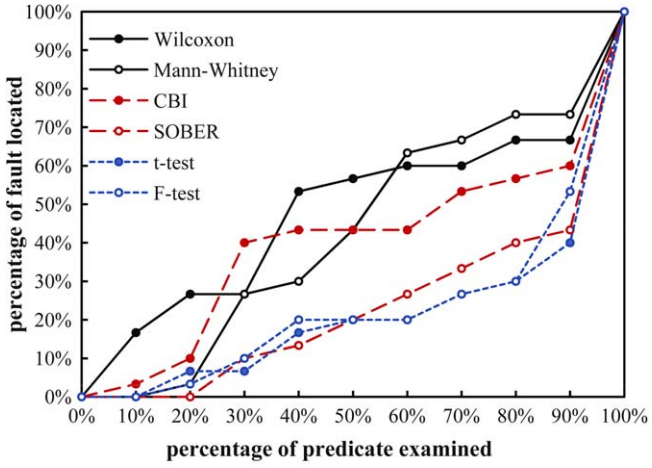

Fig. 4. Individual effectiveness comparison on tcas.

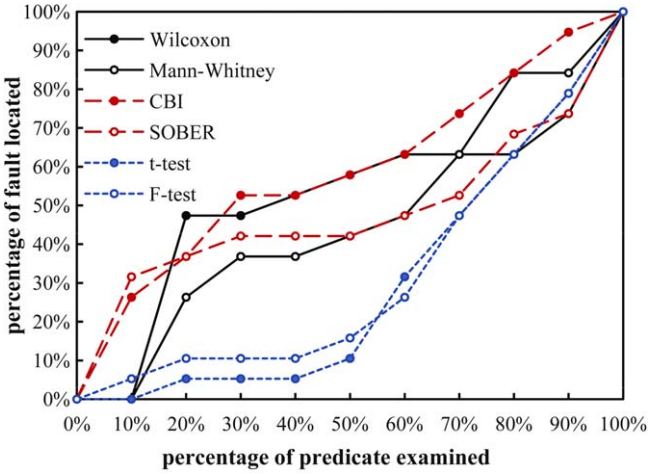

Fig. 5. Individual effectiveness comparison on tot_info.

Mann–Whitney and SOBER are the second best in most regions, while the results of CBI, *t*-test, and *F*-test are comparable to one another. From the plots for schedule and schedule2 in Fig. 3, our observation is that the results of Wilcoxon and Mann–Whitney are better than those of SOBER and CBI, and the results of the latter two are better than those of *t*-test and *F*-test. The plots for tcas in Fig. 4 show little difference. The results of Wilcoxon, Mann–Whitney, and CBI are comparable, the results of SOBER, *t*-test, and *F*-test are also comparable, while results of the former three are better than those of the latter three. In the results for tot_info in Fig. 5, TC1 techniques perform better than TC3 techniques. TC2 techniques are the least effective. In the results for print_tokens and print_tokens2 in Fig. 6, on average, TC1 techniques outperform TC2 techniques, which in turn outperform TC3 techniques. Results on tot_info, print_tokens, and print_tokens2 show that there are still improvements on top of Wilcoxon and Mann–Whitney techniques. CBI works quite well to locate some faults in these subject programs. It will also be interesting to dig out why TC2 techniques are more effective than TC3 on these subject programs. We have found that TC3 techniques can be more effective than TC2 techniques in most cases, and often outperform TC1 techniques. To further find the relative merits on individual versions, we compute the difference in effectiveness between each TC1 technique and each peer technique in TC2 or TC3, and the results are shown in Tables 6 and 7, respectively.

Table 6
Statistics of pairwise comparison (in terms of *P*-score) between Wilcoxon and other techniques on individual programs.

	Wilcoxon – Mann–Whitney	Wilcoxon – CBI	Wilcoxon – SOBER	Wilcoxon – *F*-test	Wilcoxon – *t*-test
<−1%	55	54	58	64	66
−1% to 1%	18	18	20	19	19
>1%	38	39	33	28	26
<−5%	50	53	55	61	62
−5% to 5%	26	21	28	24	27
>5%	35	37	28	26	22
<−10%	44	50	49	54	57
−10% to 10%	36	33	38	34	36
>10%	31	28	24	23	18

Table 7
Statistics of pairwise comparison (in terms of *P*-score) between Mann–Whitney and other techniques (except Wilcoxon) on individual programs.

	Mann–Whitney – SOBER	Mann–Whitney – *F*-test	Mann–Whitney – *t*-test
<−1%	55	53	60
−1% to 1%	23	29	24
>1%	33	29	27
<−5%	53	52	55
−5% to 5%	26	33	30
>5%	32	26	26
<−10%	52	48	51
−10% to 10%	29	39	35
>10%	30	24	25

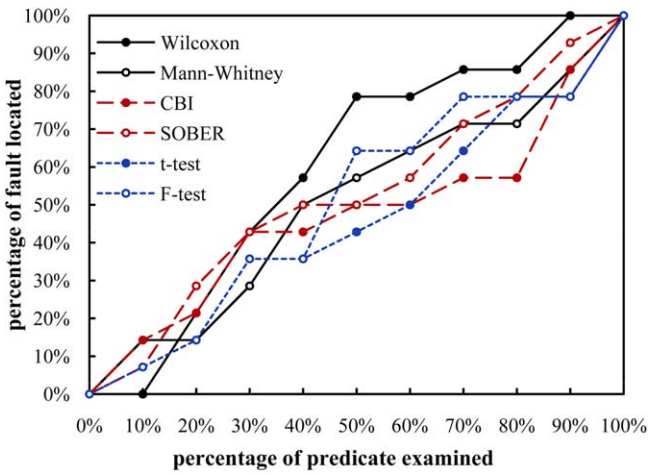

Fig. 6. Individual effectiveness comparison on print_tokens and print_tokens2.

Let us first take the column "Wilcoxon − Mann–Whitney" and the row "<−1%" in Table 6 as an example. It shows that, for 55 of the 110 faulty versions, the code examination effort of using Wilcoxon to locate a fault is less than that of using Mann–Whitney by at least 1%. Similarly, the row ">1%" shows that, for only 38 of the 110 faulty versions, the code examination effort of using Wilcoxon to locate a fault is more than that of Mann–Whitney by at least 1%. The row "−1% to 1%" shows that, for 18 faulty versions, the effectiveness between Wilcoxon and Mann–Whitney cannot be distinguished at a significance level of 1%. By comparing these three rows, we observe that Wilcoxon tends to be more effective than Mann–Whitney. The other columns can be interpreted similarly. We further observe from the two tables that, in general, Wilcoxon is more effective in locating faults than the other five techniques. Similarly, Mann–Whitney is more effective than SOBER, *F*-test, and *t*-test.

4.6. Effectiveness comparison of TC3 and TC4 techniques

We have shown experimentally that TC3 techniques outperform TC1 and TC2 techniques. In this section, we further use a case study to compare the effectiveness of TC3 techniques with TC4 techniques over space.

Fig. 7 shows the histogram for the failing rate distribution of the 28 faulty versions of space. Since most faults have failing rates less than 10%, we use logarithmic coordinates in this figure. We observe that the mode of this histogram in the range of 4–8%.

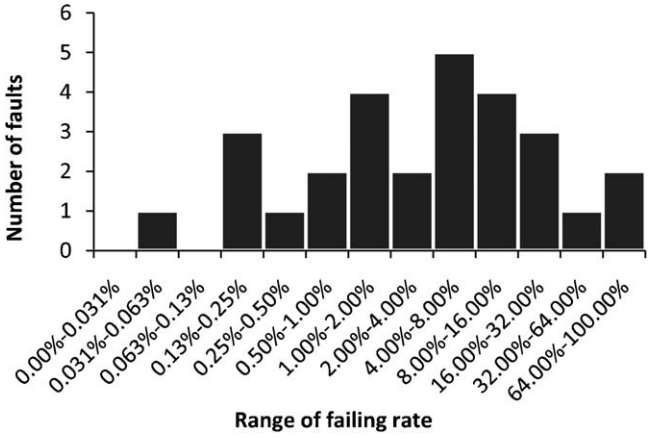

Fig. 7. Histogram for failing rate distribution of the 28 faulty versions of space.

In the experiment, we compare the effectiveness of TC3 techniques with 33 existing statement-level techniques summarized and studied in Naish et al. (in press). We have summarized them in Table 1 for reader's ease of reference. Each technique differs from Tarantula on its ranking formula only. The terms a_{ef} means the number of failed executions that exercise a statement, a_{nf} means the number of failed executions that does not exercise a statement, a_{ep} and a_{np} mean the numbers of passed executions that exercise and do not exercise a statement, respectively.

The experimental results[3] are shown in Fig. 8. Note that we group the 33 techniques into three rows according to their order of appearance in Table 1. The effectiveness of each technique is shown in figure using the box-and-whisker plot. Furthermore, for ease of comparison, we replicate the results of the two TC3 techniques (Wilcoxon and Mann–Whitney) as the two rightmost bars in each group. Let us take the Anderberg technique (the second from the left in the first row) as an example. For each of the 28 faulty versions, Anderberg outputs a ranked list of branch statements. Accordingly, by applying *P*-score on each of the 28 faulty versions, Anderberg produces an individual *P*-score value. The position of the separator dash in the box indicates a median value of 32% using Anderberg is used. It means intuitively that, by following the suggestion made by Anderberg, developers need to examine, on average, 32.0% of the branch statements in order to locate 50% of the most fault-relevant branch statements in the 28 faulty versions. The top of the upper whisker shows the maximum value (62.5%) of the 28 *P*-scores, while the bottom of the lower whisker shows the minimum value (11.1%). Accordingly, they mean the worse case and the best case scenarios when Anderberg is used. The top and bottom of the box correspond to the 75% percentile and the 25% percentile of the 28 *P*-score values. They indicate intuitively that developers need to examine, on average, 37.7% and 23.2% of the branch statements to locate the 7th (25% percentile position) and the 21st (75% percentile position) most fault-relevant branch statements in the 28 faulty versions.

We also zoom into the figure and show them in Fig. 9. From Figs. 8 and 9, we observe that the effectiveness of TC4 techniques are not always good. Some old technique (e.g., Tarantula) and some recently well evaluated techniques (e.g., Jaccard) have outstanding effectiveness. Some techniques having good mathematical supports (e.g., geometric mean) do not yield good results. Interestingly, there are also some techniques (e.g., Scott), which are not extensively mentioned in previous works, but work well on space. Comparing with the overall effectiveness of TC4 techniques, the effectiveness of TC3 techniques are above the average.

Readers may be interested to know the number of faults located with different code examining effort for these techniques. Table 8 gives such a summary. Again, we group the 33 TC4 techniques into three rows to ease our presentation. Take the column with a label "Jaccard" and the row with a label "1%" for example. The cell means that with 1% code examination effort, the technique Jaccard can find the most fault-relevant branch statements in 13 faulty versions of space. Other cells can be interpreted similarly. From this table, we observe that for almost each of the 1%, 2%, 5%, 10%, 20%, and 50% code examination efforts, TC3 techniques can outperform almost every TC4 technique in locating the most fault-relevant branch statements in the faulty versions of space.

[3] The calculation of non-parametric hypothesis testing methods Wilcoxon and Mann–Whintey are conducted using the ALGLIB library (available at http://www.alglib.net/). The wilcoxonsignedranktest and mannwhitneyutest procedures of the ALGLIB library may encounter malformatted data and output NaN (Not a Number) results in a few cases. It may be caused by unknown reasons due to the execution spectra data of the space program. In such cases, we use Tarantula's formula to continue the calculation and replace the NaN values. We choose Tarantula's formula because it is known to be unaffected by the divided-by-zero issue.

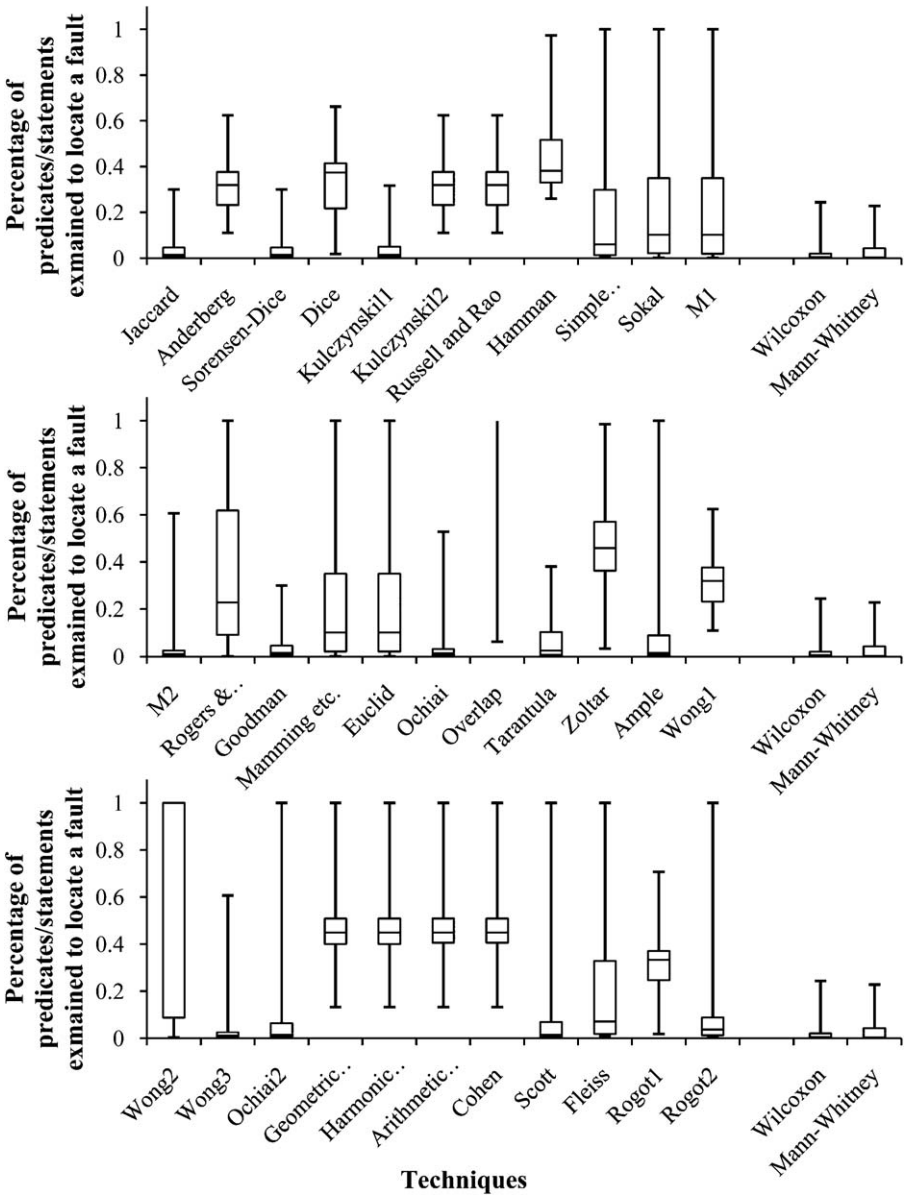

Fig. 8. Effectiveness comparison between non-parametric methods and statement-level techniques.

We further calculate the minimum, maximum, mean, median, and standard deviation of effectiveness for each technique, and show them in turn in the five plots in Fig. 10. Logorithmic coordinates are used in the *y* axes. Further, we use a triangle sign and a square sign to mark the Wilcoxon and Mann–Whintey techniques, respectively. For ease of comparison, we sort the techniques in the first plot in ascending order of their minimum effectiveness measures. For example, it shows that among the 35 techniques, Wilcoxon is the champion in effectiveness in the best case (minimum measurement), while the Mann–Whitney technique is the first runner-up. The other plots are interpreted similarly except that the maximum, mean, median, and standard deviation effectiveness measures are used for sorting. Our observation is that in terms of the minimum, maximum, mean, or median values, Wilcoxon and Mann–Whitney are always the best two among the 35 techniques. However, when looking at the standard deviations, Wilcoxon and Mann–Whintey are the 4th and 5th, respectively. In summary, this figure shows that TC3 techniques require the lowest effort in code examination to locate faults and have relatively low fluctuations in the effectiveness of locating faults. This test also consolidates the observations in previous paragraphs.

4.7. Effect of test suite size on efficacy of TC1, TC2, and TC3 techniques

We have also investigated the effect of different test suite sizes by observing the corresponding change in effectiveness. Fig. 11 shows the results. The test suite size, shown as *x*-axis in the figure, is gradually increased from 50 to 100, 150, 200, 300, 400, 500, and finally to 1000. The *y*-axis stands for the mean percentage of predicates examined to locate the most fault-relevant predicate. The test cases were randomly selected from the test pool.

We observe that, overall, the curves for Wilcoxon and Mann–Whitney show a decreasing trend as test suite size increases. On the other hand, the curves for CBI, SOBER, *F*-test, and *t*-test do not show any decreasing trend with the increase of test suite size. The results show that the use of Wilcoxon or Mann–Whitney in our model is more effective for test suites of larger sizes than for test

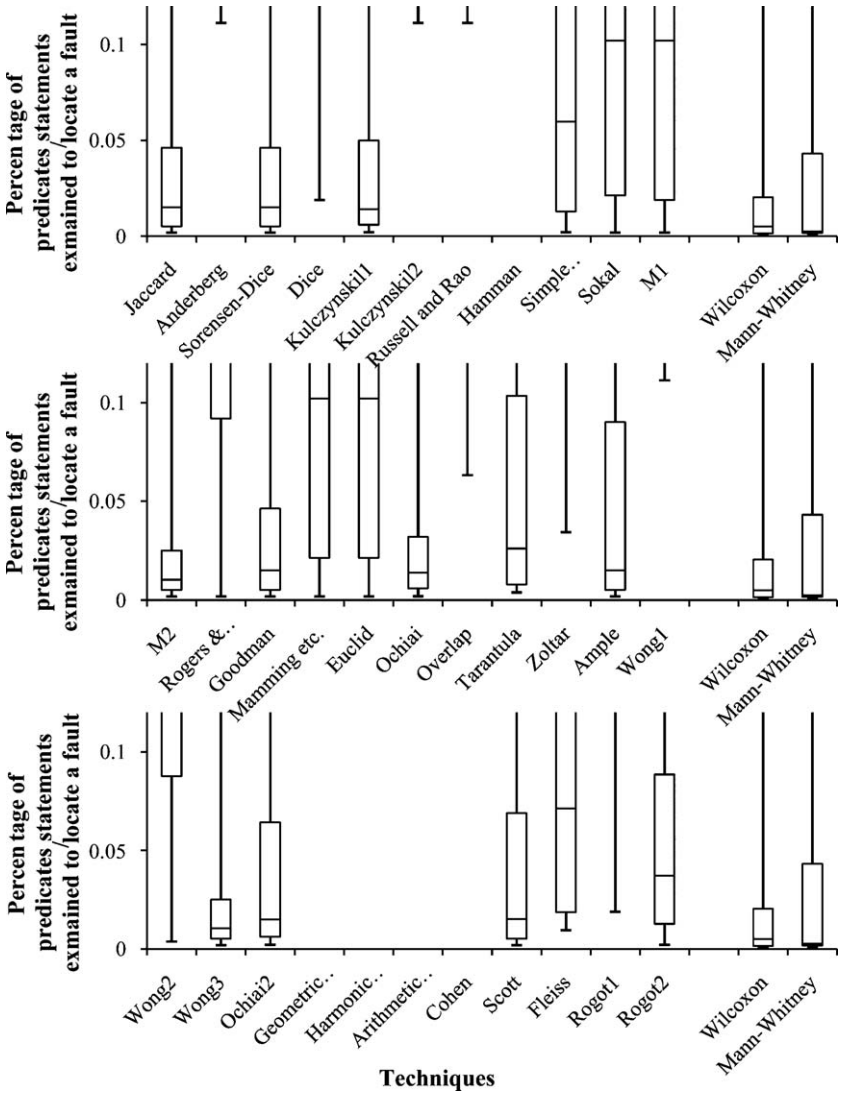

Fig. 9. Effectiveness comparison between non-parametric methods and statement-level techniques (zoomed in).

suites of smaller sizes. Thus, in terms of effectiveness improvement with increase of test suite size, Wilcoxon and Mann–Whitney are more scalable than CBI, SOBER, *F*-test, and *t*-test.

4.8. Efficiency analysis of TC1, TC2, and TC3 techniques

In this section, we report the efficiency of our implementation of the fault-localization techniques. Fig. 12 shows the mean execution time of using these techniques to rank the predicates. The test suite size is chosen as 1000. All the times spent are collected by sequentially executing each technique to rank the predicates in each faulty version. The six categories in Fig. 12 represent, respectively, the results over all the programs, the results on programs print_tokens and print_tokens2, the results on program replace, the results on programs schedule and schedule2, the results on program tcas, and the results on program tot_info. In each category, the six different bars respectively show the mean times taken by each technique on the faulty versions of the corresponding program category. For the print_tokens and print_tokens2 category, for instance, the six bars in the figure represent the mean execution times spent by Wilcoxon (1.002 s), Mann–Whitney (1.192 s), CBI (1.674 s), SOBER (1.657 s), *F*-test (1.218 s), and *t*-test (1.223 s). Note that we group the faulty versions of print_tokens and print_tokens2 into the same category because (i) each of them has too few faulty versions to form meaningful statistics individually, and (ii) these programs have similar structures and logic. For the same reason, we also group the faulty versions of programs schedule and schedule2 into the same category.

From Fig. 12, we observe that the times taken by individual techniques show an increasing trend as the program sizes increases. For example, programs replace, print_tokens, and print_tokens2 are larger in scale than programs schedule, schedule2, tcas, and tot_info, and the mean execution times of each technique in the former three programs are longer than those in the latter four programs. This is understandable because the former three programs have relatively more predicates (Zhang et al., 2010). We also observe that the tests in category TC1 (CBI and SOBER) generally run slower than those in categories TC3 (Wilcoxon and Mann–Whitney) or TC2 (*F*-test and *t*-test), while the tests in TC2 often run slightly faster than those in TC3 (apart from applying Wilcoxon to program replace). The former happens because the selected standard parametric or non-parametric methods are designed by mathematicians and have better performance. The latter happens because the algorithms of these two non-parametric methods are more complex than those of the two parametric methods.

Table 8
Number of faults located with different code examining efforts using TC3 and TC4 techniques.

	Jaccard	Anderberg	Sorensen-Dice	Dice	Kulczynski1	Kulczynski2	Russell and Rao	Hamman	Simple Matching	Sokal	M1	**Wilcoxon**	**Mann-Whitney**
1%	13	0	13	0	13	0	0	0	7	6	7	**16**	**18**
2%	16	0	16	1	15	0	0	0	8	7	7	**21**	**20**
5%	21	0	21	1	21	0	0	0	13	10	10	**23**	**22**
10%	23	0	23	1	23	0	0	0	15	14	14	**24**	**23**
20%	27	5	27	6	27	5	5	0	20	18	18	**26**	**25**
50%	28	26	28	26	28	26	26	20	21	21	21	**28**	**28**
100%	28	28	28	28	28	28	28	28	28	28	28	**28**	**28**

	M2	Rogers & Tanimoto	Goodman	Mamming etc.	Euclid	Ochiai	Overlap	Tarantula	Zoltar	Ample	Wong1	**Wilcoxon**	**Mann-Whitney**
1%	14	2	13	6	6	13	0	9	0	13	0	**16**	**18**
2%	18	3	16	7	7	16	0	14	0	15	0	**21**	**20**
5%	23	3	21	10	10	22	0	17	1	19	0	**23**	**22**
10%	24	9	23	14	14	24	1	21	1	22	0	**24**	**23**
20%	24	13	27	18	18	26	1	26	2	24	5	**26**	**25**
50%	26	20	28	21	21	27	2	28	16	27	26	**28**	**28**
100%	28	28	28	28	28	28	28	28	28	28	28	**28**	**28**

	Wong2	Wong3	Ochiai2	Geometric Mean	Harmonic Mean	Arithmetic Mean	Cohen	Scott	Fleiss	Rogot1	Rogot2	**Wilcoxon**	**Mann-Whitney**
1%	1	14	11	0	0	0	0	13	2	0	7	**16**	**18**
2%	3	19	15	0	0	0	0	15	8	1	8	**21**	**20**
5%	5	23	20	0	0	0	0	20	13	1	16	**23**	**22**
10%	8	24	24	0	0	0	0	21	16	1	21	**24**	**23**
20%	9	24	25	1	1	1	1	23	19	5	23	**26**	**25**
50%	11	26	27	19	19	19	19	23	23	24	25	**28**	**28**
100%	28	28	28	28	28	28	28	28	28	28	28	**28**	**28**

4.9. Answering research questions

In the previous sections, we have discussed the phenomenon that TC3 techniques are observably more effective than TC1 techniques, and TC1 techniques are observably more effective than TC2 techniques. To know whether the differences in effectiveness are statistically significant, we conduct hypothesis testing (using *t*-test) to verify the observations. We set up the following hypothesis:

H_1: Technique *X* and technique *Y* have no significant difference in terms of *P*-score.

Note that only the *p*-values less than 0.05 are shown in Table 9. We leave the cell as "–" if the *p*-values ≥ 0.05. Take the rightmost cell with a value of 0.0068 as an illustration. It means that the *p*-value for the hypothesis is 0.0068, indicating the null hypothesis (H_1) can be rejected at a 5% significance level ($0.0068 < 0.05$). In other words, given such samples, the probability that "the effectiveness of the techniques using *F*-test and that of the techniques using *t*-test come from the same population" is 0.68%. Since we also have an intuitive observation in Sections 4.5 and 4.7 that *F*-test is more effective overall than *t*-test, we can conclude with confidence

Table 9
Hypothesis testing results on H_1.

		X =				
		Wilcoxon	Mann–Whitney	CBI	SOBER	*F*-test
Y =	*t*-test	<0.0001	<0.0001	= 0.0002	<0.0001	= 0.0068
	F-test	<0.0001	= 0.0003	= 0.0074	= 0.0080	
	SOBER	<0.0001	= 0.0362	–		
	CBI	= 0.0025	–			
	Mann–Whitney	= 0.0327				

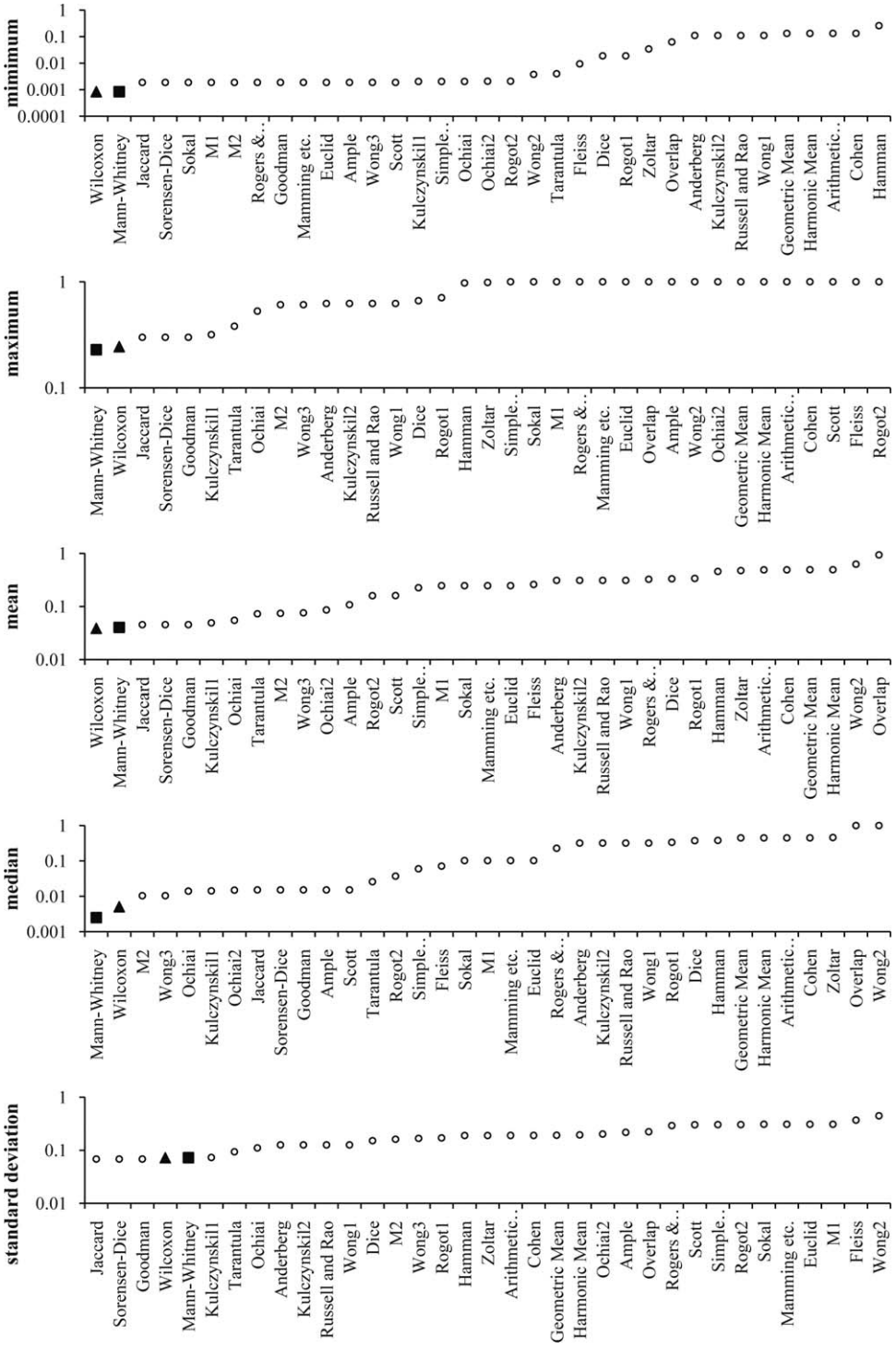

Fig. 10. Minimum, maximum, mean, median, and standard deviation of effectiveness of TC3 and TC4 techniques over the space program.

that *F*-test has statistically significant advantages over *t*-test. Similarly, we compare the other pairs of techniques and summarize the following observations from Table 9:

*R*1: Wilcoxon significantly outperforms Mann–Whitney, Mann–Whitney significantly outperforms SOBER, and SOBER significantly outperforms *F*-test.
*R*2: Wilcoxon significantly outperforms CBI, and CBI significantly outperforms *F*-test,
*R*3: *F*-test significantly outperforms *t*-test.

Based on the above statistical results and the results presented in the previous sections, we can answer the research questions in Section 3.3 thus:

*A*1: Compared with TC1 techniques, TC2 techniques are not statistically more effective (at a 5% significance level).
*A*2: Compared with TC2 techniques, TC3 techniques are statistically more effective (at a 5% significance level).
*A*3: Compared with TC1 techniques, TC3 techniques are often more effective (for 3 cases out of 4, at a 5% significance level).

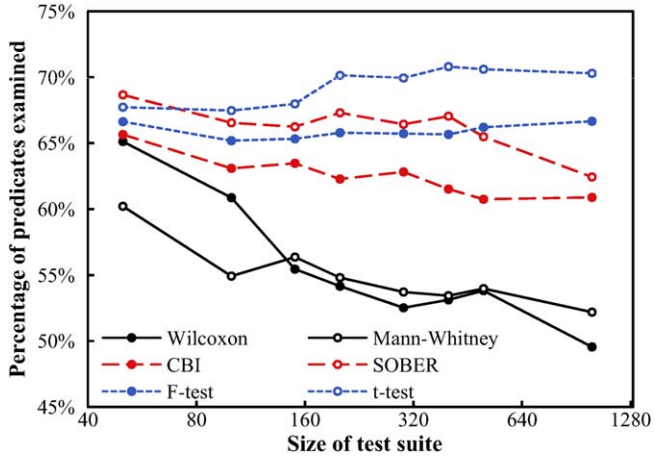

Fig. 11. Effect of test suite size. (The lower the curve is, the better will be the technique.)

We have shown that TC3 techniques outperform TC4 techniques in the minimum, maximum, mean, and median effectiveness measures. To find out whether the advantages are also statistically significant, we validate them using hypothesis testing.

Let X be the Wilcoxon or Mann–Whitney technique and Y be one of the other 33 techniques. We follow the above procedure in Section 4.9 to test hypothesis H_1. It can be interpreted as similar to the answering of research questions $Q1$ to $Q3$.

If we take 5% as the threshold to reject H_1, the null hypothesis H_1 can be rejected at a significance level of 5% when X is Wilcoxon and Y is any of the 33 TC4 techniques except Sørensen-Dice, Kulczynski1, Goodman, Ochiai, and Ochiai2. In other words, given such samples, the probability that "the effectiveness of Wilcoxon and that of the other 28 (= 33 − 5) techniques come from a same population" is less than 5%. Since we also have an intuitive observation in previous sections that Wilcoxon needs, on average, less code examination effort to locate faults on space than TC4 techniques, we conclude that Wilcoxon has statistical advantages over all the TC4 techniques studied (except Sørensen-Dice, Kulczynski1, Goodman, Ochiai, and Ochiai2) at a significance level of 5%. Similarly, Mann–Whitney also has statistical advantages over all the TC4 techniques studied (except Sørensen-Dice, Kulczynski1, M2, Goodman, Ochiai, and Ochiai2) at a significance level of 5%. When we use 10% as the significance level to test H_1, both Wilcoxon and Mann–Whiney has statistically significant advantages over 30 of out of the 33 TC4 techniques studied (except Sørensen-Dice, Kulczynski1, and Goodman). When using 30% as the threshold to test H_1, Wilcoxon has statistically significant advantages over all the TC4 techniques studied. When we use 35% as the significance level to test H_1, Mann–Whintey has statistically significant advantages over all the studied TC4 techniques. Finally, we answer research question $Q4$ as follows:

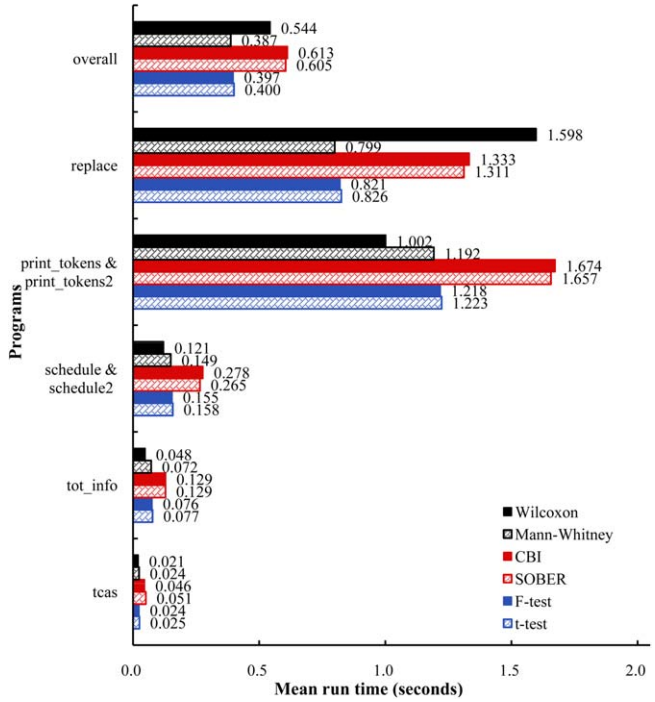

Fig. 12. Time spent by each technique on subject programs.

A4: The effectiveness of TC3 techniques is empirically better than most of the TC4 techniques.

The experimental results presented in previous sections show that Wilcoxon, Mann–Whitney, CBI, and SOBER tend to be more effective as the size of a test suite increases, whereas the overall effectiveness trends for CBI, SOBER, *F*-test, and *t*-test do not seem to increase as obviously when the size of a test suite increases.

To confirm these observations, we apply another hypothesis test:

H_2: For the same technique, there is no significance difference in *P*-score with respect to the use of different test suite sizes.

More specifically, for each curve in Fig. 11, we compute the change in *P*-score between every two adjacent points, compare the series of such changes with a series of zeros, and conduct hypothesis testing (using *t*-test) to validate H_2. The results for Wilcoxon, Mann–Whitney, CBI, SOBER, *F*-test, and *t*-test are 0.05, 0.21, 0.16, 0.16, 0.99, and 0.33, respectively.

If we deem 0.05 as the threshold to reject H_2, the effectiveness of Wilcoxon is confirmed to have changed significantly as the size of a test suite increases. Since we also have an intuitive observation that the effectiveness of Wilcoxon has a discernible increasing trend with the increase of test suite size, we can conclude with confidence that there is a statistically significant increasing trend. On the other hand, we do not find any significant difference for the other five techniques. Together with the results presented in the last section on comparing the effectiveness among techniques, we can answer research question $Q5$.

A5: The effectiveness of TC2 and TC1 techniques do not improve much with increasing test suite size; whereas TC3 techniques, particularly Wilcoxon, improves significantly as the number of available test cases increases.

In a previous section, we have observed that there is discernible difference between the times taken by two techniques to compute the predicate lists. We further set up a hypothesis test to validate this observation:

H_3: There is no significant difference between the times taken to compute the predicate lists by two techniques under study.

The result is shown in Table 10. Note that if a *p*-value is greater than 0.05, we do not show it but leave the cell as "–". Let us take the top left cell with "X = Mann–Whitney" and "Y = CBI" as an example. It means that the *p*-value is less than 0.0001 and H_3 can be rejected at a significance level of 5%. The other cells can be interpreted similarly. Based on Table 10, we can confirm that

Table 10
Hypothesis testing results on H_3.

		X =			
		Mann–Whitney	*F*-test	*t*-test	SOBER
Y =	CBI	<0.0001	<0.0001	<0.0001	–
	SOBER	<0.0001	<0.0001	<0.0001	
	t-test	<0.0001	<0.0001		
	F-test	<0.0001			

there is significant difference between the times taken to compute the predicate lists by every pair of techniques listed in the table (except the SOBER–CBI pair). On the other hand, we find that there is no significant difference between Wilcoxon and any other technique under study. (Since there is no significant difference, the comparisons with Wilcoxon are not listed in the table.)

Although there are significant differences in most pairs of techniques, we would like to study which techniques are more efficient in multiple aspects. For each technique, therefore, we compute the standard statistics to measure both the extreme values and central tendency. The results are shown in Table 11.

Take the first column of the table as example. In the best case scenario, Wilcoxon takes 0.019 s to finish whereas, in the worst case scenario, it takes 1.618 s. For the median and mean scenarios, it takes 0.086 and 0.544 s, respectively. The standard deviation of the time taken to execute Wilcoxon is 0.672. The other columns can be interpreted similarly.

We observe that Mann–Whitney attains the best mean and median results. Wilcoxon is the least efficient among TC2 and TC3 techniques, and is also the most diverse (having the largest standard deviation) among the six techniques. For min, max, mean, and median, TC1 techniques are the least efficient.

Based on the above discussion, we can answer research question *Q*6 as follows:

*A*6: In terms of the time taken to execute a technique, the efficiencies of TC1, TC2, and TC3 are comparable.

4.10. Threats to validity

In this section, we discuss the threats to validity of our experiment.

4.10.1. Internal validity

Internal validity refers to whether a causal relationship between two variables is properly demonstrated in the experiment. The authors of SOBER have released their instrumented faulty versions. We find that instrumentation to some predicates is omitted in their experiment. Since we have no clue on whether a predicate should or should not be included, we follow their specification to include all the predicates and work out their experiment again by ourselves. We have also developed a prototype to automate the experiment to minimize manual errors. To apply CBI and SOBER, we have implemented the techniques according to their published papers. We have used a few sample programs to test the correctness of our implementations. To apply *F*-test, Student's *t*-test, the Mann–Whitney test, and the Wilcoxon signed-rank test in the controlled experiment, we use a public-domain mathematics package ALGLIB rather than our own implementation. Note that a number of research projects have used this package. We have searched the Internet to look for reports on the accuracy problems of the mathematical package, and yet we are not aware of such reporting related to these four tests. We also spot check some results by working out the *p*-values using MATLAB independently to confirm whether the results computed by the package can be reliable. In addition, we use gcov, which is assumed to produce reliable statistics of execution counts.

Table 11
Statistics of time taken to execute each technique (in seconds).

	Wilcoxon	Mann–Whitney	CBI	SOBER	*F*-test	*t*-test
Min	0.019	0.021	0.038	0.022	0.022	0.023
Max	1.618	1.340	1.972	1.916	1.366	1.376
Median	0.086	0.080	0.138	0.138	0.080	0.081
Mean	0.544	0.387	0.613	0.605	0.397	0.400
Stdev	0.672	0.445	0.670	0.662	0.455	0.456

When analyzing the scalability of Wilcoxon, Mann–Whitney, CBI, SOBER, *F*-test, and *t*-test, we use the same program library at the ALGLIB website (see Section 4.4) for implementation, and do not optimize any of them. Such consideration aims to compare their effectiveness fairly. However, different implementation details may affect their run time comparison. Another threat may be due to the choice of predicates we investigate. CBI and SOBER interpret different kinds of statements as predicates. It is not easy to directly compare them with each other. On the other hand, SOBER has reported that scalar-pair predicates only have minor effects on fault localization results. Hence, we follow SOBER and exclude them from our experiment. The inclusion of scalar-pair statements may affect the performance of CBI.

4.10.2. Construct validity

Construct validity refers to whether the experiment actually measures what it intends to measure.

In the experiment, we include CBI and SOBER for comparison. Although there exist other techniques, both CBI and SOBER are representative predicate-based techniques and popularly used to compare with new predicate-level fault localization techniques. We use *P*-score to measure the effectiveness of a technique. This metric is originally adapted from its statement-level counterpart (that is, the number of statements to be examined in order to locate the faulty statement). To fairly evaluate both predicate-based and statement-level techniques and compare their results together, we apply the former on branch statements and the latter on branch predicates so that *P*-score can be used to evaluate them. The use of other metrics such as *T*-score may incur limitations as discussed in a previous section and in Cleve and Zeller (2005). The use of other metrics may also produce different comparison results.

In the experiment, we manually mark the most fault-relevant predicates in the faulty versions. Such manual work may cause threats to construct validity. We exclude those faulty versions in which the most fault-relevant predicate cannot be uniquely determined.

To strike a balance between our resources and the scale of the experiment, the efficacy, scalability, and efficiency comparisons of two TC1, two TC2, and two TC3 techniques are conducted over the Siemens suite of programs, while the efficacy comparisons of two TC3 and 33 TC4 techniques are conducted over the median-sized real-life space program. This also may cause threats to construct validity of the experiment if we compare the results across experiments.

4.10.3. External validity

In our experiment that compares predicate-level techniques, we show the fault-localization results using two non-parametric hypothesis testing methods, two parametric hypothesis testing methods, and two debugging-specific methods. The use of other non-parametric, parametric, or debugging-specific statistical methods may give different comparison results. However, six techniques have been investigated in this paper in multiple dimensions. We believe that it represents a significant effort in the controlled experiment. We have also used 33 statement-level techniques in the experiment. We have verified their formulas carefully.

External validity may also be caused by the subject programs used. The faults in the Siemens programs are seeded manually. They may not truly represent real-life faults. We have used the faulty versions of space to supplement the study of TC3 techniques. They contain real-life faults in a real-life program. The use of other programs may give different results. Moreover, the subject programs are not large in scale and are not concurrent programs. It also poses limitations on the generalization of the results of this paper.

In this paper, we have not evaluated the effectiveness of the examined techniques on multiple-fault programs. In our predicate-based fault-localization framework, we locate only one fault in every fault-localization process. After fixing the located fault, our technique can be rerun to locate the next fault.

4.11. Discussions

We have reported the experiment and analyzed the data to answer the research questions Q1–Q6. In this section, we revisit the findings and discuss their implications.

We find that standard parametric fault-localization techniques cannot outperform debugging-specific techniques. We further observe that, in the experiment, the techniques under category TC1 (namely, SOBER and CBI) uses the means either in their assessment formula or uses the means to represent the probability in the assessment formula. These techniques are parametric in nature. The answer to Q1 provides justification evidence for one to develop debugging-specific parametric techniques.

Standard non-parametric cores outperform parametric cores in terms of effectiveness. The effectiveness of the former is much more positively correlated to the maximum number of test executions for fault localization than that of standard parametric cores or debugging-specific cores. This finding is interesting. It indicates that non-parametric techniques are preferred to parametric counterparts. On the other hand, based on our findings on comparing parametric cores and debugging-specific cores, we conjecture that debugging-specific non-parametric cores may be even more effective than TC1, TC2, and TC3. Our answer to Q6 further shows that non-parametric techniques can be efficiently implemented. Therefore, our study points out a research direction in fault localization, namely, that one may further study debugging-specific non-parametric statistical fault-localization techniques.

Our results show that standard non-parametric predicate-based techniques outperform statement-level techniques over the evaluated subject programs. It means that, compared with statement-level techniques, predicate-based fault-localization techniques can be also useful. This has not been reported in previous studies. On the other hand, since we make compromises on both sides (adapting statement-level techniques to work solely on branch statements and adapting predicate-based techniques to work solely on branch predicates), both of the effectiveness measures on these subject programs may have been modified.

In many previous studies, predicate-level fault-localization techniques are often shown to be less effective than statement-level techniques. In our experiment, we have shown that using a non-parametric core for predicate-level techniques can outperform existing debugging-specific predicate-based or statement-level techniques. An interesting question is whether one may import the concept of non-parametric hypothesis testing to statement-level techniques to enhance the latter.

5. Conclusion

Fault localization is a time-consuming and yet crucial activity in software debugging. Many previous studies contrast the program features of passed executions and failed executions to locate the predicates correlated to faults. However, they overlook the investigation of the statistical distributions of the program features, on which their parametric techniques fully rely. Previous studies have argued and verified empirically that it is problematic to assume specific distributions of program features and use parameters that categorize the distributions in fault-suspiciousness assessments. However, solutions to tackle the problem have not been proposed.

In this paper, we propose a framework to handle statistical predicate-based fault localization by applying standard hypothesis testing techniques proposed by mathematicians. We have conducted a controlled experiment on the Siemens suite and the space program to evaluate the effectiveness of different hypothesis testing methods in our framework, and compare with statement-level fault-localization techniques. We have also experimentally compared the efficacy, scalability, and efficiency of using two standard non-parametric hypothesis testing methods, two standard parametric methods, and two debugging-specific methods on our framework. The experimental results show that standard non-parametric methods outperform standard parametric methods and debugging-specific methods in terms of effectiveness, and are more efficient than debugging-specific methods. Since non-parametric methods are the winners over parametric methods and debugging-specific methods on efficacy, we have also experimentally compared the efficacy of using the two standard non-parametric hypothesis testing methods on our framework with 33 statement-level fault-localization techniques to gauge whether the best predicate-based techniques may outperform statement-level techniques. The experimental results show that standard non-parametric methods also outperform statement-level techniques in terms of effectiveness when given comparably scaled input. Future studies may include the debugging issues of multi-fault programs and concurrent programs, optimizing the size of a test suite for debugging, and developing scientific non-parametric hypothesis testing methods for statement-level fault-localization techniques.

References

Abreu, R., Zoeteweij, P., Golsteijn, R., van Gemund, A.J.C., 2009. A practical evaluation of spectrum-based fault localization. Journal of Systems and Software 82 (11), 1780–1792.

Abreu, R., Zoeteweij, P., van Gemund, A.J.C., 2007. On the accuracy of spectrum-based fault localization. In: Proceedings of the Testing: Academic and Industrial Conference: Practice And Research Techniques (TAICPART-MUTATION 2007) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 89–98.

Agrawal, H., DeMillo, R.A., Spafford, E.H., 1991. An execution backtracking approach to program debugging. IEEE Software 8 (5), 21–26.

Agrawal, H., Horgan, J.R., 1990. Dynamic program slicing. In: Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation (PLDI 1990). ACM SIGPLAN Notices 25 (6), 246–256.

Anderberg, M.R., 1973. Cluster Analysis for Applications. Academic Press, New York, NY Publication: Probability and Mathematical Statistics, New York: Academic Press.

Arumuga Nainar, P., Chen, T., Rosin, J., Liblit, B., 2007. Statistical debugging using compound Boolean predicates. In: Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007) ,. ACM Press, New York, NY, pp. 5–15.

Baudry, B., Fleurey, F., Le Traon, Y., 2006. Improving test suites for efficient fault localization. In: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006) ,. ACM Press, New York, NY, pp. 82–91.

Binkley, D., Gold, N., Harman, M., 2007. An empirical study of static program slice size. ACM Transactions on Software Engineering and Methodology 16 (2), Article No. 8.

Chen, T.Y., Cheung, Y.Y., 1993. Dynamic program dicing. In: Proceedings of the 9th IEEE International Conference on Software Maintenance (ICSM 1993) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 378–385.

Chen, T.Y., Merkel, R.G., 2008. An upper bound on software testing effectiveness. ACM Transactions on Software Engineering and Methodology 17 (3), 1–27.

Chilimbi, T.M., Liblit, B., Mehra, K., Nori, A.V., Vaswani, K., 2009. HOLMES: effective statistical debugging via efficient path profiling. In: Proceedings of the 31st International Conference on Software Engineering (ICSE 2009) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 34–44.

Cleve, H., Zeller, A., 2005. Locating causes of program failures. In: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005) ,. ACM Press, New York, NY, pp. 342–351.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educational and Psychological Measurement 20, 37–46.
Collofello, J.S., Cousins, L., 1987. Towards automatic software fault location through decision-to-decision path analysis. In: Proceedings of the 1987 National Computer Conference , Chicago, IL, pp. 539–544.
Dallmeier, V., Lindig, C., Zeller, A., 2005. Lightweight bug localization with AMPLE. In: Proceedings of the 6th International Symposium on Automated Analysis-Driven Debugging (AADEBUG 2005) ,. ACM Press, New York, NY, pp. 99–104.
da Silva Meyer, A., Garcia, A.A.F., de Souza, A.P., de Souza Jr., C.L., 2004. Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (Zea mays L). Genetics and Molecular Biology 27 (1), 83–91.
Debroy, V., Wong, W.E., 2009. Insights on fault interference for programs with multiple bugs. In: Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE 2009) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 165–174.
Devore, J.L., 2008. Probability and Statistics for Engineering and the Sciences. Thomson/Brooks/Cole, Belmont, CA.
Dice, L.R., 1945. Measures of the amount of ecologic association between species. Ecology 26 (3), 297–302.
Do, H., Elbaum, S.G., Rothermel, G., 2005. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. Empirical Software Engineering 10 (4), 405–435.
Duarte, J.M., dos Santos, J.B., Melo, L.C., 1999. Comparison of similarity coefficients based on RAPD markers in the common bean. Genetics and Molecular Biology 22 (3).
Durães, J.A., Madeira, H.S., 2006. Emulation of software faults: a field data study and a practical approach. IEEE Transactions on Software Engineering 32 (11), 849–867.
Everitt, B.S., 1978. Graphical Techniques for Multivariate Data. North-Holland, New York.
Fleiss, J.L., 1965. Estimating the accuracy of dichotomous judgments. Psychometrika 30 (4), 469–479.
Goodman, L.A., Kruskal, W.H., 1954. Measures of association for cross classification. Journal of the American Statistical Association 49, 732–764.
Griesmayer, A., Staber, S., Bloem, R., 2009. Fault localization using a model checker. Software Testing, Verification and Reliability. doi:10.1002/stvr.421.
Gupta, N., He, H., Zhang, X., Gupta, R., 2005. Locating faulty code using failure-inducing chops. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005) ,. ACM Press, New York, NY, pp. 263–272.
Hamming, R.W., 1950. Error detecting and error correcting codes. Bell System Technical Journal 29 (1), 147–160.
Hao, D., Xie, T., Zhang, L., Wang, X., Sun, J., Mei, H., 2010. Test input reduction for result inspection to facilitate fault localization. Automated Software Engineering 17 (1), 5–31.
Hao, D., Zhang, L., Mei, H., Sun, J., 2006. Towards interactive fault localization using test information. In: Proceedings of the 13th Asia-Pacific Software Engineering Conference (APSEC 2006) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 277–284.
Harrold, M.J., Rothermel, G., Sayre, K., Wu, R., Yi, L., 2000. An empirical investigation of the relationship between spectra differences and regression faults. Software Testing, Verification and Reliability 10 (3), 171–194.
Horwitz, S., Reps, T., Binkley, D., 1990. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems 12 (1), 26–60.
Hu, P., Zhang, Z., Chan, W.K., Tse, T.H., 2008. Fault localization with non-parametric program behavior model. In: Proceedings of the 8th International Conference on Quality Software (QSIC 2008) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 385–395.
Jaccard, P., 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin del la Socit Vaudoise des Sciences Naturelles 37, 547–579.
Jiang, B., Zhang, Z., Tse, T.H., Chen, T.Y., 2009. How well do test case prioritization techniques support statistical fault localization. In: Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC 2009), vol. 1 ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 99–106.
Jones, J.A., Harrold, M.J., 2005. Empirical evaluation of the Tarantula automatic fault-localization technique. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005) ,. ACM Press, New York, NY, pp. 273–282.
Jones, J.A., Harrold, M.J., Bowring, J.F., 2007. Debugging in parallel. In: Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007) ,. ACM Press, New York, NY, pp. 16–26.
Jones, J.A., Harrold, M.J., Stasko, J., 2002. Visualization of test information to assist fault localization. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002) ,. ACM Press, New York, NY, pp. 467–477.
Korel, B., 1988. PELAS: Program error-locating assistant system. IEEE Transactions on Software Engineering 14 (9), 1253–1260.
Korel, B., Laski, J., 1988. Dynamic program slicing. Information Processing Letters 29 (3), 155–163.
Korel, B., Laski, J., 1988. STAD: a system for testing and debugging: user perspective. In: Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis ,. IEEE Computer Society Press, Washington, DC, pp. 13–20.
Krause, E.F., 1973. Taxicab geometry. Mathematics Teacher 66 (8), 695–706.
Lau, M.F., Yu, Y.T., 2005. An extended fault class hierarchy for specification-based testing. ACM Transactions on Software Engineering and Methodology 14 (3), 247–276.
Liblit, B., Aiken, A., Zheng, A.X., Jordan, M.I., 2003. Bug isolation via remote program sampling. In: Proceedings of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2003), ACM SIGPLAN Notices 38 (5), 141–154.
Liblit, B., Naik, M., Zheng, A.X., Aiken, A., Jordan, M.I., 2005. Scalable statistical bug isolation. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005), ACM SIGPLAN Notices 40 (6), 15–26.
Liu, C., Fei, L., Yan, X., Midkiff, S.P., Han, J., 2006. Statistical debugging: a hypothesis testing-based approach. IEEE Transactions on Software Engineering 32 (10), 831–848.
Liu, C., Yan, X., Fei, L., Han, J., Midkiff, S.P., 2005. SOBER: statistical model-based bug localization. In: Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC 2005/FSE-13), ACM SIGSOFT Software Engineering Notes 30 (5), 286–295.
Lomax, R.G., 2007. Statistical Concepts: a Second Course. Lawrence Erlbaum Associates, Mahwah, NJ.
Lourenço, F., Lobo, V., Bac¸ão, F., 2004. Binary-based similarity measures for categorical data and their application in self-organizing maps.
Lowry, R., 2006. Concepts and Applications of Inferential Statistics. Vassar College, Poughkeepsie, NY, Available at http://faculty.vassar.edu/lowry/webtext.html.
Mann, H.B., Whitney, D.R., 1947. On a test of whether one of two random variables is stochastically larger than the other. Annals of Mathematical Statistics 18 (1), 50–60.
Masri, W., 2009. Fault localization based on information flow coverage. Software Testing, Verification and Reliability. doi:10.1002/stvr.409.
Maxwell, A.E., Pilliner, A.E., 1968. Deriving coefficients of reliability and agreement for ratings. The British Journal of Mathematical and Statistical Psychology 21 (1), 105–116.
Naish, L., Lee, H.J., Ramamohanarao, K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology, in press.
Ochiai, A., 1957. Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions. Bulletin of the Japanese Society for Fish Science 22, 526–530.
Ottenstein, K.J., Ottenstein, L.M., 1984. The program dependence graph in a software development environment. In: Proceedings of the ACM Symposium on Practical Software Development Environments ,. ACM Press, New York, NY, pp. 177–184.
Renieris, M., Reiss, S.P., 2003. Fault localization with nearest neighbor queries. In: Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 30–39.
Rogers, D.J., Tanimoto, T.T., 1960. A computer program for classifying plants. Science 132 (3434), 1115–1118.
Rogot, E., Goldberg, I.D., 1966. A proposed index for measuring agreement in test–retest studies. Journal of Chronic Diseases 19 (9), 991–1006.
Russel, P.F., Rao, T.R., 1940. On habitat and association of species of Anopheline larvae in south-eastern Madras. Journal of Malarial Institute of India 3, 153–178.
González-Sánchez, A., 2007. Automatic Error Detection Techniques Based on Dynamic Invariants. Master's Thesis, Department of Software Technology, Delft University of Technology, Delft, The Netherlands.
Scott, W.A., 1955. Reliability of content analysis: the case of nominal scale coding. Public Opinion Quarterly 19 (3), 321–325.
Tip, F., 1995. A survey of program slicing techniques. Journal of Programming Languages 3 (3), 121–189.
Vessey, I., 1986. Expertise in debugging computer programs: an analysis of the content of verbal protocols. IEEE Transactions on Systems, Man, and Cybernetics 16 (5), 621–637.
Weiser, M., 1984. Program slicing. IEEE Transactions on Software Engineering SE-10 (4), 352–357.
Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biometrics Bulletin 1 (6), 80–83.
Wong, W.E., Debroy, V., Choi, B., 2010. A family of code coverage-based heuristics for effective fault localization. Journal of Systems and Software 83 (2), 188–208.
Wong, W.E., Qi, Y., 2009. BP neural network-based effective fault localization. International Journal of Software Engineering and Knowledge Engineering 19 (4), 573–597.
Wong, W.E., Qi, Y., Zhao, L., Cai, K.-Y., 2007. Effective fault localization using code coverage. In: Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 1 ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 449–456.
Wong, W.E., Wei, T., Qi, Y., Zhao, L., 2008. A crosstab-based statistical method for effective fault localization. In: Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST 2008) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 42–51.
Yu, Y., Jones, J.A., Harrold, M.J., 2008. An empirical study of the effects of test-suite reduction on fault localization. In: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008) ,. ACM Press, New York, NY, pp. 201–210.
Zeller, A., 2002. Isolating cause–effect chains from computer programs. In: Proceedings of the 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT 2002/FSE-10), ACM SIGSOFT Software Engineering Notes 27 (6), 1–10.

Zeller, A., Hildebrandt, R., 2002. Simplifying and isolating failure-inducing input. IEEE Transactions on Software Engineering 28 (2), 183–200.
Zhang, X., Gupta, N., Gupta, R., 2006. Locating faults through automated predicate switching. In: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006) ,. ACM Press, New York, NY, pp. 272–281.
Zhang, Z., Chan, W.K., Tse, T.H., Hu, P., Wang, X., 2009a. Is non-parametric hypothesis testing model robust for statistical fault localization? Information and Software Technology 51 (11), 1573–1585.
Zhang, Z., Chan, W.K., Tse, T.H., Jiang, B., Wang, X., 2009b. Capturing propagation of infected program states. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC 2009/FSE-17) ,. ACM Press, New York, NY, pp. 43–52.
Zhang, Z., Jiang, B., Chan, W.K., Tse, T.H., 2008. Debugging through evaluation sequences: a controlled experimental study. In: Proceedings of the 32nd Annual International Computer Software and Applications Conference (COMPSAC 2008) ,. IEEE Computer Society Press, Los Alamitos, CA, pp. 128–135.
Zhang, Z., Jiang, B., Chan, W.K., Tse, T.H., Wang, X., 2010. Fault localization through evaluation sequences. Journal of Systems and Software 83 (2), 174–187.

Zhenyu Zhang is an assistant research professor at the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences. He received a BEng degree and an M.Sc. degree from Tsinghua University, and a Ph.D. degree from The University of Hong Kong. Dr. Zhang's research interests include debugging and testing of software engineering artifacts. Dr. Zhang published widely in international journals and conferences, including two articles that won the best paper awards from COMPSAC 2008 and COMPSAC 2009, as well as articles published in FSE, ICSE, ASE, JSS, IST, and other venues. He is a program chair of IWPDC 2010 and serves on the program committees of many international conferences.

W.K. Chan is an assistant professor at City University of Hong Kong. He received his Ph.D. degree from The University of Hong Kong. His main research interest is in solving software engineering issues in program testing and analysis, and service composition. He is on the editorial board of *Journal of Systems and Software*.

T.H. TSe is a professor in computer science at The University of Hong Kong. He received his Ph.D. from the London School of Economics and was twice a visiting fellow at the University of Oxford. His current research interest is in program testing, debugging, and analysis. He is the steering committee chair of QSIC and an editorial board member of *Software Testing, Verification and Reliability*; *Journal of Systems and Software; Software: Practice and Experience*; and *Journal of Universal Computer Science*. He is a fellow of the British Computer Society, a fellow of the Institute for the Management of Information Systems, a fellow of the Institute of Mathematics and its Applications, and a fellow of the Hong Kong Institution of Engineers. He was decorated with an MBE by The Queen of the United Kingdom.

Yuen Tak Yu is an associate professor at Department of Computer Science of City University of Hong Kong. He received his B.Sc. degree with first class honors from The University of Hong Kong and Ph.D. degree from the University of Melbourne, Australia. His research publications have appeared in many scholarly journals, including *ACM Transactions on Software Engineering and Methodology* and *IEEE Transactions on Software Engineering*. His research interests include software testing, software quality, e-commerce and computer in education. He is a member of the editorial board of *International Journal of Web Engineering and Technology*. He was a program chair of the 2nd Asia-Pacific Conference on Quality Software (APAQS 2001) and has served on the program committees of many international conferences and workshops.

Peifeng Hu received his M.Sc. degree from Tsinghua University and his Ph.D. degree from the University of Hong Kong. He has joined China Merchant (Hong Kong) Bank as a financial analyst since 2007.