IET Software

Research Article

IET Journals
The Institution of Engineering and Technology

How test suites impact fault localisation starting from the size

ISSN 1751-8806
Received on 5th February 2017
Revised 16th November 2017
Accepted on 27th January 2018
E-First on 5th March 2018
doi: 10.1049/iet-sen.2017.0026
www.ietdl.org

Yan Lei[1,2,3] ✉, Chengnian Sun[4], Xiaoguang Mao[5], Zhendong Su[4]

[1]*Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, Chongqing 400044, People's Republic of China*

[2]*School of Software Engineering, Chongqing University, Chongqing 400044, People's Republic of China*

[3]*Information Management Center, Logistical Engineering University, Chongqing 401331, People's Republic of China*

[4]*Department of Computer Science, University of California, Davis, CA 95616-8562, USA*

[5]*College of Computer, National University of Defense Technology, Changsha 410073, People's Republic of China*

✉ *E-mail: yanlei.cs@outlook.com*

Abstract: Although a test suite is indispensable for conducting effective fault localisation, not much work has been done to study how the test suite impacts fault localisation. This study presents a systematic study for a deeper understanding of their relation. Specifically, the authors' study reveals an interesting fact that there is *no strong correlation between localisation effectiveness and the size of the test suite*. Furthermore, they show that, in a test suite, (i) the passing test cases that do not execute the faulty statements and the failing test cases have a *positive* effect on the fault localisation effectiveness, while (ii) the passing test cases that exercise the faulty statements have a *negative* impact on localisation performance. Their result is drawn from a large-scale empirical analysis on the localisation effectiveness with respect to randomly sampled test suites. This study presents the details of the study and their follow-up investigation on the findings. Their work provides a new perspective on fault localisation and suggests fresh directions of research on an extensively studied topic.

1 Introduction

Debugging is a painstaking task in software development and maintenance. In order to reduce the cost of debugging, much research has been devoted to developing *automated fault localisation* techniques such as [1–11]. Generally, these automated debugging techniques analyse dynamic runtime information to identify suspicious localisations (e.g. statements, branches and def–use pairs) which are potentially responsible for a failure, thus to help developers debug. Specifically, given a buggy program $\mathscr{P}$ and a test suite $\mathscr{T}$, the bug is localised in the following way: $\mathscr{P}$ is first instrumented and run against $\mathscr{T}$; then a set of dynamic information is collected and analysed; finally, all the statements (or other program elements) are ranked based on their correlation with faulty program executions.

The test suite $\mathscr{T}$ is an indispensable component to initiate the fault localisation process, as it is the driver to execute $\mathscr{P}$ so that runtime information can be collected and analysed. Most of the proposed fault localisation techniques assume a test suite, but it is unclear whether the test suite is adequate. We still lack a deep understanding of the criteria for a test suite to be adequate. Does the performance of fault localisation increase with the size of $\mathscr{T}$? How do different parts of $\mathscr{T}$ interact with fault localisation? These questions have not been explicitly answered yet in the literature.

Investigating the relation between test suite and fault localisation is necessary and worthwhile. This is because not only the knowledge and insights are interesting in their own right, but perhaps more importantly, they may be leveraged to further improve fault localisation. A considerable number of techniques have been proposed to enhance the fault localisation models, but a few tries to improve localisation accuracy by optimising the test suite. With deeper understandings in this direction, we can advance studies on test suites for fault localization, specifically test case generation and reduction.

In this paper, we present a systematic study on the impact of test suites on fault localisation, trying to study the structure of a test suite to pinpoint what parts of a test suite correlate with fault localisation. Our extensive empirical study is performed on a standard benchmark (`Siemens` programs) and seven popular large real-world programs (`space`, `flex`, `grep`, `sed`, `libtiff`, `python` and `gzip`) with their respective test suites. Considering its popularity in the fault localisation community, our study adopts spectrum-based fault localisation (`SFL`) [12] to perform fault localisation on the selected subject programs. Our study is based on statistical analysis. We choose the basic attribute of a test suite, which is the size of a test suite, to initiate our study for identifying the relative makeup of a test suite that matters. Specifically, for each buggy program $\mathscr{P}$ and its test suite $\mathscr{T}$, we use random sampling to generate a set of subsets of $\mathscr{T}$; next, apply `SFL` on $\mathscr{P}$ with $\mathscr{T}$ and each of its sampled subsets; finally we process and analyse the accuracy data of all the applications of `SFL` using statistical methods.

We find that the fault localisation effectiveness fluctuates drastically across the randomly sampled subsets of $\mathscr{T}$, and on average 68.63% of random samples lead to improved results over the original test suites. This rather surprising finding shows that there is no strong correlation between localisation effectiveness and the size of a test suite, i.e. enlarging $\mathscr{T}$ does not necessarily imply increased fault localisation accuracy.

We analyse the cause of this interesting result and clearly identify the *positive* or *negative* impact of different parts of a test suite on fault localisation [*positive* means beneficial for fault localisation, while *negative* means harmful for localisation effectiveness.]. More concretely, in a test suite, two parts (i.e. the passing test cases that do not execute the faulty statement and the failing test cases) have a *positive* impact on the fault localisation effectiveness, while those passing test cases that exercise the faulty statement have a *negative* effect on localisation performance. We further find that `SFL` can always benefit from the failing test cases, and thus the observed fluctuations across the samples are mainly due to passing test cases: Passing test cases cannot guarantee that their executions do not exercise the faulty statement s_f. If they execute s_f, then these test cases will reduce the suspiciousness of s_f, consequently affecting the final ranking of s_f.

© The Institution of Engineering and Technology 2018

$$
\begin{array}{c}
\\
N\ test\ cases
\end{array}
\begin{array}{c}
\begin{array}{cccc} \multicolumn{4}{c}{M\ statements} \end{array} \\
\left[\begin{array}{cccc|c}
x_{11} & x_{12} & \dots & x_{1M} & r_1 \\
x_{21} & x_{22} & \dots & x_{2M} & r_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
x_{N1} & x_{N2} & \dots & x_{NM} & r_N
\end{array}\right] \\
\begin{array}{cccc} s_1 & s_2 & \dots & s_M \end{array}
\end{array}
\begin{array}{c}
failing/ \\ passing \\ \\
\begin{array}{c} t_1 \\ t_2 \\ \vdots \\ t_N \end{array}
\end{array}
$$

Fig. 1 *Input to software fault localisation*

Table 1 Maximal formulas of SFL

	Name	Formula
ER5	Wong1	$a_{11}(s_j)$
	Russel & Rao	$\frac{a_{11}(s_j)}{a_{11}(s_j)+a_{01}(s_j)+a_{10}(s_j)+a_{00}(s_j)}$
	Binary	$\begin{cases} 0, & \text{if } a_{01}(s_j) > 0 \\ 1, & \text{if } a_{01}(s_j) = 0 \end{cases}$
ER1′	Naish1	$\begin{cases} -1, & \text{if } a_{01}(s_j) > 0 \\ a_{00}(s_j), & \text{if } a_{01}(s_j) = 0 \end{cases}$
	Naish2	$a_{11}(s_j) - \frac{a_{10}(s_j)}{a_{10}(s_j)+a_{00}(s_j)+1}$
	GP13	$a_{11}(s_j)\left(1+\frac{1}{2a_{10}(s_j)+a_{11}(s_j)}\right)$
	GP02	$2(\, a_{11}(s_j)+\sqrt{a_{00}(s_j)}\,)+\sqrt{a_{10}(s_j)}$
	GP03	$\sqrt{\lvert a_{11}(s_j)^2-\sqrt{a_{10}(s_j)}\rvert}$
	GP19	$a_{11}(s_j)\sqrt{\lvert a_{11}(s_j)-a_{01}(s_j)+a_{00}(s_j)-a_{10}(s_j)\rvert}$

Based on this analysis result, we define a new metric *Passing Test Discrimination* (PTD), to compare two test suites by identifying whose passing test cases help more in increasing the suspiciousness of the faulty statement. In general, it is the ratio of the number of passing test cases *not* executing the faulty statement over the number of all passing test cases.

Given a test suite $\mathcal{T}$, we can leverage PTD as a criterion to optimise $\mathcal{T}$ so as to increase its PTD and consequently improve the fault localisation performance. To demonstrate its feasibility, we propose a simple heuristic to improve PTD. It is based on the observation that if a passing test case t executes most of the statements in S, where S is the set of statements executed by all the failing test cases, then t tends to execute the faulty statement with a high likelihood. If the similarity between those statements executed by t and S exceeds a threshold, we remove t from $\mathcal{T}$ and use the remaining test cases for fault localisation. Our experimental results demonstrate that in general, the optimised test suite performs better than the original test suite. And this application of PTD further confirms our theoretical analysis.

Paper organisation: The remaining of this paper is organised as follows. Section 2 introduces necessary background information. Section 3 presents our study and theoretical analysis. Section 4 demonstrates a simple application of using PTD. Section 6 surveys related work and Section 7 concludes.

2 Background on SFL

Given a program $\mathcal{P}$, let $\mathcal{S} = \{s_1, s_2, \dots, s_M\}$ denote its statements and $\mathcal{T} = \{t_1, t_2, \dots, t_N\}$ be the test suite. Fig. 1 shows the input to SFL, an $N \times (M+1)$ matrix. An element x_{ij} is 1 if the statement s_j is covered by the execution of test case t_i, 0 otherwise. The result vector $\boldsymbol{r}$ at the rightmost column of the matrix represents the test results. The element r_i is 1 if t_i is failing, 0 otherwise.

SFL normally evaluates the suspiciousness of a statement by utilising the similarity between the statement vector and result vector of the matrix in Fig. 1. SFL defines four statistical variables for each statement to facilitate the similarity computation, as defined below:

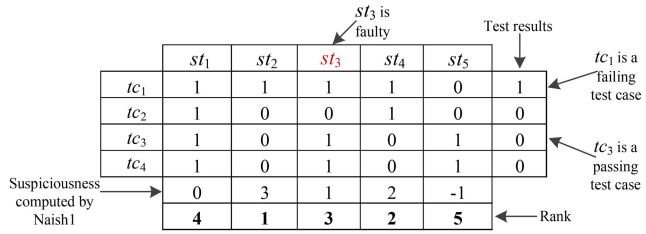

Fig. 2 *Example illustrating SFL*

$$
\begin{aligned}
a_{00}(s_j) &= |\{i \mid x_{ij} = 0 \wedge r_i = 0\}| \\
a_{01}(s_j) &= |\{i \mid x_{ij} = 0 \wedge r_i = 1\}| \\
a_{10}(s_j) &= |\{i \mid x_{ij} = 1 \wedge r_i = 0\}| \\
a_{11}(s_j) &= |\{i \mid x_{ij} = 1 \wedge r_i = 1\}|
\end{aligned} \tag{1}
$$

$a_{00}(s_j)$ and $a_{01}(s_j)$ represent the number of test cases that do *not* execute the statement s_j and return the *passing* and *failing* test results, respectively; $a_{10}(s_j)$ and $a_{11}(s_j)$ stand for the number of test cases that *execute* s_j, and return the *passing* and *failing* testing results, respectively.

With the four variables, a large number of formulas have been proposed to evaluate the suspiciousness of a statement being faulty and they output a ranking list of all statements in descending order of their suspiciousness. Xie *et al.* [6, 13] recently have theoretically proved that 5 out of 30 human-discovered SFL suspiciousness evaluation formulas and 4 out of 30 genetic programming-evolved formulas are the most effective – referred to as the *maximal formulas* – in the single-fault scenario.

Table 1 lists the nine maximal formulas with their definitions: Wong1 [14], Russel & Rao [4], Binary [4], Naish1 [4], Naish2 [4], GP13 [13], GP02 [13], GP03 [13] and GP19 [13]. Moreover, among the nine maximal formulas, three formulas are equivalent and constitute a group ER5, and the other three equivalent formulas compose a group ER1′.

Fig. 2 illustrates a simple example using Naish1 to show how SFL works. As shown in Fig. 2, the example, with a faulty statement st_3, contains five statements $\{st_1, st_2, st_3, st_4, st_5\}$ and four test cases $\{tc_1, tc_2, tc_3, tc_4\}$. The cells below each statement indicates whether the statement was covered by the execution of a test case or not and the rightmost cells represent whether the execution of a test case is failing or not. The detailed description of the cells can refer to the input matrix of SFL presented in Fig. 1. Take statement st_1 as an example. We can obtain the values of its four variables defined in (1), i.e. $a_{00}(st_1) = 0$, $a_{01}(st_1) = 0$, $a_{10}(st_1) = 3$ and $a_{11}(st_1) = 1$. Thus, Naish1 assigns 0 to st_1 as its suspiciousness value. Based on the statement coverage and test results of four test cases, Naish1 computes the suspiciousness of each statement and outputs a ranking list of all statements in descending order of suspiciousness: $\{st_2, st_4, st_3, st_1, st_5\}$. The faulty statement st_3 is ranked 3rd.

3 Study design and results

This section presents the details of our extensive empirical study on the impact of test suites on fault localisation. The general idea is that we generate a large number of test suites that are randomly sampled from the original test suite $\mathcal{T}$, and then measure and compare localisation effectiveness by using $\mathcal{T}$ and the sampled subsets of $\mathcal{T}$. If the size of a test suite is correlated with localisation effectiveness, the distribution of localisation effectiveness over the sizes of test suites should be consistent with the sizes, namely, there should be no drastic fluctuation which is inconsistent with the sizes. However, if we obtain apparent fluctuation among the samples and furthermore, many samples perform equally to or better than $\mathcal{T}$, it possibly indicates no strong correlation between the size of a test suite and localisation effectiveness.

© The Institution of Engineering and Technology 2018

Table 2 Summary of subject programs

Program	LoC	Test	Type	Description	Version
print_tokens (2 ver.)	570/726	316/393	seeded	lexical analyser	print_tokens: v1, v2, v3, v5, v7 print_tokens2: v1, v2, v3, v4, v5, v6, v7, v8, v9, v10
replace	564	395	seeded	pattern recognition	v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v13, v14, v15, v16, v17, v18, v19, v20, v21, v22, v23, v24, v25, v26, v27, v28, v29, v30, v31
schedule (2 ver.)	374/412	228/235	seeded	priority scheduler	schedule: v1, v2, v3, v4, v5, v6, v7, v8, v9 schedule2: v1, v2, v3, v4, v5, v6, v7, v8, v10
tcas	173	83	seeded	altitude separation	v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v15, v16, v17, v18, v19, v20, v21, v22, v23, v24, v25, v26, v27, v28, v29, v30, v31, v32, v33, v34, v35, v37, v39, v40, v41
tot_info	565	194	seeded	information measure	v1, v2, v3, v4, v5, v7, v8, v9, v11, v12, v13, v14, v15, v16, v17, v18, v20, v22, v23
space	6199	4333	real	ADL interpreter	v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18, v19, v20, v21, v22, v23, v24, v25, v26, v28, v29, v30, v31, v33, v35, v36, v37, v38
flex	10,459	567	seeded	lexical analyser	v2: F_HD_1, F_HD_2, F_HD_4, F_HD_5, F_HD_6, F_HD_7, F_HD_8, F_AA_1, F_AA_2, F_AA_3, F_AA_4, F_AA_5, F_AA_6, F_JR_1, F_JR_2, F_JR_3, F_JR_4, F_JR_5, F_JR_6, F_JR_7; v3: F_HD_1, F_HD_2, F_HD_4, F_HD_5, F_HD_6, F_AA_1, F_AA_4, F_AA_5, F_JR_1, F_JR_2, F_JR_3, F_JR_4, F_JR_5; v4: F_HD_3, F_HD_4, F_HD_5, F_AA_1, F_AA_2, F_AA_3, F_AA_4, F_AA_5, F_AA_6, F_AA_7, F_JR_1, F_JR_2, F_JR_3, F_JR_4; v5: F_AA_3, F_AA_4, F_AA_5, F_JR_1, F_JR_2
grep	10,068	809	seeded	pattern recognition	v1: FAULTY_F_DG_4, FAULTY_F_DG_8, FAULTY_F_KP_2, FAULTY_F_KP_5; v2: FAULTY_F_DG_1, FAULTY_F_DG_2, FAULTY_F_KP_3, FAULTY_F_KP_4; v3: FAULTY_F_DG_1, FAULTY_F_DG_2, FAULTY_F_DG_3, FAULTY_F_DG_8, FAULTY_F_DG_10, FAULTY_F_KP_2, FAULTY_F_KP_3, FAULTY_F_KP_7, FAULTY_F_KP_8; v4: FAULTY_F_KP_9, FAULTY_F_DG_3, FAULTY_F_KP_8
sed	14,427	370	real	stream editor	v2: FAULTY_F_AG_2, FAULTY_F_AG_19; v3: FAULTY_F_AG_5, FAULTY_F_AG_6, FAULTY_F_AG_11, FAULTY_F_AG_15, FAULTY_F_AG_17, FAULTY_F_AG_18; v4: FAULTY_F_KRM_1, FAULTY_F_KRM_2, FAULTY_F_KRM_3, FAULTY_F_KRM_4, v5: FAULTY_F_KRM_1, FAULTY_F_KRM_2, FAULTY_F_KRM_8, FAULTY_F_KRM_10; v6: FAULTY_F_KRM_1, FAULTY_F_KRM_4, FAULTY_F_KRM_5, FAULTY_F_KRM_7, FAULTY_F_KRM_8, FAULTY_F_KRM_9; v7: FAULTY_F_KRM_1, FAULTY_F_KRM_2, FAULTY_F_KRM_5, FAULTY_F_KRM_6
libtiff	77,000	78	real	image processing	bug-2007-11-02-371336d-865f7b2, bug-2007-11-23-82e378c-cf05a83, bug-2008-04-15-2e8b2f1-0d27dc0, bug-2008-09-05-d59e7df-5f42dba, bug-2008-12-30-362dee5-565eaa2, bug-2009-02-05-764dbba-2e42d63, bug-2009-06-30-b44af47-e0b51f3, bug-2009-08-28-e8a47d4-023b6df, bug-2009-09-03-6406250-6b6496b, bug-2010-06-30-1563270-1136bdf, bug-2010-11-27-eb326f9-eec7ec0, bug-2010-12-13-96a5fb4-bdba15c
python	407,000	355	real	general-purpose language	bug-69709-69710, bug-69783-69784, bug-69785-69789, bug-69831-69833, bug-69934-69935, bug-69945-69946, bug-70056-70059, bug-70098-70101
gzip	491,000	12	real	data compression	bug-2009-08-16-3fe0caeada-39a362ae9d, bug-2009-09-26-a1d3d4019d-f17cbd13a1, bug-2009-10-09-1a085b1446-118a107f2d, bug-2010-01-30-fc00329e3d-1204630c96, bug-2010-02-19-3eb6091d69-884ef6d16c

3.1 Study design

Benchmark selection: `Siemens` [http://sir.unl.edu/portal/index.php] is a widely used benchmark in fault localisation. It contains seven programs: `print_tokens`, `print_tokens2`, `replace`, `schedule`, `schedule2`, `tcas and tot_info`, each of which has a set of faulty versions. Siemens developed by Siemens Corporate Research performs a variety of tasks: `print_tokens` and `print_tokens2` are lexical analysers, `replace` performs pattern matching and substitution, `schedule` and `schedule2` are priority schedulers, `tcas` is an aircraft collision avoidance system, and `tot_info` computes statistics given input data. Although `Siemens` is all small-sized programs with seeded faults and even a little older, we still include Siemens due to its wide usage in fault localisation.

Besides `Siemens`, our study uses seven popular large real-world programs, each of which also has a set of faulty versions. `space` was first written by the European Space Agency, and functions as an interpreter for an array definition language (ADL). `flex`, `grep`, `sed` and `gzip` are four real-life typical UNIX utility programs, conducting lexical analysis, pattern matching and printing, stream editing and data compression, respectively. `libtiff` is a free and open-source library for reading and writing TIFF (Tagged Image File Format) graphics files on 32- and 64-bit machines. python is a widely used general-purpose, high-level programming language. In order to obtain reliable faults, all large programs adopt real-happening faults except for `flex` and `grep`.

Table 2 lists the statistics including subject names, lines of code, numbers of test cases, fault type, the functional description, as well as the information of each faulty version. We merge the results of `print_tokens` and `print_tokens2`, `schedule` and `schedule2` as each two have similar structures and functionalities. `libtiff`, `python` and `gzip` are collected from ManyBugs [http://repairbenchmarks.cs.umass.edu/ManyBugs/], and the others are from SIR [http://sir.unl.edu/portal/index.php]. We used Gcov 5.4.0 [https://gcc.gnu.org/onlinedocs/gcc/Gcov.html] to collect coverage information. As a reminder, the two large programs of `flex` and `grep` use seeded faults because SIR only provides seeded faults for the two programs.

`SFL` *formula selection*: Recent studies [6, 13] have theoretically proved that ER1′, ER5, GP02, GP03 and GP13 are the maximal formulas for `SFL` in the scenarios of single faults. This line of research, to the best of our knowledge, provides the best criteria for choosing an effective set of formulas from a large number of existing `SFL` formulas. Therefore, we adopt these formulas,

© The Institution of Engineering and Technology 2018

Input: a test suite $\mathcal{T}$
Output: a subset randomly sampled from $\mathcal{T}$, containing at least one failing and one passing test cases

begin sample *passing* test cases
  let $\mathcal{T}^+ \subset \mathcal{T}$ denote the set of *passing* test cases in $\mathcal{T}$
  $\mathcal{T}^+_{sample} \leftarrow$ `sample`$(\mathcal{T}^+)$
begin sample *failing* test cases
  let $\mathcal{T}^- \subset \mathcal{T}$ denote the set of *failing* test cases in $\mathcal{T}$
  $\mathcal{T}^-_{sample} \leftarrow$ `sample`$(\mathcal{T}^-)$
return $\mathcal{T}^+_{sample} \cup \mathcal{T}^-_{sample}$

Function `sample` (*Set* S)
  `/* generate a random number` $n \in [1, |S|]$ `*/`
  $n \leftarrow$ `random_num`$(1, |S|)$
  S′ ← randomly draw n test cases from S
  return S′

Fig. 3 *Algorithm 1: sampling a subset from a test suite $\mathcal{T}$*

specifically, Binary out of the three equivalent maximal formulas in ER5, Nash1 out of the three equivalent formulas in ER1′ and the other three formulas GP02, GP03 and GP19. In light of the equivalence in each group, the remaining of the paper uses ER5 and ER1′ to represent Binary and Naish1, respectively.

Random sampling: We leverage random sampling to generate random test suites from the original test suite. Since `SFL` needs the dynamic runtime information of passing and failing executions to initialise the localisation process, we require that each of the test suite samples should contain at least one failing and one passing test cases. Algorithm 1 (see Fig. 3) describes the details of how to generate a sample from $\mathcal{T}$.

Evaluation process: The steps of the experiments are designed as follows. For each buggy program $\mathcal{P}$,

i. We first randomly generate 10,000 subsets of $\mathcal{T}$. For `gzip`, we sample 1000 subsets instead because the number of test cases in `gzip` is very small.
ii. Next, we run `SFL` with $\mathcal{T}$ and its subsets by using the five maximal formulas and compute the accuracy of the localisation.
iii. Finally, we investigate the relation between the fault localisation effectiveness and the size of test suites (i.e. $\mathcal{T}$ and its samples). Specifically, we validate whether the size of $\mathcal{T}$ matters in fault localisation and whether the largest test suite $\mathcal{T}$ outperforms all its subsets.

All the experiments were conducted on an Ubuntu 16.04 machine with 2.40 GHz Intel(R) Core(TM) i5 CPU and 8 GB of memory.

Evaluation metric: We use the absolute rank of the faulty statement in the ranking list of the statements as a metric to evaluate the effectiveness of `SFL`, as recommended by Parnin and Orso [15]. A higher rank of the faulty statement means better fault localisation performance. For those statements with the same suspiciousness, we adopt a conventional strategy [16] ranking them according to their original line order in the source code.

3.2 Quantitative results

We use pooled mean and standard deviation [17] to compute and present our experimental results. In statistics, pooled mean and standard deviation are methods for estimating the mean and standard deviation of different sample sets when the mean of each sample set may be different. Our experiment has a similar context. In our experiment, each subject program has a set of distinct faulty versions, and each faulty version has a large number of test suites randomly sampled from $\mathcal{T}$. For each sample, we apply `SFL` to obtain a data point, i.e., the rank of the faulty statement. For a subject program, the mean of the localisation effectiveness for each faulty version is usually different. Therefore, these two statistic methods well fit here.

Suppose a program has a total of r faulty versions and there are n_i random samples of $\mathcal{T}$ for the faulty version i. Let X_{ij} denote the rank of the faulty statement obtained by applying `SFL` with the jth sample test suite of $\mathcal{T}$ (where $1 \le j \le n_i$). Then for a faulty version $i \in [1, r]$, its mean of localisation accuracy is defined as

$$\overline{X_i} = \frac{\sum_{j=1}^{n_i} X_{ij}}{n_i}$$

and its standard deviation is defined as

$$\sigma_i = \sqrt{\frac{\sum_{j=1}^{n_i} (X_{ij} - \overline{X_i})^2}{(n_i - 1)}}$$

For a subject program, its pooled mean and standard deviation are computed as follows:

$$\text{mean} = \frac{\sum_{i=1}^{r}\sum_{j=1}^{n_i} X_{ij}}{\sum_{i=1}^{r} n_i} = \frac{\sum_{i=1}^{r} n_i \overline{X_i}}{\sum_{i=1}^{r} n_i}$$

$$\text{StdDev} = \sqrt{\frac{\sum_{i=1}^{r}\sum_{j=1}^{n_i} (X_{ij} - \overline{X_i})^2}{\sum_{i=1}^{r} (n_i - 1)}} = \sqrt{\frac{\sum_{i=1}^{r} (n_i - 1)\sigma_i^2}{\sum_{i=1}^{r} (n_i - 1)}}$$

Tables 3 and 4 show the statistics of the original test suites, the sampled test suites and the accuracy data of fault localisation by using these test suites in small programs (`Siemens`) and large programs, respectively. The columns are explained below:

- $|\mathcal{T}|$: the number of test cases in the original test suite $\mathcal{T}$.
- *Mean size of all samples:* the mean of the sizes of the test suites randomly sampled from $\mathcal{T}$.
- *StdDev of sizes of samples:* the standard deviation of sizes of the sampled test suites.
- *Mean rank of original suite:* the rank of the faulty statement output by `SFL` with $\mathcal{T}$ and the formula specified in the parentheses (e.g. ER5, GP02).
- *Mean rank of all samples:* the pooled mean rank of the faulty statement output by `SFL` with the sampled test suites.
- *StdDev of ranks of all samples:* the pooled standard deviation of the rank of the faulty statement with the sampled test suites.
- *Mean percentage of better and equal samples:* the mean proportion of the samples with which the localisation accuracy is equal or improved.
- *Mean size of better and equal samples:* the mean size of the samples with which the localisation accuracy remains or improves.
- *StdDev of sizes of better and equal samples:* the standard deviation of the sizes of the samples with which the localisation accuracy remains or improves.

Take `print_tokens` (2 ver.) using ER5 formula (shown in the first row of data in Table 3) as an example. The mean size of all the samples is 181.98, i.e. 49.59% of $|\mathcal{T}|$. The mean rank of the faulty statement by using samples is a little smaller (that is better) than that computed with the whole test suite $\mathcal{T}$. The standard deviation of the ranks is 15.95. Compared to the mean rank (46.56) of all samples, this number indicates an obvious fluctuation of localisation effectiveness. Moreover, on average 73.15% of the samples perform better than or equal to $\mathcal{T}$ in terms of faulty statement ranking, although their sizes are much smaller than $\mathcal{T}$, only 48.21% of $|\mathcal{T}|$.

Based on the results in Tables 3 and 4, it is surprising to find that a total average of 69.36 and 67.90% of the random samples, with smaller sizes, deliver improved localisation results over the original test suites in small programs (`Siemens`) and large programs, respectively. It is also unexpected that there is usually a large standard deviation of the ranks of the faulty statements using random samples in both small programs (`Siemens`) and large programs.

© The Institution of Engineering and Technology 2018

Table 3 Original test suites versus sampled test suites in small programs (Siemens)

Program	$\|\mathcal{T}\|$	Mean size of samples	StdDev of sizes of samples	Mean rank of original suite	Mean rank of all samples	StdDev of ranks of all samples	Mean percentage of better and equal samples	Mean size of better and equal samples	StdDev of sizes of better and equal samples
print_tokens (2 ver.)	367	181.98	98.21	49.60 (ER5)	46.56 (ER5)	15.95 (ER5)	73.15% (ER5)	176.94 (ER5)	97.60 (ER5)
				36.53 (ER1′)	36.20 (ER1′)	18.16 (ER1′)	67.62% (ER1′)	188.63 (ER1′)	97.19 (ER1′)
				27.87 (GP02)	30.59 (GP02)	22.75 (GP02)	59.40% (GP02)	183.66 (GP02)	96.92 (GP02)
				59.93 (GP03)	68.82 (GP03)	46.00 (GP03)	59.08% (GP03)	185.70 (GP03)	98.33 (GP03)
				86.27 (GP19)	86.42 (GP19)	21.48 (GP19)	62.07% (GP19)	182.07 (GP19)	101.44 (GP19)
replace	395	197.77	110.41	59.59 (ER5)	58.47 (ER5)	15.96 (ER5)	71.81% (ER5)	188.61 (ER5)	108.00 (ER5)
				16.00 (ER1′)	18.93 (ER1′)	10.16 (ER1′)	68.78% (ER1′)	217.39 (ER1′)	106.78 (ER1′)
				30.19 (GP02)	30.12 (GP02)	17.91 (GP02)	69.37% (GP02)	203.20 (GP02)	106.91 (GP02)
				64.63 (GP03)	87.07 (GP03)	63.46 (GP03)	60.50% (GP03)	207.08 (GP03)	109.65 (GP03)
				72.33 (GP19)	72.54 (GP19)	19.70 (GP19)	65.08% (GP19)	192.45 (GP19)	110.85 (GP19)
schedule (2 ver.)	231	116.44	64.90	61.31 (ER5)	60.85 (ER5)	8.92 (ER5)	69.04% (ER5)	122.21 (ER5)	63.34 (ER5)
				58.56 (ER1′)	60.27 (ER1′)	15.53 (ER1′)	54.29% (ER1′)	121.67 (ER1′)	62.06 (ER1′)
				56.31 (GP02)	57.50 (GP02)	10.87 (GP02)	58.30% (GP02)	121.95 (GP02)	62.27 (GP02)
				59.63 (GP03)	61.97 (GP03)	35.74 (GP03)	58.24% (GP03)	130.48 (GP03)	60.29 (GP03)
				79.25 (GP19)	79.04 (GP19)	16.68 (GP19)	67.27% (GP19)	131.19 (GP19)	60.88 (GP19)
tcas	83	42.52	22.97	26.50 (ER5)	24.71 (ER5)	4.49 (ER5)	96.28% (ER5)	42.64 (ER5)	22.74 (ER5)
				14.40 (ER1′)	14.31 (ER1′)	2.38 (ER1′)	86.74% (ER1′)	43.65 (ER1′)	22.60 (ER1′)
				18.13 (GP02)	18.07 (GP02)	5.22 (GP02)	74.80% (GP02)	40.33 (GP02)	22.27 (GP02)
				26.17 (GP03)	26.53 (GP03)	11.15 (GP03)	72.20% (GP03)	44.52 (GP03)	22.23 (GP03)
				26.88 (GP19)	26.37 (GP19)	7.51 (GP19)	76.76% (GP19)	45.44 (GP19)	22.35 (GP19)
tot_info	194	85.02	39.06	70.00 (ER5)	65.83 (ER5)	14.12 (ER5)	85.53% (ER5)	87.82 (ER5)	39.51 (ER5)
				55.28 (ER1′)	51.85 (ER1′)	19.28 (ER1′)	71.71% (ER1′)	89.22 (ER1′)	39.88 (ER1′)
				56.67 (GP02)	56.63 (GP02)	23.14 (GP02)	58.65% (GP02)	86.10 (GP02)	40.22 (GP02)
				55.28 (GP03)	55.33 (GP03)	18.99 (GP03)	66.66% (GP03)	84.44 (GP03)	38.62 (GP03)
				35.17 (GP19)	40.25 (GP19)	18.51 (GP19)	69.85% (GP19)	94.14 (GP19)	37.52 (GP19)

To obtain more understanding and insights on the results above, we further use the Kendall rank correlation coefficient [18] (denoted by r) to measure the correlation between the fault localisation effectiveness and test suite sizes. In statistics, the Kendall rank correlation coefficient is a statistic used to measure the correlation between two measured variables. The value of the correlation coefficient r is in the range of $[-1, 1]$. When r lies around $+1$ or -1, then it is said to be a perfect degree of correlation between the two variables. As the correlation coefficient value goes towards 0, the correlation between the two variables will be weaker. Based on the absolute value of the correlation coefficient r, five cases are usually used:

1. $|r| \in [0.8, 1.0]$, very strong correlation;
2. $|r| \in [0.6, 0.8)$, strong correlation;
3. $|r| \in [0.4, 0.6)$, medium correlation;
4. $|r| \in [0.2, 0.4)$, weak correlation;
5. $|r| \in [0.0, 0.2)$, very weak correlation or independence.

Table 5 shows the Kendall rank correlation coefficient between the ranks of the faulty statements and test suite sizes in each program using different maximal formulas. As shown in Table 5, the absolute value of the coefficient r is smaller than 0.2 in all cases. This means that there is no statistically significant correlation between the localisation effectiveness and test suites sizes.

'The results above show that there is no strong correlation between localisation effectiveness and the size of a test suite, namely, a large test suite does not necessarily imply high localisation accuracy'.

3.3 Theoretical analysis and explanation

In this section, we conduct theoretical analysis to understand why localisation effectiveness is not correlated with the size of a test suite. Since how to evaluate the faulty statement with high suspiciousness is the main concern of fault localisation, our theoretical analysis will focus on the calculation of the suspiciousness of the faulty statement in each maximal formula. Specifically, we theoretically analyse the structure of each maximal formula to identify which factors decide the calculation of the suspiciousness of the faulty statement. With those identified factors, we can see which parts of a test suite are correlated with them, and furthermore recognise which parts of a test suite are beneficial or harmful in increasing the suspiciousness of the faulty statement. Thus, we can understand why localisation effectiveness is not correlated with the size of a test suite, and more importantly, we may reveal how a test suite impacts fault localisation by increasing or decreasing the suspiciousness of the faulty statement.

Since our study focuses on single-fault subject programs, we assume that a buggy program $\mathcal{P}$ has only one fault. A single fault could contain a single faulty statement, or a sequence of faulty statements of which executing each implies the execution of the others (e.g. a list of faulty statements in the same basic block). Without loss of generality, we assume that $\mathcal{P}$ has only a faulty statement s_f. We further use N_{st} (and N_{ft}) to represent the total number of passing (and failing) test cases in $\mathcal{T}$, respectively. Hence, based on the definitions of the four variables in (1), we have $N_{st} = a_{00}(s_j) + a_{10}(s_j)$ and $N_{ft} = a_{01}(s_j) + a_{11}(s_j)$. In general, a faulty statement should be executed when a failure occurs, especially in the context of a single fault. Thus, for the faulty statement s_f, $a_{01}(s_f) = 0$ and $a_{11}(s_f) = N_{ft}$ in our study. Note that N_{ft} is the maximal value of the variable a_{11}.

ER5: We take the Binary formula out of the three equivalent formulas to represent ER5. Unlike the other maximal formulas, ER5 only utilises the variable a_{01} that only depends on the information of failing test cases. In other words, all passing test cases are unused for ER5 and therefore can be removed. If $a_{01}(s_j) = 0$ holds, it means that statement s_j is executed by all failing test cases, i.e. $a_{11}(s_j) = N_{ft}$. Therefore, ER5 will assign a suspiciousness value of 1 to those statements whose a_{11} equals to

© The Institution of Engineering and Technology 2018

Table 4 Original test suites versus sampled test suites in large programs

Program	$\|\mathcal{T}\|$	Mean size of samples	StdDev of sizes of samples	Mean rank of original suite	Mean rank of all samples	StdDev of ranks of all samples	Mean percentage of better and equal samples	Mean size of better and equal samples	StdDev of sizes of better and equal samples
space	4333	2186.67	1152.08	1543.72 (ER5)	1350.32 (ER5)	458.97 (ER5)	57.87% (ER5)	2606.35 (ER5)	456.03 (ER5)
				1467.06 (ER1′)	1445.23 (ER1′)	324.89 (ER1′)	48.51% (ER1′)	2448.47 (ER1′)	373.85 (ER1′)
				1466.06 (GP02)	1445.23 (GP02)	332.22 (GP02)	51.10% (GP02)	2466.59 (GP02)	373.90 (GP02)
				1470.89 (GP03)	1290.75 (GP03)	450.04 (GP03)	54.35% (GP03)	2209.97 (GP03)	393.41 (GP03)
				1506.26 (GP19)	1452.16 (GP19)	375.51 (GP19)	53.93% (GP19)	2092.48 (GP19)	302.01 (GP19)
flex	567	284.65	132.40	1921.45 (ER5)	1949.58 (ER5)	424.16 (ER5)	45.88% (ER5)	276.80 (ER5)	127.99 (ER5)
				1870.38 (ER1′)	1868.95 (ER1′)	138.46 (ER1′)	47.46% (ER1′)	298.72 (ER1′)	126.81 (ER1′)
				1931.58 (GP02)	1925.40 (GP02)	151.91 (GP02)	60.44% (GP02)	306.05 (GP02)	129.57 (GP02)
				1923.25 (GP03)	1914.67 (GP03)	125.56 (GP03)	60.35% (GP03)	307.56 (GP03)	128.74 (GP03)
				1937.98 (GP19)	1928.78 (GP19)	182.98 (GP19)	64.44% (GP19)	291.86 (GP19)	129.95 (GP19)
grep	809	405.33	214.19	679.60 (ER5)	686.69 (ER5)	195.19 (ER5)	65.21% (ER5)	431.06 (ER5)	206.19 (ER5)
				335.70 (ER1′)	327.29 (ER1′)	75.17 (ER1′)	65.53% (ER1′)	442.14 (ER1′)	207.53 (ER1′)
				455.45 (GP02)	422.05 (GP02)	317.26 (GP02)	64.50% (GP02)	407.32 (GP02)	206.44 (GP02)
				414.35 (GP03)	516.31 (GP03)	347.32 (GP03)	63.85% (GP03)	434.95 (GP03)	209.37 (GP03)
				374.85 (GP19)	365.71 (GP19)	94.18 (GP19)	66.11% (GP19)	419.31 (GP19)	206.50 (GP19)
sed	360	181.18	97.77	342.77 (ER5)	365.94 (ER5)	117.18 (ER5)	62.20% (ER5)	194.58 (ER5)	90.53 (ER5)
				347.58 (ER1′)	353.96 (ER1′)	89.88 (ER1′)	64.64% (ER1′)	203.29 (ER1′)	92.92 (ER1′)
				470.08 (GP02)	466.19 (GP02)	80.48 (GP02)	70.21% (GP02)	187.93 (GP02)	98.63 (GP02)
				567.00 (GP03)	603.25 (GP03)	202.04 (GP03)	70.97% (GP03)	185.83 (GP03)	98.81 (GP03)
				425.85 (GP19)	412.38 (GP19)	152.52 (GP19)	65.06% (GP19)	187.94 (GP19)	99.42 (GP19)
libtiff	78	37.47	18.75	4634.00 (ER5)	4626.32 (ER5)	134.43 (ER5)	48.04% (ER5)	51.47 (ER5)	13.18 (ER5)
				4634.17 (ER1′)	4672.22 (ER1′)	72.77 (ER1′)	43.95% (ER1′)	53.09 (ER1′)	11.79 (ER1′)
				5326.06 (GP02)	5263.08 (GP02)	97.01 (GP02)	92.71% (GP02)	38.51 (GP02)	17.54 (GP02)
				5326.53 (GP03)	5269.18 (GP03)	81.71 (GP03)	95.43% (GP03)	35.19 (GP03)	18.34 (GP03)
				4969.89 (GP19)	4754.71 (GP03)	104.73 (GP03)	97.14% (GP03)	33.93 (GP03)	16.95 (GP19)
python	355	176.27	76.12	645.25 (ER5)	646.90 (ER5)	102.25 (ER5)	73.21% (ER5)	48.40 (ER5)	43.91 (ER5)
				668.25 (ER1′)	668.41 (ER1′)	100.47 (ER1′)	92.06% (ER1′)	48.64 (ER1′)	43.89 (ER1′)
				828.00 (GP02)	799.48 (GP02)	74.09 (GP02)	85.38% (GP02)	48.69 (GP02)	43.99(GP02)
				867.25 (GP03)	830.66 (GP03)	87.99 (GP03)	92.52% (GP03)	48.63 (GP03)	43.97 (GP03)
				668.75 (GP19)	668.66 (GP19)	82.60 (GP19)	95.28% (GP19)	48.57 (GP19)	43.96 (GP19)
gzip	12	6.78	3.44	674.67 (ER5)	655.79 (ER5)	165.49 (ER5)	64.55% (ER5)	4.28 (ER5)	2.42 (ER5)
				757.50 (ER1′)	733.19 (ER1′)	157.93 (ER1′)	66.77% (ER1′)	5.26 (ER1′)	2.33 (ER1′)
				510.17 (GP02)	519.92 (GP02)	78.44 (GP02)	57.65% (GP02)	5.52 (GP02)	2.82 (GP02)
				514.17 (GP03)	505.89 (GP03)	59.23 (GP03)	84.80% (GP03)	4.92 (GP03)	2.53 (GP03)
				643.67 (GP19)	634.83 (GP19)	82.18 (GP19)	84.43% (GP19)	4.82 (GP19)	2.44 (GP19)

© The Institution of Engineering and Technology 2018

Table 5 Kendall rank correlation coefficient between the localisation effectiveness and test suite sizes

Program	ER5	ER1′	GP02	GP03	GP19
print_tokens (2 ver.)	0.07	−0.03	0.01	−0.02	0.04
replace	0.05	0.02	0.02	0.06	0.08
schedule (2 ver.)	0.03	0.01	0.00	0.02	−0.01
tcas	0.05	−0.02	0.05	0.03	0.07
tot_info	0.07	0.03	0.06	−0.02	0.07
space	0.02	0.01	0.01	−0.01	0.01
flex	−0.03	0.01	0.00	0.01	0.00
grep	−0.04	−0.06	0.01	0.04	−0.06
sed	−0.06	−0.05	−0.06	−0.06	0.03
libtiff	−0.05	−0.06	0.07	0.07	0.10
python	0.11	0.08	0.09	0.04	0.06
gzip	0.10	0.09	0.03	0.06	0.10

N_{ft}, 0 otherwise. For the faulty statement s_f, its a_{11} is always N_{ft} and thus its suspiciousness value is always 1. It means that the faulty statement has the highest suspiciousness value. Therefore, the only factor influencing the localisation effectiveness is those statements of which the a_{11} values equal to N_{ft} because they have the same suspiciousness as the faulty statement. Those statements of which the a_{11} values equal to N_{ft} mean that they are executed by all failing test cases. If we add more failing test cases decreasing the number of those statements executed by all failing test cases, the number of the statements with the same suspiciousness as the faulty statement will decrease.

Because the set of failing test cases is different among the samples, the sets of statements executed by all failing test cases are often different. This is the main reason why the localisation effectiveness of ER5 fluctuates among samples.

ER1′, GP03, GP19: We take Naish1 from the three equivalent formulas to represent ER1′. For any statement s_j, $a_{10}(s_j) = N_{st} - a_{00}(s_j)$ and $a_{01}(s_j) = N_{ft} - a_{11}(s_j)$ hold. For simplicity, we replace $a_{10}(s_j)$ and $a_{01}(s_j)$ with $N_{st} - a_{00}(s_j)$ and $N_{ft} - a_{11}(s_j)$ in Naish1, GP03 and GP19, respectively, and thus these formulas have only two variables (i.e. a_{11} and a_{00}) thereafter. The variable a_{11} is only determined by failing test cases and a_{00} is determined by passing test cases. Furthermore, the two variables are independent of each other. Therefore, we just need to study each separately by considering the other as a constant to investigate the impact of its related test cases (i.e. failing test cases or passing test cases) on the suspiciousness evaluation of the faulty statement.

The impact of failing test cases (a_{11}): In this case, we assume a_{00} to be a constant. For a faulty statement, since it a_{11} is the maximal value N_{ft}, the faulty statement has obtained the maximal benefit (the highest possible suspiciousness) from failing test cases in Naish1, GP03 and GP19. Therefore, the only optimisation related to failing test cases is to minimise the number of correct statements which are executed by all the failing test cases.

The impact of passing test cases (a_{00}): In this case, we assume a_{11} to be a constant. Since the faulty statement s_f has obtained the maximal benefit from failing test cases (i.e. $a_{11}(s_f) = N_{ft}$), we further replace $a_{11}(s_f)$ with N_{ft} in each formula. Therefore, its suspiciousness is mainly determined by the magnitude of benefit from passing test cases. For ER1′, since $a_{01}(s_f) = 0$ always holds, the suspiciousness value of s_f equals $a_{00}(s_f)$. For GP03, the suspiciousness value of s_f is decided by $-\sqrt{N_{ft} - a_{00}(s_f)}$, which is also determined by $a_{00}(s_f)$. For GP19, the new equivalent form of calculating the suspiciousness of s_f is $N_{st}\sqrt{|2a_{00}(s_f) + N_{ft} - N_{st}|}$, also decided by $a_{00}(s_f)$.

Based on the analysis above, the suspiciousness of the faulty statement s_f calculated by ER1′, GP03 and GP19 is mainly determined by $a_{00}(s_f)$, and the faulty statement can achieve the highest suspiciousness when $a_{00}(s_f)$ increases to the maximal value N_{st}. Since N_{st} varies in different test suites, it is difficult to distinguish which absolute value of $a_{00}(s_f)$ should be better among different test suites. To address this problem, based on $N_{st} = a_{00}(s_f) + a_{10}(s_f)$, we apply normalisation to define a new metric *Passing Test Discrimination* (PTD) consistent with $a_{00}(s_f)$, i.e.

$$\mathrm{PTD}(s_f) = \frac{a_{00}(s_f)}{N_{st}} \quad \text{(passing test discrimination)}$$

To verify the analysis above, we use Kendall rank correlation coefficient [18] (denoted as r) to measure the correlation between fault localisation effectiveness and PTD. Table 6 shows that r is smaller than −0.8 in all cases, i.e. as PTD increases, the ranks of faulty statements always become higher. The analysis and results above show that the suspiciousness of the faulty statement is consistent with PTD of a test suite and has no strong correlation with the size of a test suite. In practice, since it cannot be guaranteed that the execution of a passing test case is free of a faulty statement, PTD can greatly vary among the samples. That is why there is an apparent fluctuation of localisation effectiveness in these formulas.

GP02: Likewise, we conduct case analysis on GP02:

The impact of failing test cases (a_{11}): Similarly, we replace $a_{10}(s_j)$ with $N_{st} - a_{00}(s_j)$ in GP02, and assume a_{00} to be a constant. In this case, a statement can also obtain the highest value if it a_{11} is the maximal value N_{ft}. Since $a_{11}(s_j) = N_{ft}$, the faulty statement s_f has also obtained the maximal benefit from failing test cases in GP02.

The impact of passing test cases (a_{00}): The analysis of failing test cases above shows that the suspiciousness of the faulty statement s_f is mainly decided by the magnitude of the benefit obtained from passing test cases. Therefore, we further replace $a_{11}(s_f)$ with N_{ft} in GP02. Apparently, the suspiciousness of s_f is determined by $2\sqrt{a_{00}(s_f)} + \sqrt{N_{st} - a_{00}(s_f)}$. With differentiation, we can easily identify that the faulty statement s_f obtains the highest suspiciousness value when $a_{00}(s_f) = 4N_{st}/5$. The executions of passing test cases cannot guarantee that $a_{00}(s_f)$ is around $4N_{st}/5$, and thus there is an also noticeable fluctuation in GP02.

Summary: Let us recall the basic idea of SFL [5]. SFL tries to assign a high suspiciousness value to a statement frequently executed by failing test cases and infrequently executed by passing test cases. Following this idea, we should increase $a_{11}(s_f)$ and decrease $a_{00}(s_f)$, e.g. optimising test suites, to maximise the suspiciousness value of the faulty statement s_f. In general, a faulty statement should be executed by a failing test case when a failure occurs. Therefore, the a_{11} is always maximal for the faulty statement. ER5 takes advantage of this property by just using the information of failing test cases and therefore passing test cases do not affect ER5. Besides failing test cases, ER1′, GP03 and GP19 use the information of passing test cases and try to obtain a high a_{00} for the faulty statement. Although GP02 also utilises the information of passing test cases, its intuition is different from that of most existing SFL techniques. For example, suppose that a faulty statement is not executed by any passing test case, i.e. its a_{00} equals to N_{st}. Because the best value of a_{00} for the faulty statement

© The Institution of Engineering and Technology 2018

Table 6 Kendall rank correlation coefficient between the localisation effectiveness and PTD

Program	ER1′	GP03	GP19
print_tokens (2 ver.)	−0.87	−0.88	−0.88
replace	−0.88	−0.82	−0.93
Schedule (2 ver.)	−0.90	−0.86	−0.90
tcas	−0.89	−0.82	−0.83
tot_info	−0.90	−0.84	−0.88
space	−0.82	−0.81	−0.91
flex	−0.83	−0.81	−0.80
grep	−0.84	−0.86	−0.81
sed	−0.86	−0.85	−0.86
libtiff	−0.86	−0.88	−0.89
python	−0.91	−0.81	−0.86
gzip	−0.88	−0.84	−0.90

Table 7 Statistical comparison among $\mathcal{T}_{good}$, $\mathcal{T}_{orig}$ and $\mathcal{T}_{bad}$

Formula	Comparison	2-tailed	1-tailed (right)	1-tailed (left)	Conclusion
ER1′	$\mathcal{T}_{good}$ versus $\mathcal{T}_{orig}$	1.29×10^{-41}	1.00	6.50×10^{-42}	BETTER
	$\mathcal{T}_{orig}$ versus $\mathcal{T}_{bad}$	8.72×10^{-55}	1.00	4.40×10^{-55}	BETTER
GP02	$\mathcal{T}_{good}$ versus $\mathcal{T}_{orig}$	4.50×10^{-4}	0.99	2.25×10^{-4}	BETTER
	$\mathcal{T}_{orig}$ versus $\mathcal{T}_{bad}$	4.95×10^{-4}	0.99	2.48×10^{-4}	BETTER
GP03	$\mathcal{T}_{good}$ versus $\mathcal{T}_{orig}$	4.53×10^{-44}	1.00	2.28×10^{-44}	BETTER
	$\mathcal{T}_{orig}$ versus $\mathcal{T}_{bad}$	1.35×10^{-42}	1.00	6.78×10^{-43}	BETTER
GP19	$\mathcal{T}_{good}$ versus $\mathcal{T}_{orig}$	2.05×10^{-43}	1.00	1.03×10^{-43}	BETTER
	$\mathcal{T}_{orig}$ versus $\mathcal{T}_{bad}$	3.52×10^{-47}	1.00	1.77×10^{-47}	BETTER

is $4N_{st}/5$, the suspiciousness of the faulty statement is not maximal in GP02. GP02 [13] is learned from genetic programming and may have new insights and intuitions ignored by human beings. This interesting topic should be further studied.

In contrast to the fact that the executions of failing test cases can always include the faulty statement, the executions of passing test cases cannot be guaranteed to be free of a faulty statement, leading to a coincidental correctness problem [19]. Therefore, the different PTD of passing test cases in different sampled test suites is the reason for causing the significant fluctuation among these samples.

Based on the analysis above, the PTD of a test suite is consistent with the suspiciousness of the faulty statement being faulty and therefore is correlated with faulty localisation effectiveness. However, the size of a test suite has no strong connection with localisation effectiveness. Therefore, we summarise the impact of the different parts of a test suite on fault localisation as follows:

Failing test cases: In a single-fault program, the faulty statement must be executed by all the failing test cases. Therefore, the a_{11} of the faulty statement has its maximum value N_{ft}. However, for those non-faulty statements which are executed by all the failing test cases (denoted as a set of statements S_{correct}), their a_{11} is also N_{ft}. In order to improve the rank of the faulty statement, what we can do with the failing test cases is to create new failing test cases to decrease the non-faulty statements in S_{correct}. In other words, minimise the number of non-faulty statements executed by all failing test cases.

Passing test cases: Based on the definition of PTD, two kinds of passing test cases can affect the fault localisation effectiveness. One is the passing test cases which do not execute the faulty statement and this component has a positive impact on fault localisation. The other is the passing test cases which execute the faulty statement and this component penalise fault localisation. Therefore, in order to improve localisation accuracy, we should try to expand the first component and eliminate the second one.

To further understand and verify the impact of passing test cases, we compare the fault localisation effectiveness among the original test suites $\mathcal{T}_{orig}$, the original test suites without those passing test cases which do not execute the faulty statement (denoted as $\mathcal{T}_{bad}$), and the original test suites without those passing test cases which exercise the faulty statement (denoted as $\mathcal{T}_{good}$). Based on the analysis, the relationship of localisation effectiveness should be $\mathcal{T}_{good} > \mathcal{T}_{orig} > \mathcal{T}_{bad}$. To verify the above relationship, we adopt a rigorous and scientific method: the paired Wilcoxon-signed-rank test [18]. The paired Wilcoxon-signed-rank test is a non-parametric statistical hypothesis test for testing that the differences between pairs of measurements $F(x)$ and $G(y)$, which do not follow a normal distribution. The study conducts two paired Wilcoxon-signed-rank tests: the localisation effectiveness of all faulty versions using $\mathcal{T}_{good}$ versus that using $\mathcal{T}_{orig}$ and the localisation effectiveness using $\mathcal{T}_{orig}$ versus that using $\mathcal{T}_{bad}$ to verify the above relationship. Each test uses both the 2-tailed and 1-tailed checking at the σ level of 0.05. Because ER5 does not use the information of passing test cases, its effectiveness remains the same among the three test suites. Therefore, Table 7 does not include the statistical results of ER5. As shown in Table 7, all formulas even including GP02 conform to the effectiveness relationship:

$$\mathcal{T}_{good} > \mathcal{T}_{orig} > \mathcal{T}_{bad}$$

4 Simple application of using PTD

In addition to the theoretical analysis above, this section presents the further empirical validation of our hypothesis.

4.1 Test suite optimisation

Our analysis in Section 3.3 pinpoints that the PTD of a test suite is consistent with the suspiciousness of the faulty statement. To empirically validate the analysis result, we leverage this knowledge and propose a simple test suite optimisation technique, referred to as *Passing Tests Discrimination-based Test-suite Optimisation* (PTD-TO). The goal of PTD-TO is to obtain a smaller test suite that performs no worse than the original one. In light of the relationship between PTD and the suspiciousness of a faulty statement, the

© The Institution of Engineering and Technology 2018

basic idea of PTD-TO is *to increase the suspiciousness of the faulty statement by increasing the* PTD *of the test suite*. More concretely, PTD-TO tries to identify and remove passing test cases that execute the faulty statement.

PTD-TO takes the advantage of the dynamic runtime information from failing test cases to identify such passing test cases. Generally, if a passing test case executes similar statements as the failing test cases, then we deem it as a candidate likely to execute the faulty statement. The following describes this heuristic as well as the measurement of the similarity between passing and failing test cases: We define the set of statements that are executed by *all* failing test cases as *Minimum Suspicious Set* (MSS). Clearly, the faulty statement is in MSS as it is the reason why a test case is classified as failing, and the other statements are benign. Next, given a passing test case t and its covered statements S, if S shares a large portion of MSS (i.e. $|S \cap \mathrm{MSS}|/|\mathrm{MSS}| \geq \theta$ where θ is a threshold), then we remove t as we believe that t tends to execute the faulty statement with a certain likelihood. The likelihood is determined by the similarity threshold θ.

There is one issue related to our approach: what is the appropriate value for θ? For example, suppose a passing test case only covers one statement in MSS. For this passing test case, its chance of covering the faulty statement is low and our approach should not remove this test case. If $\theta = 100\%$, then the passing test case definitely exercises the faulty statement and should be removed. It is necessary to conduct an experimental study to investigate what the proper value of θ should be in practice. The following study empirically investigates a feasible percentage for PTD-TO. Because ER5 only relies on the information of failing test cases and does not use any passing test cases, PTD-TO will not change its fault localisation results. Therefore, we do not include it in this study.

Note that since PTD-TO takes as input the coverage matrix generated by existing fault localisation techniques and performs a linear scan over the matrix, hence it incurs negligible overhead.

4.2 Experimental design

This experiment also uses the subject programs, their original test suites and all random samples of the original test suites in Section 3. The process of our test suite optimisation is listed as follows:

i. PTD-TO removes those passing tests whose execution covers more than a specific percentage of the statements in MSS. Our optimisation adopts the percentage of the statements in MSS from 10 to 90%, in 10% increments. This step only performs the optimisation on existing test suites, i.e. the original test suites.
ii. We analyse the optimisation results on original test suites to find a feasible percentage, and investigate the reasons for this feasible percentage.
iii. Besides the original test suites, our optimisation should be applied to more test suites to investigate its effectiveness. Furthermore, it also needs to answer whether PTD-TO performs better than the approach that randomly generates a subset of the original test suite as an optimised test suite. Therefore, we apply our optimisation approach to all random samples used in Section 3 and compare PTD-TO with a pure random approach that randomly generates the subsets of the random samples as the optimised test suites.

4.3 Optimisation on existing test suites

Table 8 shows the mean rank of the faulty statement using the existing test suites and their optimised test suites generated by PTD-TO in each program. Take ER1′ in *replace* as an example. The mean rank of the faulty statement using the existing test suite for *replace* is 16.00 (column 'Original rank') in ER1′. PTD-TO removes those passing test cases whose execution covers more than a percentage (between 10 and 90%) of the statements in MSS. For example, after PTD-TO removes those passing test cases of which the execution covers >90% of the statements in MSS, the mean rank of the faulty statement using the optimised test suites for *replace* is 11.67 (column '90%') in ER1′, which is <16.00. That means ER1′ using the original test suite performs a bit worse than ER1′ using the optimised test suite. In GP02, PTD-TO does not perform well, which conforms to our aforementioned analysis. Compared with the other percentages, we observe that PTD-TO using 90% in ER1′, GP03 and GP19 always performs better than the original test suites in all subject programs except for *print_tokens(2 ver.)*. This suggests that the feasible percentage value for PTD-TO is 90%, i.e. PTD-TO should remove those passing test cases that execute <90% of the statements in MSS.

Fig. 4 shows the ratio of the size of the optimised test suites to that of the original test suites when our optimisation approach adopts 90% as its reduction value. Table 9 illustrates the execution time in seconds of the optimised test suites and the original test suites, and the ratio of the time between them. As shown in Fig. 4 and Table 9, our approach produces significantly smaller test suites incurring considerably less time cost.

As a reminder, PTD-TO aims at producing a smaller test suite that performs no worse than the original one. Surprisingly, the smaller test suite performs slightly better than the original one.

From the results above, we further analyse the reason why PTD-TO performs well using 90% reduction value. It seems that those passing test cases covering more than a high percentage of the statements in MSS have a high probability of executing the faulty statement. We select several high percentages (from 90 to 99%, in 1% increments) of the statements in MSS to investigate this conjecture. Table 10 illustrates the percentage of the test cases executing the faulty statement in those passing test cases covering more than a specific percentage of the statements in MSS. As shown in Table 10, the set of those passing test cases covering more than a higher percentage of the statements in MSS, the percentage of the test cases executing the faulty statement in this set increases. Thus, it reveals why PTD-TO performs well using 90% as the reduction value. As a reminder, the feasible value does not mean it is the optimal value for PTD-TO. This study just concentrates on validating the analysis in Section 3.3 and demonstrating the potential application of PTD. Finding an optimal value for PTD-TO needs much future research, and is not the main focus of this study.

The above study shows that it is useful to improve fault localisation by removing those passing tests whose execution statements is highly similar to the statements in MSS. It is interesting to investigate the effect of keeping those similar test cases on fault localisation. Thus, we remove those dissimilar test cases that cover a low percentage of the statements in MSS to keep the similar ones, and compared their localisation effectiveness with that of the original suites.

Table 11 shows the statistical results using the paired Wilcoxon-signed-rank test [18] at the σ level of 0.05. Take 1% as an example. After removing those test cases whose executions cover less than 1% of the statements in MSS, the reduced test suites in all suspiciousness evaluation formulas perform WORSE than the original ones. As shown in Table 11, except for few SIMILAR results, almost all reduced test suites in different percentages significantly have WORSE results than the original ones. Therefore, the passing test cases whose execution statements have high similarity with the statements in MSS are not much useful for fault localisation.

4.4 PTD-TO and pure random approach on all random samples

In order to further verify the effectiveness of our optimisation approach, we apply PTD-TO to all random samples generated for each subject program in Section 3 and compare PTD-TO with the pure random approach that randomly generates the subsets of the random samples as the optimised test suites. Since 90% is a feasible threshold for PTD-TO, we use this value for this experiment. For each faulty version of a subject program, there is a large number of random samples (10,000 random samples from the experiments in Section 3). Therefore, we conduct a rigorous and

© The Institution of Engineering and Technology 2018

Table 8 Localisation effectiveness of the existing test suites and their optimised suites

Program	Formula	Original rank	Percentage of the statements in MSS								
			10%	20%	30%	40%	50%	60%	70%	80%	90%
print_token (2 ver.)	ER1′	36.53	44.20	44.20	43.53	43.53	43.40	43.20	38.20	37.73	37.47
	GP02	27.87	44.20	44.20	43.53	43.53	43.40	46.40	38.47	35.73	37.00
	GP03	59.93	60.67	60.67	60.00	60.00	59.87	59.67	54.67	54.20	53.93
	GP19	86.27	55.27	55.27	49.87	49.53	49.20	48.60	40.93	37.87	35.40
replace	ER1′	16.00	42.57	42.10	37.13	36.57	34.40	34.10	28.87	16.87	11.67
	GP02	30.19	40.13	42.08	40.10	34.83	35.17	41.57	45.00	34.93	27.40
	GP03	64.63	55.80	55.50	54.53	52.63	52.80	50.13	50.53	46.83	42.83
	GP19	72.33	58.07	57.63	52.47	56.57	51.00	45.90	47.60	47.47	44.57
schedule (2 ver.)	ER1′	58.56	66.28	66.17	66.56	60.83	60.28	58.11	58.44	57.39	51.89
	GP02	56.31	65.33	60.17	60.28	67.28	63.89	64.28	61.17	55.44	53.06
	GP03	59.63	64.06	62.83	62.11	59.22	60.67	60.44	55.33	54.72	53.17
	GP19	79.75	66.06	66.44	60.61	61.78	60.22	56.61	55.61	48.06	44.61
tcas	ER1′	14.40	14.19	14.19	14.19	14.19	14.19	13.62	13.46	13.78	12.16
	GP02	18.13	14.19	14.19	14.19	14.19	14.19	13.46	13.35	13.35	17.14
	GP03	26.17	20.11	20.43	20.43	20.57	16.35	16.35	16.35	16.35	15.78
	GP19	26.88	16.24	17.08	17.08	15.68	14.19	14.19	14.19	14.19	13.43
tot_info	ER1′	55.28	61.25	61.25	61.25	60.42	59.58	58.75	58.67	55.00	51.42
	GP02	56.67	63.08	63.08	63.08	63.08	58.08	58.08	58.33	51.75	48.17
	GP03	55.28	57.92	57.92	57.92	57.92	57.75	57.75	57.75	51.75	48.17
	GP19	25.17	37.50	37.08	37.08	37.08	34.33	34.50	34.33	20.67	23.17
space	ER1′	1467.06	1420.40	1524.00	1678.80	1422.40	1426.87	1427.00	1426.20	1425.07	1424.80
	GP02	1466.06	1507.81	1442.61	1676.21	1421.55	1425.35	1425.41	1424.61	1423.48	1423.21
	GP03	1470.89	1584.71	1487.37	1283.77	1453.97	1438.37	1436.17	1437.51	1438.64	1438.97
	GP19	1506.26	1521.81	1523.95	1530.48	1530.35	1528.68	1527.81	1499.28	1517.61	1463.75
flex	ER1′	1870.38	1909.64	1909.64	1909.64	1896.49	1891.92	1876.98	1876.98	1867.55	1858.11
	GP02	1931.58	1925.89	1925.89	1925.89	1934.77	1936.83	1924.70	1924.70	1921.96	1921.96
	GP03	1923.25	1926.40	1926.40	1926.40	1918.36	1916.45	1919.91	1919.91	1919.91	1919.91
	GP19	1937.98	1932.72	1932.72	1932.72	1920.81	1919.38	1925.77	1925.77	1933.55	1933.83
grep	ER1′	335.70	359.70	363.10	349.90	373.05	361.50	346.00	354.85	331.95	308.85
	GP02	455.45	422.40	412.80	389.50	459.90	481.85	528.15	465.85	610.35	589.65
	GP03	414.35	415.80	409.85	357.95	374.45	362.95	333.15	391.95	372.00	376.40
	GP19	374.85	392.15	379.60	349.45	368.85	357.30	341.45	351.05	310.40	308.35
sed	ER1′	347.58	387.85	387.23	387.23	386.54	385.85	377.50	347.31	347.31	340.62
	GP02	470.08	414.77	414.73	414.73	414.92	415.12	405.54	492.54	493.54	486.65
	GP03	567.00	414.77	414.15	414.15	413.46	412.77	416.12	572.54	572.54	565.85
	GP19	425.85	390.65	390.92	390.92	391.12	390.42	391.96	391.08	395.46	388.77
libtiff	ER1′	4634.17	4644.43	4644.43	4643.57	4643.57	4643.57	4643.43	4633.57	4631.57	4591.57
	GP02	5326.06	5339.62	5301.62	5290.62	5285.34	5283.91	5283.91	5283.91	5255.34	5255.34
	GP03	5326.53	5347.48	5284.91	5291.34	5300.05	5293.91	5293.91	5293.91	5265.34	5265.34
	GP19	4969.89	4964.43	4953.34	4950.76	4948.54	4944.43	4944.43	4940.71	4934.57	4932.57
python	ER1′	668.25	668.75	665.25	662.50	652.50	650.00	645.00	645.00	642.00	634.50
	GP02	828.00	837.75	832.50	807.50	785.00	762.50	732.50	725.00	707.50	682.50
	GP03	867.25	875.25	870.00	845.00	822.50	799.50	769.50	762.25	744.25	719.75
	GP19	668.75	673.75	667.75	665.25	661.75	657.75	654.25	651.25	648.75	647.75
gzip	ER1′	757.50	777.40	770.00	766.80	766.80	752.00	752.00	752.00	706.00	706.00
	GP02	510.17	518.80	518.80	521.40	518.80	516.40	515.80	504.80	492.20	484.80
	GP03	514.17	564.00	564.00	516.40	516.40	507.40	490.00	481.00	476.20	472.20
	GP19	643.67	662.80	662.80	662.80	656.60	656.60	645.60	652.00	644.80	630.40

scientific comparison, the paired Wilcoxon-signed-rank test [20], to evaluate the effectiveness on each faulty version of a subject program using PTD-TO overall random samples and their random subsets generated by the pure random approach.

The experiments performed two paired Wilcoxon-signed-rank tests [18]: the localisation effectiveness of the optimised test suites by PTD-TO on all random samples versus that of all random samples, and versus that of optimised test suites randomly generated by pure random approach on all random samples in each faulty version. The test uses both the 2-tailed and 1-tailed checking at the σ level of 0.05 to check that the optimised test suites have SIMILAR, WORSE or BETTER effectiveness over their original random samples and their random subsets generated by the pure random approach in each faulty version.

Table 12 summarises the number of SIMILAR, BETTER and WORSE cases in all versions of each program by using the effectiveness of test suites optimised by PTD-TO over that of all original random samples, and over that of the test suites optimised by the pure random approach. Table 12 also lists the execution time of the optimised test suites (column 'Optimised Tests'), the execution time of the original random samples (column 'Original Tests'), the ratio of the execution time of the optimised test suites to that of the original random samples (column 'Time Ratio'), and the ratio of the size of the optimised test suites to that of the

© The Institution of Engineering and Technology 2018

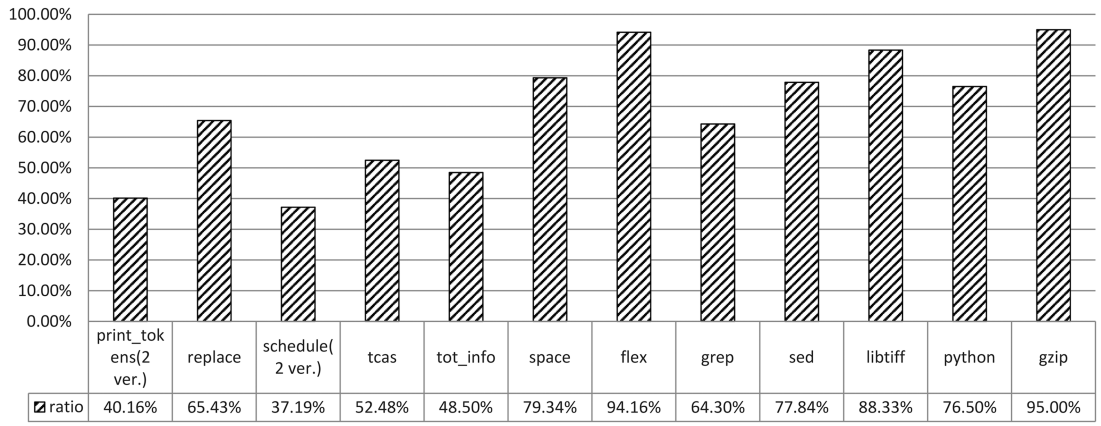

Fig. 4 *Size of optimised test suites versus that of the original test suites with the reduction value of 90%*

Table 9 Time cost of optimised test suites versus that of the original test suites with the reduction value of 90%

Program	Execution time, s		Time ratio, %
	Optimised tests	Original tests	
print_tokens (2 ver.)	3.81	8.17	46.59
replace	5.66	8.83	64.07
schedule (2 ver.)	5.42	12.14	44.62
tcas	1.62	3.77	43.04
tot_info	4.97	9.77	50.90
space	274.16	375.41	73.03
flex	77.40	89.25	86.73
grep	58.31	92.75	62.87
sed	34.00	40.83	83.27
libtiff	13.95	16.43	84.90
python	9.95	14.00	71.06
gzip	2.56	2.80	91.56

Table 10 Percentage of the test cases executing the faulty statement in the removed test cases

Program	90%	91%	92%	93%	94%	95%	96%	97%	98%	99%
print_tokens (2 ver.)	61.59	62.75	63.82	65.86	75.39	76.29	81.21	87.28	97.62	99.92
replace	66.13	66.87	67.25	67.45	67.74	69.99	75.06	82.06	83.22	93.33
schedule (2 ver.)	94.68	94.90	95.30	95.60	96.13	96.31	97.15	98.46	98.43	98.47
tcas	92.32	92.80	94.14	94.26	94.26	97.05	97.73	100.00	100.00	100.00
tot_info	94.39	98.79	98.79	98.81	100.00	100.00	100.00	100.00	100.00	100.00
space	55.90	57.62	58.12	59.18	59.43	59.85	60.67	65.26	78.74	79.66
flex	88.07	88.70	89.95	90.64	90.77	91.02	91.35	91.48	91.64	91.81
grep	56.87	59.94	60.39	65.89	67.09	72.23	76.62	90.13	96.00	100.00
sed	35.85	35.93	42.88	42.91	48.01	53.13	58.26	63.39	68.52	74.40
libtiff	96.75	96.75	97.11	97.11	97.11	97.11	98.19	98.19	98.19	100.00
python	76.50	76.50	76.50	76.50	76.50	76.50	76.50	76.50	76.50	76.58
gzip	96.67	96.67	96.67	98.33	98.33	100.00	100.00	100.00	100.00	100.00

original random samples ('Size Ratio') in each program. The last row ('Overall') shows the overall results of all programs. As shown in Table 12, PTD-TO obtains BETTER results over pure random approach in almost all faulty versions, demonstrating that PTD-TO significantly outperforms pure random approach. In case of the optimisation on all original random samples, for ER1′, GP03 and GP19, the optimised test suites performs similarly or better than the random samples in most faulty versions and in the overall results. For GP02, the optimised test suites do not obviously show an advantage over the random samples in most faulty versions and in the overall results. These results also confirm the previous analysis of GP02 in Section 3, i.e. GP02 has different intuitions from the general ones of SFL techniques. Our optimisation approach is designed from the general intuition of SFL and it is not surprising for it not to work well on GP02. In addition, the size ratio in Table 12 shows that PTD-TO significantly reduces the size of the original test suites, and the execution time and time ratio illustrate that the optimised test suites of PTD-TO consume considerably less execution time.

Based on all results above, we conclude that PTD-TO performs better than the random approach, and is effective to reduce the size of the original test suites and execution time by exhibiting a similar or better result than the original test suites, thus providing empirical support for our analysis in Section 3.3.

Please note that the main purpose of PTD-TO is to serve as a simple example illustrating the potential application of PTD, and offers empirical support for our analysis in Section 3.3. Currently, its algorithm is simple. Our future research will include further optimisation of PTD-TO (e.g. using concolic execution to generate 'useful' test cases), and the extension of the experiments with more programs and comparisons.

© The Institution of Engineering and Technology 2018

Table 11 Statistical comparison between original test suites and their similar ones

	Percentage	ER1′	GP02	GP03	GP19
1%	2-tailed	1.12×10^{-12}	3.24×10^{-27}	4.0527×10^{-302}	2.56×10^{-28}
	1-tailed(right)	5.60×10^{-13}	1.64×10^{-27}	2.0289×10^{-302}	1.28×10^{-28}
	1-tailed(left)	1.00	1.00	1.00	1.00
	conclusion	WORSE	WORSE	WORSE	WORSE
2%	2-tailed	8.33×10^{-5}	3.10×10^{-118}	1.02×10^{-4}	1.02×10^{-1}
	1-tailed(right)	7.45×10^{-5}	1.55×10^{-118}	8.68×10^{-4}	0.97
	1-tailed(left)	0.98	1.00	0.97	8.68×10^{-2}
	conclusion	WORSE	WORSE	WORSE	SIMILAR
3%	2-tailed	3.85×10^{-2}	3.96×10^{-4}	9.87×10^{-17}	6.79×10^{-2}
	1-tailed(right)	2.03×10^{-2}	1.99×10^{-4}	5.06×10^{-17}	0.98
	1-tailed(left)	0.98	1.00	1.00	5.02×10^{-2}
	conclusion	WORSE	WORSE	WORSE	SIMILAR
4%	2-tailed	1.76×10^{-52}	4.88×10^{-5}	4.31×10^{-2}	1.93×10^{-2}
	1-tailed(right)	8.81×10^{-53}	2.44×10^{-5}	2.95×10^{-2}	9.67×10^{-3}
	1-tailed(left)	1.00	1.00	0.99	0.99
	conclusion	WORSE	WORSE	WORSE	WORSE
5%	2-tailed	5.87×10^{-23}	3.76×10^{-2}	3.86×10^{-1}	8.33×10^{-2}
	1-tailed(right)	2.94×10^{-23}	1.88×10^{-2}	2.07×10^{-1}	0.98
	1-tailed(left)	1.00	0.98	8.27×10^{-1}	7.45×10^{-2}
	conclusion	WORSE	WORSE	WORSE	WORSE
6%	2-tailed	7.53×10^{-3}	8.15×10^{-3}	1.96×10^{-4}	8.73×10^{-1}
	1-tailed(right)	4.18×10^{-3}	5.37×10^{-3}	1.07×10^{-4}	4.35×10^{-1}
	1-tailed(left)	0.99	0.99	1.00	5.68×10^{-1}
	conclusion	WORSE	WORSE	WORSE	SIMILAR
7%	2-tailed	4.31×10^{-2}	1.41×10^{-4}	1.83×10^{-4}	1.81×10^{-1}
	1-tailed(right)	2.95×10^{-2}	7.58×10^{-5}	1.05×10^{-4}	0.96
	1-tailed(left)	0.98	1.00	1.00	1.92×10^{-1}
	conclusion	WORSE	WORSE	WORSE	SIMILAR
8%	2-tailed	4.88×10^{-2}	7.85×10^{-22}	1.83×10^{-4}	2.56×10^{-6}
	1-tailed(right)	4.45×10^{-2}	3.98×10^{-22}	1.05×10^{-4}	1.35×10^{-6}
	1-tailed(left)	0.98	1.00	1.00	1.00
	conclusion	WORSE	WORSE	WORSE	WORSE
9%	2-tailed	7.37×10^{-3}	1.78×10^{-152}	1.16×10^{-2}	4.55×10^{-2}
	1-tailed(right)	4.39×10^{-3}	8.88×10^{-153}	7.07×10^{-3}	3.59×10^{-2}
	1-tailed(left)	0.99	1.00	0.99	0.98
	conclusion	WORSE	WORSE	WORSE	WORSE
10%	2-tailed	1.43×10^{-2}	4.09×10^{-3}	1.80×10^{-5}	1.78×10^{-3}
	1-tailed(right)	9.83×10^{-3}	2.11×10^{-3}	9.63×10^{-6}	9.29×10^{-4}
	1-tailed(left)	0.99	0.99	1.00	0.99
	conclusion	WORSE	WORSE	WORSE	WORSE

5 Threats to validity

Threats to internal validity: Chances are that some implementation flaws exist in our experiment and could have affected the results. To assure the correctness of the implementation, we have carefully realised the implementation of the five SFL techniques and conducted adequate testing.

Threats to external validity: Our study assumes that faulty statements are executed when a failure happens. This assumption should generally hold, especially in the single-fault scenarios of our study. However, the assumption may not hold in some cases, such as in the context of multiple faults. For multiple faults, we can apply the clustering technology (e.g. [21]) to transform the context of multiple faults into that of single faults, and thus our approach can be applied to multiple faults. In addition, the recent research [22] has shown that multiple faults pose a negligible effect on the effectiveness of fault localisation despite the effect of fault localisation interference. These findings increase our confidence in the experimental results in the context of multiple faults. It would be worthwhile to incrementally extend our study to more applications to obtain additional insight on this issue.

Another threat is the subject programs. We choose seven small programs (a standard benchmark Siemens) with seeded faults and seven large real-life programs mostly with real-happening faults to obtain reliable experimental results because they are commonly used in debugging. Even so, there still exist many unknown and complicated factors in realistic debugging. Therefore, it would be worthwhile to use more programs (e.g. multiple-faults programs and large-sized programs) to further strengthen the experimental results.

Threats to construct validity: We use the rank of the faulty statement in the ranking list to evaluate the effectiveness of SFL techniques. This metric is highly recommended by the recent research [15] and thus the threat is acceptably mitigated.

© The Institution of Engineering and Technology 2018

Table 12 Optimisation of PTD-TO on all original random samples and its comparison on pure random approach

PTD-TO		Optimisation on all original random samples			Comparison of pure random approach			Execution time, s		Time ratio	Size ratio
		BETTER	SIMILAR	WORSE	BETTER	SIMILAR	WORSE	Optimised tests	Original tests		
print_token (2 ver.)	ER1′	10	4	1	15	0	0	22391.19	38433.32	58.26	52.82
	GP02	4	3	8	14	1	0				
	GP03	10	4	1	15	0	0				
	GP19	13	2	0	15	0	0				
replace	ER1′	19	6	5	30	0	0	30056.14	42820.52	70.19	66.76
	GP02	12	5	13	28	1	1				
	GP03	19	8	3	30	0	0				
	GP19	24	3	3	30	0	0				
schedule (2 ver.)	ER1′	10	7	1	18	0	0	37474.04	59501.66	62.98	59.54
	GP02	10	4	4	18	0	0				
	GP03	10	6	2	18	0	0				
	GP19	13	5	0	18	0	0				
tcas	ER1′	25	9	3	37	0	0	14158.30	18254.68	77.56	73.12
	GP02	26	0	11	36	1	0				
	GP03	21	11	5	37	0	0				
	GP19	27	10	0	37	0	0				
tot_info	ER1′	12	2	5	19	0	0	28927.07	39944.03	72.42	79.85
	GP02	10	0	9	17	2	0				
	GP03	12	3	4	19	0	0				
	GP19	12	3	4	19	0	0				
space	ER1′	27	5	3	35	0	0	122720.13	173062.70	70.91	74.35
	GP02	15	14	6	34	0	1				
	GP03	31	4	0	35	0	0				
	GP19	29	3	3	35	0	0				
flex	ER1′	36	11	6	52	1	0	367451.81	424608.27	86.54	91.97
	GP02	40	2	11	51	2	0				
	GP03	42	4	7	53	1	0				
	GP19	31	20	2	52	1	0				
grep	ER1′	10	10	1	21	0	0	323744.02	424459.30	76.27	73.84
	GP02	6	8	7	18	1	2				
	GP03	12	8	1	21	0	0				
	GP19	12	8	1	21	0	0				
sed	ER1′	12	15	2	29	0	0	139090.37	195085.52	71.30	79.73
	GP02	14	10	5	28	0	1				
	GP03	13	14	2	29	0	0				
	GP19	16	10	3	29	0	0				
libtiff	ER1′	7	4	1	12	0	0	64284.45	75947.76	84.64	81.21
	GP02	6	2	4	10	1	1				
	GP03	6	5	1	12	0	0				
	GP19	7	5	0	12	0	0				
python	ER1′	4	4	0	8	0	0	46015.52	68365.70	67.31	71.74
	GP02	4	1	3	6	0	2				
	GP03	4	3	1	8	0	0				
	GP19	5	3	0	8	0	0				
gzip	ER1′	3	2	0	4	0	1	13555.38	14995.24	90.40	92.83
	GP02	1	1	3	2	1	2				
	GP03	2	3	1	4	1	0				
	GP19	2	3	0	4	1	0				
overall	ER1′	175	79	28	280	1	1	1209868.41	1575478.72	76.79	74.81
	GP02	148	50	84	262	10	10				
	GP03	182	73	28	280	2	0				
	GP19	191	75	16	280	2	0				

6 Related work

This section surveys closely related work on test case generation and test suite reduction for fault localisation [More other work on fault localisation can refer to the surveys [11, 23].].

Test case generation: Baudry *et al.* [24] add new test cases to existing test cases by maximising the number of distinguished dynamic basic blocks. Artzi *et al.* [25] propose a concolic-execution based approach to generate new test cases that are similar to a given failing test case. Rößler *et al.* [26] utilise a failing test case and a set of facts correlated with executed program

© The Institution of Engineering and Technology 2018

```
if (e.hasMoreElements()) { //was: while(e.hasMoreElements())
… e.nextElement();…
}
if (e.hasMoreElements()) failure = true; // oracle added check
```

Note: the seeded fault is an **if** that replaced a **while** and that the condition for failure is set whenever the enumerator had more than one element in it.

Fig. 5 *Code snippet from Masri and Assi [59]*

entities and states to isolate failure causes. Wang and Roychoudhury [27] alter the outcome of the branches in a failing run to generate a feasible successful run and construct a bug report for debugging engineers according to the branch instances evaluated differently in the failing run and the successful run. Lei *et al.* [28] simulate the interactions between developers and fault localisation to iteratively generate new test cases for the existing test suite. An entropy-based test generation approach [29] applies probability theory concepts using entropy to generate new test cases for fault localisation. Xuan and Monperrus [30] generate purified versions of failing test cases, which include only one assertion per test and exclude unrelated statements of this assertion, for improving fault localisation. Abreu *et al.* [31] use a generation strategy of passing test cases in terms of the failing ratio. Artzi *et al.* [32] use the variation on combined concrete and symbolic execution to generate a test suite for locating the faults of the dynamic web application. Zhang *et al.* [33] generate new test cases by cloning failing test cases to improve fault localisation. González-Sanchez *et al.* [34] utilise the structural characteristics of the test coverage matrix to construct an analytic model, introducing coverage density and coverage distribution for guiding test case generation. In addition, González-Sanchez *et al.* [35] introduce a metric called ambiguity for a test suite to identify the same involvement pattern among test cases and generate new test cases to improve the ambiguity of a test suite for fault localisation. Perez *et al.* [36] recently summarise test cases generation approaches (e.g. [24, 34, 35]), and propose an integrated metric DDU by taking advantage of three metrics of density [34], diversity [37] and uniqueness [35], for the guidance of generating optimised tests for fault localisation. Inozemtseva and Holmes [38] conduct an empirical study showing that the coverage of a test suite is not strongly with the effectiveness of fault detection. Differently, our study focuses on the effect of test suites on the effectiveness of fault localisation rather than fault detection. Search-based test data generation [39, 40], as a typical application of search-based software engineering, has been a burgeoning interest for many researchers [41–43] and also shown its promising results on generating effective test suites for finding faults [44–46]. It is simple to adopt search-based test generation to different test data generation problem by simply changing the input space and fitness function, thus this approach is a natural candidate for utilising our findings in the future test data generation research on fault localisation. In contrast to these approaches above, our work focuses on investigating the impact of test suites on fault localisation to obtain a deeper understanding of the relation between test suites and fault localisation. We believe that our findings provide a new perspective on test case generation for fault localisation and can benefit existing and further research on this direction, such as utilising the promising search-based test data generation for obtaining a high-quality localisation test suite.

Test case reduction for fault localisation: Test case reduction for fault localisation can be roughly categorised into two types: one is to remove redundant test cases; the other one including our study is to remove harmful test cases. Yu *et al.* [47] investigate whether traditional test suite reduction strategies used in testing work well in fault localisation. Their results show that traditional test suite reduction strategies do not work well on fault localisation. They propose a vector-based reduction approach for fault localisation to reduce those redundant test cases with the same set of executed statements. Hao *et al.* [48] propose three reduction strategies to remove redundant test inputs for fault localisation. Their strategies select test cases from the given test collection based on the execution traces of the test collection to distinguish as many suspiciousness values as possible for fault localisation. Zhao *et al.* [49] define the notion of a coverage vector to obtain distinct execution paths and measure the relation of each coverage vector with the failing behaviours to reduce similar test cases. Gong *et al.* [50] introduce the concept of Diversity Speedup to order a set of unlabelled test cases and use this order to obtain a smaller number of test cases by delivering good localisation results. Bandyopadhyay [51] leverage difference set and union set of executed statements in passing test cases and failing ones, and use their ratio as a heuristic to reduce a test suite. Masri and Assi [52] use the Euclidean metric to measure the similarity between passing test cases and failing ones, and removing those similar passing ones from the current test suite. Genetic algorithms [53] are adopted to identify the relation between program statements and failures for exploring the reduction of test suites. Dandan *et al.* [54] present a two-step test suite reduction approach. Their approach not only uses test cases coverage to perform coverage matrix-based reduction but also analyses the concrete path information to conduct path vector based reduction. Perez *et al.* [36] pinpoint that the test-suite reduction based on the adequacy criteria (e.g. multi-object optimisation [55, 56] greedy-based reduction [57], and profile-based reduction [58]) could preserve localisation accuracy by incorporating diagnostic metrics. Differently, our approach utilises the coverage of `MSS` to optimise test suites, and our findings provide a theoretical and empirical explanation and support for their approach. Mutation-based fault localisation [9, 10] utilise mutation analysis to propose an evaluation metric for improving fault localisation, and our impact finding test cases are also applicable to the evaluation metric, e.g. Equation 1 in [9]. Masri and Assi [59] propose a technique named `Technique-I`. `Technique-I` is based on the concept of defect-based failures claiming that those passing test cases causing a defect-based failure are harmful to fault localisation, and tries to remove those passing test cases. As a reminder, the discussion on defect-based failures, in comparison to our findings, is at the next subsection of *Defect-based Failures*. They realise it by discarding those passing test cases whose execution covers a statement executed by all failing test cases but not so often by all passing test cases. `Technique-I` focuses on just a statement executed by all failing test case, and the execution frequency of this statement in all passing test cases. In contrast, our technique `PTD-TO` is based on our findings in Section 3 showing that those passing test cases covering the faulty statements are harmful to fault localisation, and tries to discard those passing test cases. In general, a failing test case implies that its executed statements should include a faulty statement. It means that the faulty statement should be included in those statements executed by all failing test cases. Thus, those passing test cases covering a high faction, rather than one, of those statements, are more probable to hit the faulty statement. We implement `PTD` by removing those passing test cases. On the contrary, a passing test case does not provide a definite conclusion whether its executed statements are free of a faulty statement, `PTD` does not use the uncertain information delivered by passing test cases. It also reveals that a statement executed not so often in all passing test cases does not necessarily mean that the statement has a strong correlation with the failures. However, `Technique-I` still uses this uncertain information as the clue for removing potentially harmful passing test cases. Furthermore, this heuristic of `PTD-TO` is supported by the data presented in Table 10 and empirically demonstrated to be effective.

Defect-based failures: For understanding the insight of the technique of Maris and Assi [59], we will discuss defect-based failures in comparison to our study. Masri and Assi [59] define a defect-based failure: the execution of a test case satisfying the values of the oracle added flags. Take the illustration used in their work as an example (see Fig. 5). The last line of code is inserted into the original program as an oracle added flag. If the execution of a test case satisfies the condition, it is called a defect-based failure. In contrast to defect-based failures, an output-based failure is determined by executing the original and seeded versions, and compares their respective outputs. If their respective outputs are different, a defect-based failure happens. The goal of introducing

© The Institution of Engineering and Technology 2018

defect-based failures is to exclude coincidentally correct test cases caused by using output-based failures. Their definition on defect-based failures classifies those passing test cases executing the faulty statement into two types: those causing defect-based failures are harmful for fault localisation; whereas the others are not harmful. However, our study has theoretically and empirically proved that the above two types of passing test cases are both harmful for fault localisation. As shown in Fig. 5, in case of the enumerator having one element in it, a passing test executing the faulty statement will not cause a defect-based failure. Nevertheless, this passing test is still harmful for coverage-based fault localisation, as all the suspiciousness metrics (including the one used in [59]) only take as input the coverage matrix. Furthermore, our study considers not only the effect of the above passing test cases, but also the impact of those passing test cases not executing the faulty statement.

In comparison with these approaches above, the main purpose of our test suite optimisation is to empirically validate our theoretical analysis and provide further insight on our findings. In addition, our study focuses on studying the impact of test suites on fault localisation to provide a deeper understanding of the impact of test suites on fault localisation, such as the insight missed by Masri and Assi [59].

7 Conclusion

This paper has presented an extensive empirical analysis to quantify and explain the impact of test suites on fault localisation. In particular, we have found that localisation effectiveness is not strongly correlated with test suite size. We have also identified the positive or negative impact of different parts of a test suite. That is, in a test suite, the passing test cases that do not execute the faulty statements and the failing test cases have a positive impact on the fault localisation effectiveness, whereas the passing test cases that exercise the faulty statements have a negative effect on localisation performance. Furthermore, we find that the `SFL` can obtain the maximal benefit from the failing test cases, and thus the fluctuations of the localisation performance are mainly caused by the passing test cases. Therefore, we introduced the *Passing Tests Discrimination* metric to shed light on the relationship between localisation effectiveness and a test suite, and demonstrated its potential with a simple but effective test suite optimisation approach.

We believe that our findings provide a new perspective on fault localisation and suggest interesting directions for future work. For example, it will be worthwhile to seek test prioritisation for fault localisation by utilising the commonality of tests with `MSS`, and also investigate a stronger case for the benefits of `PTD-TO` in practice. It will be also interesting to utilise our findings to improve test case generation and test suite optimisation for fault localisation, and extend our study to multiple-fault scenarios.

8 Acknowledgment

The authors thank the developers of ManyBugs (http://repairbenchmarks.cs.umass.edu/ManyBugs/) and SIR (http://sir.unl.edu/portal/index.php), based on which our experiments were built. This work was partially supported by the National Natural Science Foundation of China under grant nos. 61602504, 61672529, 61379054 and 61502015.

9 References

[1] Cleve, H., Zeller, A.: 'Locating causes of program failures'. Proc. the 27th Int. Conf. Software Engineering (ICSE 2005), 2005, pp. 342–351
[2] Jin, W., Orso, A.: 'F3: fault localization for field failures'. Proc. the 2013 Int. Symp. on Software Testing and Analysis (ISSTA 2013), 2013, pp. 213–223
[3] Wong, W.E., Debroy, V., Gao, R., *et al.*: 'The dstar method for effective software fault localization', *IEEE Trans. Reliab.*, 2014, **63**, (1), pp. 290–308
[4] Naish, L., Lee, H., Ramamohanarao, K.: 'A model for spectra-based software diagnosis', *Trans. Softw. Eng. Methodol. (TOSEM)*, 2011, **20**, (3), p. 11
[5] Lei, Y., Mao, X., Dai, Z., *et al.*: 'Effective statistical fault localization using program slices'. Proc. the 36th Annual Int. Computer Software and Applications Conf. (COMPSAC 2012), 2012, pp. 1–10
[6] Xie, X., Chen, T. Y., Kuo, F.-C., *et al.*: 'A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization', *Trans. Soft. Eng. Methodol. (TOSEM)*, 2013, **22**, (4), p. 31
[7] Sun, C., Khoo, S.-C.: 'Mining succinct predicated bug signatures'. Proc. the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013), 2013, pp. 576–586
[8] Zhang, L., Zhang, L., Khurshid, S.: 'Injecting mechanical faults to localize developer faults for evolving software'. Proc. the 2013 ACM SIGPLAN Int. Conf. Object Oriented Programming Systems Languages and Applications (OPPSLA 2013), 2013, pp. 765–784
[9] Moon, S., Kim, Y., Kim, M., *et al.*: 'Ask the mutants: mutating faulty programs for fault localization'. Proc. the 7th IEEE Int. Conf. Software Testing, Verification and Validation (ICST 2014), 2014, pp. 153–162
[10] Papadakis, M., Traon, Y.L.: 'Metallaxis-fl: mutation-based fault localization', *Softw. Test. Verif. Reliab.*, 2015, **25**, pp. 605–628
[11] Wong, W.E., Gao, R., Li, Y., *et al.*: 'A survey on software fault localization', *IEEE Trans. Softw. Eng.*, 2016, **42**, (8), p. 1
[12] Abreu, R., Zoeteweij, P., Van Gemund, A.: 'On the accuracy of spectrum-based fault localization'. Proc. the Testing: Academic and Industrial Conf. Practice and Research Techniques – MUTATION, 2007, pp. 89–98
[13] Xie, X., Kuo, F.-C., Chen, T.Y., *et al.*: 'Provably optimal and human-competitive results in SBSE for spectrum based fault localisation'. Proc. the 5th Symp. on Search-Based Software Engineering (SSBSE 2013), 2013, pp. 224–238
[14] Wong, W., Qi, Y., Zhao, L., *et al.*: 'Effective fault localization using code coverage'. Proc. the 31st Annual Int. Computer Software and Applications Conf. (COMPSAC 2007), 2007, pp. 449–456
[15] Parnin, C., Orso, A.: 'Are automated debugging techniques actually helping programmers?'. Proc. the 2011 Int. Symp. on Software Testing and Analysis (ISSTA 2011), 2011, pp. 199–209
[16] Xie, X., Wong, W.E., Chen, T.Y., *et al.*: 'Metamorphic slice: an application in spectrum-based fault localization', *Inf. Softw. Technol.*, 2013, **55**, (5), pp. 866–879
[17] McNaught, A.D., Wilkinson, A.: '*Compendium of chemical terminology*' (Blackwell Science, Oxford, 1997)
[18] Wohlinv, C., Runeson, P., Höst, M., *et al.*: '*Experimentation in software engineering*' (Springer, Berlin, 2012)
[19] Wang, X., Cheung, S., Chan, W., *et al.*: 'Taming coincidental correctness: coverage refinement with context patterns to improve fault localization'. Proc. the 31st Int. Conf. Software Engineering (ICSE 2009), 2009, pp. 45–55
[20] Corder, G.W., Foreman, D.I.: '*Nonparametric statistics for non-statisticians: a step-by-step approach*' (John Wiley & Sons, Hoboken, 2009)
[21] Jones, J.A., Bowring, J.F., Harrold, J.F.: 'Debugging in parallel'. Proc. the 2007 Int. Symp. on Software Testing and Analysis (ISSTA 2007), 2017, pp. 16–26
[22] DiGiuseppe, N., Jones, J.: 'On the influence of multiple faults on coverage-based fault localization'. Proc. the 2011 Int. Symp. on Software Testing and Analysis (ISSTA 2011), 2011, pp. 210–220
[23] Steinder, M., Sethi, A.S.: 'A survey of fault localization techniques in computer networks', *Sci. Comput. Program.*, 2004, **53**, (2), pp. 165–194
[24] Baudry, B., Fleurey, F., Le Traon, Y.: 'Improving test suites for efficient fault localization'. Proc. the 28th Int. Conf. Software Engineering (ICSE 2006), 2006, pp. 82–91
[25] Artzi, S., Dolby, J., Tip, F., *et al.*: 'Directed test generation for effective fault localization'. Proc. the 19th Int. Symp. on Software Testing and Analysis (ISSTA 2010), 2010, pp. 49–60
[26] Rößler, J., Fraser, G., Zeller, A., *et al.*: 'Isolating failure causes through test case generation'. Proc. the 2012 Int. Symp. on Software Testing and Analysis (ISSTA 2012), 2012, pp. 309–319
[27] Wang, T., Roychoudhury, A.: 'Automated path generation for software fault localization'. Proc. the 20th IEEE/ACM Int. Conf. Automated Software Engineering (ASE 2005), 2005, pp. 347–351
[28] Lei, Y., Mao, X., Dai, Z., *et al.*: 'Effective fault localization approach using feedback', *IEICE Trans. Inf. Syst.*, 2012, **95**, (9), pp. 2247–2257
[29] Campos, J., Abreu, R., Fraser, G., *et al.*: 'Entropy-based test generation for improved fault localization'. The 28th Int. Conf. Automated Software Engineering (ASE 2013), 2013, pp. 257–267
[30] Xuan, J., Monperrus, M.: 'Test case purification for improving fault localization'. Proc. the 22nd ACM SIGSOFT Int. Symp. on Foundations of Software Engineering (FSE 2014), 2014, pp. 52–63
[31] Abreu, R., Zoeteweij, P., Golsteijn, R., *et al.*: 'A practical evaluation of spectrum-based fault localization', *J. Syst. Softw.*, 2009, **82**, (11), pp. 1780–1792
[32] Artzi, S., Dolby, J., Tip, F., *et al.*: 'Fault localization for dynamic web applications', *IEEE Trans. Softw. Eng.*, 2012, **38**, (2), pp. 314–335
[33] Zhang, L., Yan, L., Zhang, Z., *et al.*: 'A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization', *J. Syst. Softw.*, 2017, **129**, pp. 35–57
[34] González-Sanchez, A., Gross, H., Van Gemund, A.J.C.: 'Modeling the diagnostic efficiency of regression test suites'. Proc. the 4th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2011), Workshop Proc., 2011, pp. 634–643
[35] González-Sanchez, A., Abreu, R., Gross, H., *et al.*: 'Prioritizing tests for fault localization through ambiguity group reduction'. Proc. the 26th IEEE/ACM Int. Conf. Automated Software Engineering (ASE 2011), 2011, pp. 83–92
[36] Perez, A., Abreu, R., Deursen, A.V.: 'A test-suite diagnosability metric for spectrum-based fault localization approaches'. Proc. the 39th Int. Conf. Software Engineering (ICSE 2017), 2017, pp. 654–664
[37] Jost, L.: 'Entropy and diversity', *Oikos*, 2006, **113**, (2), pp. 363–375
[38] Inozemtseva, L., Holmes, R.: 'Coverage is not strongly correlated with test suite effectiveness'. Proc. the 36th Int. Conf. Software Engineering (ICSE 2014), 2014, pp. 435–445

© The Institution of Engineering and Technology 2018

[39] Harman, M.: 'Automated test data generation using search based software engineering'. Proc. the 2nd Int. Workshop on Automation of Software Test (AST 2007), 2007, pp. 1–2
[40] McMinn, P.: 'Search-based software test data generation: a survey', *Softw.Test. Verif. Reliab.*, 2004, **14**, (2), pp. 105–156
[41] Zhang, Y., Harman, M., Jia, Y., *et al.*: 'Inferring test models from Kate's bug reports using multi-objective search'. Proc. the 7th Int. Conf. Search Based Software Engineering (SSBSE 2015), 2015, pp. 301–307
[42] Geronimo, L.D., Ferrucci, F., Murolo, A., *et al.*: 'A parallel genetic algorithm based on Hadoop MapReduce for the automatic generation of JUnit test suites'. Proc. the 5th Int. Conf. Software Testing, Verification and Validation (ICST 2012), 2012, pp. 785–793
[43] Zhang, X., Tallam, S., Gupta, N., *et al.*: 'Towards locating execution omission errors', *ACM SIGPLAN Notices*, 2007, **42**, (6), pp. 415–424
[44] Fraser, G., Staats, M., McMinn, P., *et al.*: 'Does automated unit test generation really help software testers? A controlled empirical study', *Trans. Softw. Eng. Methodol. (TOSEM)*, 2015, **24**, (4), p. 23
[45] Fraser, G., Arcuri, A.: '1600 faults in 100 projects: automatically finding faults while achieving high coverage with evosuite', *Empir. Softw. Eng.*, 2015, **20**, (3), pp. 611–639
[46] Arcuri, A., Fraser, G.: 'On the effectiveness of whole test suite generation'. Proc. the 6th Int. Conf. Search Based Software Engineering (SSBSE 2014), 2014, pp. 1–15
[47] Yu, Y., Jones, J.A., Harrold, M.J.: 'An empirical study of the effects of test-suite reduction on fault localization'. Proc. the 30th Int. Conf. Software Engineering (ICSE 2008), 2008, pp. 201–210
[48] Hao, D., Xie, T., Zhang, L., *et al.*: 'Test input reduction for result inspection to facilitate fault localization', *Autom. Softw. Eng.*, 2010, **17**, (1), pp. 5–31
[49] Zhao, L., Zhang, Z., Wang, L., *et al.*: 'PAFL: fault localization via noise reduction on coverage vector'. Proc. the 23rd Int. Conf. Software Engineering and Knowledge Engineering (SEKE 2011), 2011, pp. 203–206
[50] Gong, L., Lo, D., Jiang, L., *et al.*: 'Diversity maximization speedup for fault localization'. Proc. the 27th IEEE/ACM Int. Conf. Automated Software Engineering (ASE 2012), 2012, pp. 30–39
[51] Bandyopadhyay, A.: 'Mitigating the effect of coincidental correctness in spectrum based fault localization'. Proc. the 5th Int. Conf. Software Testing, Verification and Validation (ICST 2012), 2012, pp. 479–482
[52] Masri, W., Assi, R.A.: 'Prevalence of coincidental correctness and mitigation of its impact on fault localization', *ACM Trans. Softw. Eng. Methodol.*, 2014, **23**, (1), p. 8
[53] Masri, W., Abouassi, R., Elghali, M., *et al.*: 'An empirical study of the factors that reduce the effectiveness of coverage-based fault localization'. Proc. the 2009 Int. Workshop on Defects in Large Software Systems, 2009, pp. 1–5
[54] Dandan, G., Tiantian, W., Xiaohong, S., *et al.*: 'A test-suite reduction approach to improving fault-localization effectiveness', *Comput. Lang. Syst. Struct.*, 2013, **39**, (3), pp. 95–108
[55] Yoo, S., Harman, M.: 'Using hybrid algorithm for pareto efficient multi-objective test suite minimisation', *J. Syst. Softw.*, 2010, **83**, (4), pp. 689–701
[56] Alipour, M.A., Shi, A., Gopinath, R., *et al.*: 'Evaluating non-adequate test-case reduction'. Proc. the 31st IEEE/ACM Int. Conf. Automated Software Engineering (ASE 2016), 2016, pp. 16–26
[57] Lin, C.T., Tang, K.W., Wang, J.S., *et al.*: 'Empirically evaluating greedy-based test suite reduction methods at different levels of test suite complexity', *Sci. Comput. Program.*, 2017, **150**, pp. 1–25
[58] Wang, R., Qu, B., Lu, Y.: 'Empirical study of the effects of different profiles on regression test case reduction', *IET Softw.*, 2015, **9**, (2), pp. 29–38
[59] Masri, W., Assi, R.A.: 'Cleansing test suites from coincidental correctness to enhance fault-localization'. Proc. the 3rd Int. Conf. Software Testing, Verification and Validation (ICST 2010), 2010, pp. 165–174

© The Institution of Engineering and Technology 2018