Abstract Test Case Prioritization Using Repeated Small-Strength Level-Combination Coverage

Rubing Huang, *Member, IEEE*, Weifeng Sun, Tsong Yueh Chen, *Senior Member, IEEE*, Dave Towey, *Member, IEEE*, Jinfu Chen, *Member, IEEE*, Weiwen Zong, and Yunan Zhou

***Abstract*—Abstract test cases (ATCs) have been widely used in practice, including in combinatorial testing and in software product line testing. When constructing a set of ATCs, due to limited testing resources in practice (e.g., in regression testing), test case prioritization (TCP) has been proposed to improve the testing quality, aiming at ordering test cases to increase the speed with which faults are detected. One intuitive and extensively studied TCP technique for ATCs is λ-*wise Level-combination Coverage based Prioritization* (λLCP), a static, black-box prioritization technique that only uses the ATC information to guide the prioritization process. A challenge facing λLCP, however, is the necessity for the selection of the fixed prioritization strength λ before testing—testers need to choose an appropriate λ value before testing begins. Choosing higher λ values may improve the testing effectiveness of λLCP (e.g., by finding faults faster), but may reduce the testing efficiency (by incurring additional prioritization costs). Conversely, choosing lower λ values may improve the efficiency, but may also reduce the effectiveness. In this paper, we propose a new family of λLCP techniques, *Repeated Small-strength Level-combination Coverage-based Prioritization* (RSLCP), that repeatedly achieves the full combination coverage at lower strengths. RSLCP maintains λLCP's advantages of being static and black box, but avoids the challenge of prioritization strength selection. We have performed an empirical study involving five different versions of each of five C programs. Compared with λLCP, and *Incremental-strength LCP* (ILCP), our results show that RSLCP could provide a good trade-off between testing effectiveness and efficiency. Our results also show that RSLCP is more effective and efficient than two popular techniques of *Similarity-based Prioritization* (SP). In addition, the results of empirical studies also show that RSLCP can remain robust over multiple system releases.**

***Index Terms*—Abstract test case, level-combination coverage, regression testing, software testing, test case prioritization.**

I. Introduction

In PRACTICE, software systems are usually influenced by different *parameters* or *factors* (such as configuration options and user inputs), with each parameter possibly having a finite set of different *levels* or *values*. An abstract test case (ATC) represents a combination of levels of different parameters, and has been used in different testing situations, including combinatorial testing [1], software product lines testing [2], and highly configurable systems testing [3].

When an ATC set has been constructed, it is desirable to execute all the test cases—in which case execution order does not matter. However, due to often limited testing resources, it is often possible to only run some of the ATCs in the set. In such situations, the ATC execution order may become critical, because a well-prioritized test case execution sequence may identify failures more quickly, and thus may enable earlier fault characterization, diagnosis, and correction [1]. Generally speaking, the process of scheduling the order of test cases is called *Test Case Prioritization* (TCP) [4], and the prioritization of ATCs is called *Abstract Test Case Prioritization* (ATCP) [5].

Many strategies have been proposed to guide ATCP according to different criteria, for example, *random TCP* [6], [7], and *Similarity-based Prioritization* (SP) [8]–[10]. The most widely used ATCP is λ-*wise Level-combination Coverage-based Prioritization* (λLCP) [11], which adopts a fixed strength λ (called the *prioritization strength*) to choose each ATC in a greedy manner: When selecting each next ATC from the candidates, λLCP calculates the number of parameter-level combinations at a fixed prioritization strength λ covered by each candidate that has not yet been covered by executed test cases, and then chooses the one with the maximum number of uncovered λ-wise parameter-level combinations. λLCP has many advantages, including that it is simple and intuitive [11]. Furthermore, because it only uses the level-combination coverage information derived from the test cases, rather than information from the source code or program execution, λLCP is a static, black-box technique [12].

Manuscript received April 16, 2018; revised November 16, 2018; accepted March 22, 2019. Date of publication May 3, 2019; date of current version March 2, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61502205, Grant 61872167, Grant 71471092, and Grant U1836116, in part by the Senior Personnel Scientific Research Foundation of Jiangsu University under Grant 14JDG039, in part by the Ningbo Science and Technology Bureau under Grant 2014A35006, and in part by the Young Backbone Teacher Cultivation Project of Jiangsu University. The work of D. Towey was supported by the Artificial Intelligence and Optimisation Research Group of the University of Nottingham Ningbo China, the International Doctoral Innovation Centre, the Ningbo Education Bureau, the Ningbo Science and Technology Bureau, and the University of Nottingham. Associate Editor: J. Zhang. *(Corresponding author: Rubing Huang.)*

R. Huang is with the School of Computer Science and Communication Engineering, and also with Jiangsu Key Laboratory of Security Technology for Industrial Cyberspace, Jiangsu University, Zhenjiang, Jiangsu 212013, China (e-mail: rbhuang@ujs.edu.cn).

W. Sun, J. Chen, W. Zong, and Y. Zhou are with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, Jiangsu 212013, China (e-mail: 3140608036@stmail.ujs.edu.cn; jinfuchen@ujs.edu.cn; vevanzong@ujs.edu.cn; zhouyn@ujs.edu.cn).

T. Y. Chen is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Hawthorn, VIC 3122, Australia (e-mail: tychen@swin.edu.au).

D. Towey is with the School of Computer Science, University of Nottingham Ningbo China, Ningbo, Zhejiang 315100, China (e-mail: dave.towey@nottingham.edu.cn).

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

Digital Object Identifier 10.1109/TR.2019.2908068

Although previous studies have shown that λLCP is an effective prioritization technique, in terms of fault detection [6], [11], [13], [14], it does have a constraint that the fixed strength λ must be set before prioritization begins. Different λ values may lead to different performances, with investigations [13], [14] finding that larger λ values may improve the testing effectiveness of λLCP (e.g., by finding faults faster), but may reduce the testing efficiency (by incurring additional prioritization costs); conversely, lower λ values may improve the efficiency, but may also reduce the effectiveness.

Although it is intuitive that a small prioritization strength for λLCP may be efficient (in terms of overheads), it has also been shown that over 50% of faults can be triggered by one parameter (1-wise combination coverage), and more than 70% can be triggered by two (2-wise combination coverage) [15], [16]. This indicates that choosing a small prioritization strength may be effective for λLCP, especially when the number of ATCs is small. However, when the ATC candidate set is large, λLCP with small prioritization strengths may become ineffective [17]. This is because, when the small-strength (1-wise or 2-wise) level-combination coverage is fully achieved by the selected or executed ATCs, the remaining candidates are effectively randomly ordered.

In this paper we propose a new family of λLCP techniques, *Repeated Small-strength Level-combination Coverage-based Prioritization* (RSLCP). RSLCP attempts to overcome the limitations of current versions of LCP, attempting to better balance the tradeoff between testing effectiveness and efficiency. In particular, RSLCP begins with a small prioritization strength λ $(\lambda = 1, 2)$ to implement the λLCP algorithm—which means that RSLCP is also initially λLCP. Once the λ-wise level-combination coverage of selected or executed ATCs is fully achieved—i.e., the number of λ-wise level combinations covered by the selected ATCs is equal to that covered by all candidates—then RSLCP restarts with the same prioritization strength λ, repeating full λ-wise level-combination coverage in the next round. This process is repeated until all candidates have been chosen. RSLCP has the following three advantages: 1) it is very simple, adopting a similar mechanism to λLCP; 2) similar to λLCP, it is a static, black-box prioritization method, using only the level-combination coverage to guide the ATCP (this means that it is not necessary to obtain source code information, nor to execute the program); and 3) unlike λLCP, it is not necessary to set the prioritization strength before prioritizing ATCs.

To evaluate the proposed technique, we conducted empirical studies on five C programs, each of which had five different versions. In summary, the main contributions of this paper are as follows:

1) We propose a new strategy to guide the ATC prioritization, *repeated small-strength level-combination coverage-based prioritization* (RSLCP), and describe a framework to support it.
2) Based on the proposed framework, we provide two categories (using two different strategies) involving six algorithms to implement RSLCP.
3) We report on empirical studies investigating each RSLCP technique, comparing with λLCP and SP, from the perspectives of: testing effectiveness (the speed of interaction coverage and fault detection); testing efficiency (the prioritization cost); and robustness [how well the overall fault detection potential is maintained across different versions of the software under test (SUT)].

The rest of this paper is organized as follows. Section II describes the background information. Section III introduces the RSLCP method, including the framework, algorithm, complexity analysis, and mechanism. Section IV presents the research questions and experimental setup. Section V reports on the empirical studies conducted to answer the research questions. Section VI reviews related work, and, finally, Section VII concludes the paper.

II. Background

In this section, we describe some background information about ATCs and TCP.

A. Abstract Test Case

Given some SUT that has k parameters that constitute a parameter set $P = \{p_1, p_2, \ldots, p_k\}$, with a corresponding level set $L = \{L_1, L_2, \ldots, L_k\}$, where each parameter p_i has some valid levels from the finite set L_i $(i = 1, 2, \ldots, k)$. In practice, parameters may represent anything that influences the performance of the SUT, such as components, configuration options, user inputs, and so on. Let $\mathcal{Q}$ be the set of constraints on level combinations.

Definition 2.1. Input Parameter Model: An input parameter model (or input model) for the SUT, denoted as $Model(P, L, \mathcal{Q})$, is a model of the SUT that includes the set of some parameters P that may influence the SUT, the set of level sets L for each parameter, and constraints set $\mathcal{Q}$ on level combinations.

For example, Fig. 1 shows a screenshot of the font settings for Microsoft Word 2013.[1] As shown in the red box, we only consider the Effects aspects of the font settings, for which there are seven choices. Table I gives an input parameter model for the font effects of Microsoft Word 2013. The effects have seven parameters, each of which can have two levels. It is not possible to have both parameters from any of the sets (**Strikethrough**, **Double Strikethrough**), (**Superscript**, **Subscript**), or (**Small caps**, **All caps**) to be "`Yes`" at the same time. Therefore, there are three level combination constraints. To simplify the representation of this problem, each parameter can be denoted by p_i $(i = 1, 2, \ldots, 7)$, and each level can be labelled by an integer (starting at 0), as shown in Table I.

This example yields the following input parameter model: $Model$ $(P = \{p_1, p_2, \ldots, p_7\},\ L = \{\{\text{“0”}, \text{“1”}\},\ \{\text{“2”}, \text{“3”}\}, \{\text{“4”}, \text{“5”}\}, \{\text{“6”}, \text{“7”}\}, \{\text{“8”}, \text{“9”}\}, \{\text{“10”}, \text{“11”}\},\ \{\text{“12”}, \text{“13”}\}\}, \mathcal{Q} = \{p_1 = \text{“0”} \leftrightarrow p_2 = \text{“3”}, p_3 = \text{“4”} \leftrightarrow p_4 = \text{“7”}, p_5 = \text{“8”} \leftrightarrow p_6 = \text{“11”}\})$, where the symbol $\leftrightarrow$ represents implication. Because the specific values of each parameter have no impact on

[1] [Online]. Available: https://products.office.com/en-us/microsoft-word-2013.

TABLE I
INPUT PARAMETER MODEL FOR MICROSOFT WORD 2013 FONT EFFECTS

Parameter	p_1: `Strikethrough`	p_2: `Double Strikethrough`	p_3: `Superscript`	p_4: `Subscript`	p_5: `Small caps`	p_6: `All caps`	p_7: `Hidden`
Level	Yes (0)	Yes (2)	Yes (4)	Yes (6)	Yes (8)	Yes (10)	Yes (12)
	No (1)	No (3)	No (5)	No (7)	No (9)	No (11)	No (13)

(`Strikethrough` = "Yes") ↔ (`Double Strikethrough` = "No"), i.e., (p_1 = "0") ↔ (p_2 = "3").
(`Superscript` = "Yes") ↔ (`Subscript` = "No"), i.e., (p_3 = "4") ↔ (p_4 = "7").
(`Small caps` = "Yes") ↔ (`All caps` = "No"), i.e., (p_5 = "8") ↔ (p_6 = "11").

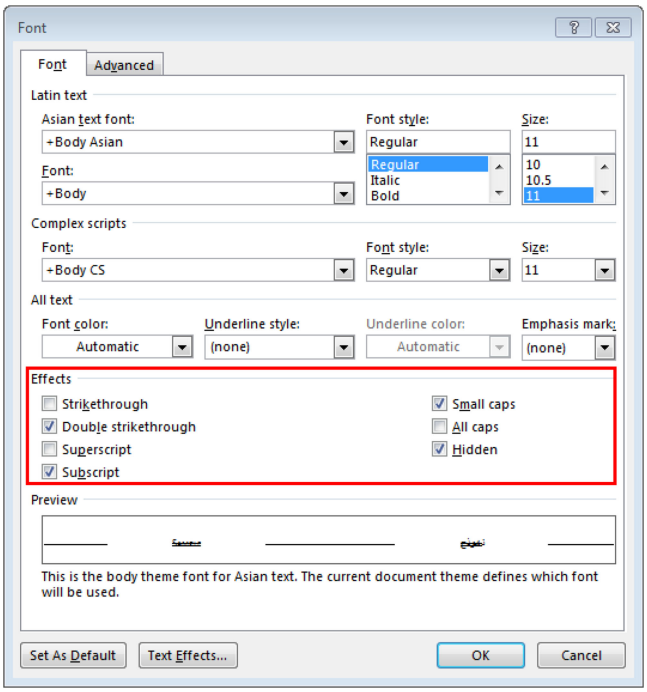


Fig. 1. Screenshot of the font settings from Microsoft Word 2013.

the SUT model, without loss of generality, we can use the following abbreviated version: $Model(|L_1||L_2|\cdots|L_k|, \mathcal{Q})$. Therefore, the above model can be represented as: $Model(2^7, \mathcal{Q} = \{\text{"0"} \leftrightarrow \text{"3"}, \text{"4"} \leftrightarrow \text{"7"}, \text{"8"} \leftrightarrow \text{"11"}\})$.

Definition 2.2. Abstract Test Case: A k-tuple $(l_1, l_2, \ldots, l_k)$ is an ATC of the SUT where $l_i \in L_i, 1 \leq i \leq k$.

If all the level constraints in $\mathcal{Q}$ are satisfied, then the ATC is said to be *valid*, otherwise it is *invalid*. An example of a valid ATC for the previous model is ("0", "3", "4", "7", "8", "11", "12"); and an example of an invalid one is ("0", "2", "5", "6", "9", "10", "13")—because it violates the constraint ("0" ↔ "3").

Definition 2.3: η-wise Level Combination: An η-wise level combination is a k-tuple $(\widehat{l}_1, \widehat{l}_2, \ldots, \widehat{l}_k)$ involving η parameters with fixed levels (called fixed parameters) and $(k - \eta)$ parameters with arbitrary allowable levels (called free parameters that are denoted by "–"), where $0 \leq \eta \leq k$ and:

$$\widehat{l}_i = \begin{cases} l_i \in L_i, & \text{if } p_i \text{ is a fixed parameter} \\ -, & \text{if } p_i \text{ is a free parameter} \end{cases}. \quad (1)$$

An η-wise level combination is also called an *η-wise schema* [1]. Without loss of generality, to more clearly describe the problem, free parameters can be ignored. In other words, an η-wise level combination can be considered an η-tuple. Intuitively speaking, any ATC can cover some η-wise level combinations: for example, an ATC ("1", "2", "4", "7", "8", "11", "13") covers seven 1-wise level combinations ("1"), ("2"), ("4"), ("7"), ("8"), ("11"), and ("13"). Similar to ATCs, an η-wise level combination may also be either valid or invalid: for example, a 2-wise level combination ("1", "2") is valid; but another one, ("0", "2"), is invalid. Obviously, a valid ATC covers all valid η-wise level combinations, regardless of η values.

For ease of description, we define a function $\psi(\eta, tc)$ for an ATC tc that returns the set of all η-wise level combinations covered by tc, i.e.

$$\psi(\eta, tc) = \{(v_{j_1}, v_{j_2}, \ldots, v_{j_\eta}) | 1 \leq j_1 < j_2 < \cdots < j_\eta \leq k\}. \quad (2)$$

Similarly, a function $\psi(\eta, T)$ for a set T of test cases can be defined to return the set of all η-wise level combinations covered by all model inputs in T, i.e.

$$\psi(\eta, T) = \bigcup_{tc \in T} \psi(\eta, tc). \quad (3)$$

Obviously, the size of $\psi(\eta, tc)$ ($|\psi(\eta, tc)|$) is equal to $C(k, \eta)$ (the number of η-combinations from k elements).

We next present the definition of η-wise level-combination coverage for an ATC, or for a subset of the given test set.

Definition 2.4: η-wise Level-Combination Coverage: Given a valid test suite T, a valid ATC tc, and a subset T' of T ($tc \in T$ and $T' \subseteq T$), the η-wise level-combination coverage of tc against T can be defined as the ratio of the number of η-wise level combinations covered by tc to those covered by T: $\frac{|\psi(\eta, tc)|}{|\psi(\eta, T)|}$. The η-wise level-combination coverage of test set T' against T can be written as $\frac{|\psi(\eta, T')|}{|\psi(\eta, T)|}$.

B. Test Case Prioritization

TCP seeks to schedule test cases such that those with higher priority, according to some criteria, are executed earlier than those with lower priority. When testing resources are limited or insufficient for the execution of all test cases in a test suite, a well-designed test case execution order can be crucial. The problem of TCP is defined as follows [4]:

Definition 2.5: Test Case Prioritization: Given a tuple (T, Ω, g), where T is a test suite, Ω is the set of all possible permutations of T, and g is a fitness function from Ω to real numbers, the goal of TCP is to find a prioritized test suite (also called a test sequence) $S \in \Omega$ such that

$$(\forall S')\ (S' \in \Omega)\ (S' \neq S)\ [g(S) \geq g(S')]. \quad (4)$$

According to Rothermel *et al.* [4], prioritization can be done according to many possible criteria, including, for example,

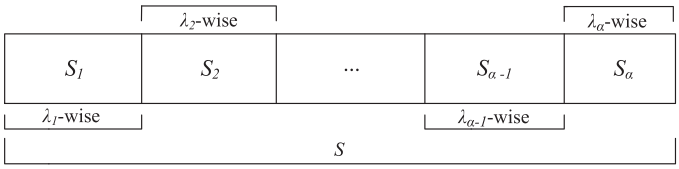


Fig. 2. Illustration of RSLCP.

code coverage [18]. To date, many TCP strategies have been proposed, based on various concepts, including: fault severity [19]; source code coverage [4], [20], [21]; search-based techniques [18]; integer linear programming [22]; risk exposure [23]; historical records from recent regression tests [24]; and information retrieval [25], [26]. Most strategies can be classified as either meta-heuristic search methods or greedy methods [27]. When TCP is applied to ATCs, it is called *abstract TCP* (ATCP) [5].

III. Repeated Small-Strength Level-Combination Coverage-Based Prioritization

In this section, we present a new family of λLCP techniques that work by repeatedly using repeated, small-strength level-combination coverage. We call these techniques RSLCP. We introduce two RSLCP versions in this section, and present an analysis of the space and time complexity for each version.

A. Framework

Unlike λLCP, because the RSLCP prioritization strength is limited to 1 or 2, it is not necessary that a value be assigned to λ before prioritizing ATCs.

As shown in Fig. 2, RSLCP prioritizes an unordered set of ATCs (denoted T) into a prioritized set S that has been divided into α ($\alpha \geq 1$) disjoint and ordered parts $\langle S_1, S_2, \ldots, S_\alpha \rangle$, where each S_i ($i = 1, 2, \ldots, \alpha$) has also been prioritized using a prioritization strength λ_i. Formally, the following five conditions must be satisfied:

$$\left.\begin{array}{l} 1)\text{Each } S_i \text{ is a non-empty test sequence}, 1 \leq i \leq \alpha \\ 2)\ T = S_1 \bigcup S_2 \bigcup \cdots \bigcup S_\alpha \\ 3)\ S = \langle S_1, S_2, \ldots, S_\alpha \rangle \\ 4)\ S_i \bigcap S_j = \emptyset, 1 \leq i \neq j \leq \alpha \\ 5)\ \lambda_i \text{ is used for the construction of } S_i \end{array}\right\}. \quad (5)$$

Condition 1 means each subset S_i is both non-empty and ordered. Condition 2 means that all test cases are divided amongst the α subsets. Condition 3 means that S is ordered by sequencing $S_1, S_2, \ldots, S_\alpha$ successively [which means that S_{j+1} follows S_j ($1 \leq j < \alpha$)]. According to Condition 4, no test case belongs to more than one test sequence, and, finally, Condition 5 means that S_i is constructed using the prioritization strength λ_i.

Although the value of α is fully determined by the given T, it does not impact on the framework or on the following algorithms. We created two versions of the framework, an *independent* and a *partially independent* version.

1) RSLCP Independent Version: The RSLCP independent version (RSLCP-IV) guarantees that construction of S_{i+1} is independent of construction of S_i ($1 \leq i < \alpha$). Formally, the following two conditions must be satisfied:

$$\left.\begin{array}{l} 1)\ \lambda_l \in \{1, 2\}, 1 \leq l \leq \alpha \\ 2)\ \psi(\lambda_l, S_l) = \psi(\lambda_l, T \setminus \bigcup_{i=1}^{l-1} S_i) \end{array}\right\}. \quad (6)$$

Condition 1 means that each subset S_i adopts a small strength (1 or 2) to guide the λLCP process. Condition 2 means that each subset S_i covers all λ_i-wise level combinations that could be covered by the candidates remaining before constructing S_i.

Although the construction of test sequences S_i and S_{i+1} ($1 \leq i < \alpha$) are independent, actually, construction of S_i may impact on the construction of S_{i+1} (because S_{i+1} is constructed using only those test cases remaining after S_i's construction). The algorithms used to prioritize each test sequence S_i will be presented in Section III-B.

2) RSLCP Partially Independent Version: The RSLCP partially independent version (RSLCP-PV) is similar to the independent version, but involves some S_{i+1} constructions that are based on the S_i construction. The following three conditions must be satisfied (assuming $S_0 = \emptyset$):

$$\left.\begin{array}{l} 1)\ \lambda_{2x-1} = 1, \lambda_{2x} = 2, 1 \leq x \leq \lceil \frac{\alpha}{2} \rceil \\ 2)\ \psi(\lambda_{2x-1}, S_{2x-1}) = \psi(\lambda_{2x-1}, T \setminus \bigcup_{i=1}^{2x-2} S_i) \\ 3)\ \psi(\lambda_{2x}, S_{2x-1} \bigcup S_{2x}) = \psi(\lambda_{2x}, T \setminus \bigcup_{i=1}^{2x-2} S_i) \end{array}\right\}. \quad (7)$$

where x is an integer.

Condition 1 differs from that of RSLCP-IV by assigning a prioritization strength of 1 to each S_i when i is an odd number, and a strength of 2 when i is even. Conditions 2 and 3 mean that, when i is odd, the corresponding test sequence S_i is constructed independently to achieve the highest 1-wise level-combination coverage; but when i is even, the S_i is constructed so as to guarantee that S_i and S_{i-1} cover the same 2-wise level combinations as those covered by the remaining candidates. In effect, RSLCP-PV first uses a prioritization strength of 1 to construct the subset S_{2x-1}, and then considers S_{2x-1} as the already selected ATCs for construction of the test sequence S_{2x}. This process is then repeatedly applied to the remaining candidates.

B. Algorithm

Algorithm 1 describes the basic RSLCP procedure, which includes iteratively constructing each S_i ($i = 1, 2, \ldots, \alpha$) (Line 4). Once an S_i is completely constructed, it is added to the end of S ($S \leftarrow S \succ S_i$) (Line 5), and removed from the candidate set T' (Line 6). The test sequence S_{i+1} can then be constructed, with such constructions continuing until all candidates have been chosen. Clearly, although construction of the test sequence S_i is independent of construction of S_j ($1 \leq i \neq j \leq \alpha$), as can be seen, because S_j is constructed using elements from candidates remaining after S_i's construction, the construction of S_i can impact that of S_j.

We propose an algorithm to complete the construction process for each S_i. The algorithm draws from the well-known greedy approach, *Additional Greedy Approach* [18], which iteratively selects the element of maximum weight (for the problem) from those parts not yet selected or executed. The problem for construction of S_i is to cover the maximum number of λ_i-wise

Algorithm 1: RSLCP Procedure.

Input: $T = \{tc_1, tc_2, \ldots, tc_n\}$ ▷ Unordered ATCs
Output: S ▷ Prioritized ATCs
1: $i \leftarrow 1$
2: $T' \leftarrow T$
3: **while** $|S| \neq n$ **do**
4: *Construct S_i by selecting elements from the remaining candidates T' as subsequent ATCs in S, according to a specified criterion and $\lambda_i \in \{1, 2\}$, i.e., Algorithm2(T', λ_i) or Algorithm3(T', λ_i).*
5: $S \leftarrow (S \succ S_i)$ ▷ Add S_i into the end of S
6: $T' \leftarrow (T' \setminus S_i)$
7: $i \leftarrow (i + 1)$
8: **end while**
9: **return** S

Algorithm 2: RSLCP-IV S_i Construction (T', λ_i).

Input: $T' \subseteq T$ ▷ Remaining candidates from T
Output: S_i ▷ Prioritized ATCs
1: $S_i \leftarrow \langle\rangle$
2: $TempSet \leftarrow \psi(\lambda_i, T')$
3: **while** $T' \neq \emptyset$ && $\psi(\lambda_i, S_i) \neq TempSet$ **do**
4: *Select $tc \in T'$, where $max\big(\big|\psi(\lambda_i, tc) \bigcup \psi(\lambda_i, S_i)\big|\big)$* ▷ Take a random one in case of equality
5: $S_i \leftarrow (S_i \succ \langle tc\rangle)$
6: $T' \leftarrow (T' \setminus \{tc_i\})$
7: **end while**
8: **return** S_i

level combinations not yet covered by test cases that have already been selected or executed. Algorithm 2 describes the Additional Greedy algorithm to construct S_i for RSLCP-IV, and Algorithm 3 describes it for RSLCP-PV.

1) RSLCP-IV Algorithm: As shown in Algorithm 2, the RSLCP-IV algorithm chooses one of the candidates as the next ATC in S_i such that it covers the maximum number of λ_i-wise level combinations that have not yet been covered by the already selected or executed ATCs in S_i (Line 5). If more than one candidate has the highest λ_i-wise level-combination coverage, then a *random tie-breaking* mechanism [28] is used, so that one best candidate is selected. This process is repeated until either of the following two conditions is satisfied (Line 3): 1) all candidates have been selected (i.e., $T' = \emptyset$) or 2) S_i achieves full λ_i-wise level-combination coverage against T' [i.e., $\psi(\lambda_i, S_i) = TempSet$, where $TempSet$ is the set of λ_i-wise level combinations covered by the remaining ATCs after completely constructing S_{i-1}].

For each prioritization strength λ_i $(1 \leq i \leq \alpha)$ used for constructing S_i, we use the following five assignment categories:

1) Pure 1-wise RSLCP-IV: Each prioritization strength λ_i is assigned a value of 1: $\lambda_1 = \lambda_2 = \cdots = \lambda_\alpha = 1$.
2) Pure 2-wise RSLCP-IV: Similar to the Pure 1-wise RSLCP-IV, this category assigns each prioritization strength λ_i a value of 2: $\lambda_1 = \lambda_2 = \cdots = \lambda_\alpha = 2$.
3) $(1 + 2)$-wise RSLCP-IV: Unlike the previous two assignment categories, this category uses a combination of 1 and 2 for the prioritization strengths. For S_i where i is an odd number, the prioritization strength λ_i is assigned a value of 1; and when i is an even number, λ_i is assigned a value of 2: $\lambda_1 = \lambda_3 = \cdots = \lambda_{2\lceil\frac{\alpha}{2}\rceil-1} = 1$; and $\lambda_2 = \lambda_4 = \cdots = \lambda_{\lfloor\frac{\alpha}{2}\rfloor} = 2$.
4) $(2 + 1)$-wise RSLCP-IV: This category inverts the $(1 + 2)$-wise RSLCP-IV category. For S_i with even i numbers, λ_i is assigned a value of 1; S_i with odd i numbers is assigned 2: $\lambda_1 = \lambda_3 = \cdots = \lambda_{2\lceil\frac{\alpha}{2}\rceil-1} = 2$; and $\lambda_2 = \lambda_4 = \cdots = \lambda_{\lfloor\frac{\alpha}{2}\rfloor} = 1$.
5) Random Assignment RSLCP-IV: In this category, each prioritization strength λ_i is randomly assigned either a 1 or 2 value: $\lambda_i = rand(1, 2)$, where $rand(x, y)$ is a function returning an integer in the range $[x, y]$.

Algorithm 3: RSLCP-PV S_i Construction(T', λ_i).

Input: $T' \subseteq T$ ▷ Remaining candidates from T
Output: S_i ▷ Prioritized ATCs
1: $S_i \leftarrow \langle\rangle$
2: **if** $\lambda_i == 1$ **then then**
3: $S' \leftarrow \emptyset$
4: **else if** ▷ For the case of $\lambda_i = 2$ **then**
5: $S' \leftarrow S_{i-1}$
6: **end if**
7: $TempSet \leftarrow \psi(\lambda_i, T' \bigcup S')$
8: **while** $T' \neq \emptyset$ && $\psi(\lambda_i, S_i \bigcup S') \neq TempSet$ **do**
9: *Select $tc \in T'$, where $max\big(\big|\psi(\lambda_i, tc) \bigcup \psi(\lambda_i, S_i \bigcup S')\big|\big)$* ▷ Take a random one in case of equality
10: $S_i \leftarrow (S_i \succ \langle tc\rangle)$
11: $T' \leftarrow (T' \setminus \{tc_i\})$
12: **end while**
13: **return** S_i

2) RSLCP-PV Algorithm: The RSLCP-PV algorithm (Algorithm 3) is similar to the $(1 + 2)$-wise RSLCP-IV algorithm. Construction of S_i with odd values of i (S_{2x-1}, $1 \leq x \leq \alpha/2$) uses the same mechanism as the RSLCP-IV algorithm (a prioritization strength of 1), indicating that this part is independent of previous constructions (Line 3). However, when constructing S_i for even values of i (S_{2x}), although the same prioritization strength of 2 is used, this part is partially *dependent* (not completely independent): information about the λ_{2x-1}-wise level combinations covered by ATCs in S_{2x-1} is used (Line 5). Random tie-breaking [28] is again used when there is more than one candidate covering the same maximum level of combinations.

The RSLCP-PV algorithm first uses a prioritization strength of 1 to prioritize ATCs. When 1-wise level-combination coverage has been fully achieved for S_{2x-1}, then a value of 2 is used for the prioritization strength. Effectively, the RSLCP-PV algorithm uses incremental prioritization strengths (from 1 to 2) to construct S_i.

C. Complexity Analysis

In this section, we provide a brief analysis of both the space and time complexity of RSLCP. We first introduce the data

structure used to store the λ_i-wise level combinations. Given $Model(|L_1||L_2|\cdots|L_k|, Q)$ and ATC set T with size n, we assume that $\delta = max_{1\le i\le k}\{|L_i|\}$.

A 2-layer hierarchical data structure, denoted H_{all}, is used to store all λ_i-wise level combinations derived from the input parameter model. The first layer of H_{all} is an array of $C(k,\lambda_i)$ elements, each of which is a parameter combination with size λ_i, denoted $FC_{\lambda_i} = (p_{j_1}, p_{j_2}, \ldots, p_{j_{\lambda_i}})$, where $1 \le j_1 < j_2 < \cdots < j_{\lambda_i} \le k$. In other words, this array contains all possible λ_i-wise parameter combinations. Each parameter combination in the first level is actually a pointer to the next layer. Each structure in the second layer is a bitmap for all λ_i-wise level combinations derived from each λ_i-wise parameter combination. Each bitmap uses a single bit for each λ_i-wise level combination, with a value of 1 indicting that the relevant level combination has already been covered by previously selected ATCs, but a value of 0 meaning that it has not yet been covered.

For each candidate $tc \in T$, we use an array H_{each} of size $C(k,\lambda_i)$, each element of which represents the index of the λ_i-wise level combination of the corresponding FC_{λ_i} in the second level of H_{all}. To check whether each λ_i-wise level combination is covered or not, its index can be used to locate the relevant position in the bitmap.

1) Space Complexity: We next present an analysis of the space complexity of RSLCP, which is determined by two parameters: 1) the number of candidates, n and 2) the number of η-wise ($\eta \in \{1,2\}$) level combinations derived from the input parameter model.

Because each candidate covers η-wise level combinations of size $C(k,\eta)$, the space complexity for parameter (1) is $O\big(n \times C(k,\eta)\big)$. The space complexity for parameter (2) is determined by the input parameter model. As described in the previous section, the data structure used to store the possible η-wise level combinations, H_{all}, has two layers. The first layer of H_{all} contains all η-wise parameter combinations, resulting in a space complexity of $O_1 = O\big(C(k,\eta)\big)$. The space complexity of the second layer, O_2, can be described as follows:

$$\begin{aligned} O_2 &= O\left(\sum_{1\le j_1<j_2<\cdots<j_\eta\le k}\big(|L_{j_1}||L_{j_2}|\cdots|L_{j_{\rho_i}}|\big)\right) \\ &< O\left(\sum_{1\le j_1<j_2<\cdots<j_\eta\le k}\big(\delta^\eta\big)\right) \\ &= O\big(C(k,\eta)\times\delta^\eta\big). \end{aligned} \tag{8}$$

Therefore, the RSLCP space complexity is

$$\begin{aligned} O(\text{RSLCP}) &= O\big(n\times C(k,\eta)\big) + O_1 + O_2 \\ &< O\big(n\times C(k,\eta)\big) + O\big(C(k,\eta)\big) \\ &\quad + O\big(C(k,\eta)\times\delta^\eta\big) \\ &= O\big(C(k,\eta)\times\big(n+1+\delta^\eta\big)\big) \\ &= O\big(C(k,\eta)\times\big(n+\delta^\eta\big)\big). \end{aligned} \tag{9}$$

Because η is limited to a value of either 1 or 2, the best space complexity is when $\eta = 1$, giving $O\big(C(k,1)\times(n+\delta)\big)$, which is of the same order as $O\big(k\times(n+\delta)\big)$. The worst space complexity, when $\eta = 2$, is $O\big(C(k,2)\times\big(n+\delta^2\big)\big)$, which is of the same order as $O\big(k^2\times\big(n+\delta^2\big)\big)$. Of the different versions of RSLCP, only Pure 1-wise RSLCP-IV has the best space complexity.

2) Time Complexity: We next present an analysis of the time complexity of RSLCP, which is also determined by two parameters: 1) the number of candidates involved, n and 2) the time complexity of calculating uncovered η-wise level combinations for each candidate.

Regarding parameter (1), when selecting the ith model input from candidates, RSLCP needs to check each of the $(n-i+1)$ candidates. For parameter (2), there is a need to check whether or not the η-wise level combinations covered by each candidate tc are covered by previously selected ATCs. Since H_{each} stores the index of each η-wise level combination, this check takes $O(1)$ time for each η-wise level combination. Therefore, the RSLCP time complexity can be presented as

$$\begin{aligned} O(\text{RSLCP}) &= O\left(\sum_{i=1}^{n}\big((n-i+1)\times C(k,\eta)\big)\right) \\ &= O\left(\left(\sum_{i=1}^{n}(n-i+1)\right)\times C(k,\eta)\right) \\ &= O\left(\frac{n(n+1)}{2}\times C(k,\eta)\right) \\ &= O\big(n^2\times C(k,\eta)\big). \end{aligned} \tag{10}$$

Similar to the results of the space complexity analysis, RSLCP has best time complexity ($O\big(n^2\times k\big)$) when $\eta = 1$, and worst complexity ($O\big(n^2\times k^2\big)$) when $\eta = 2$. Again, of the different RSLCP versions, only Pure 1-wise RSLCP-IV has the best time complexity.

Previous investigations [27], [29] have shown that the order of time complexity of λLCP is equal to $O\big(n^2\times C(k,\lambda)\big)$. This means that when $1 \le \lambda \le \lceil k/2\rceil$, then as λ increases, the prioritization time of λLCP also generally increases; however, when $\lceil k/2\rceil < \lambda \le k$, then the prioritization time generally decreases as λ increases. As discussed by Petke *et al.* [13], [14], λ is generally assigned a value between 1 and 6, which means that λ is generally less than $\lceil k/2\rceil$, especially when k is large. Since λLCP's order of time complexity is $O\big(n^2\times C(k,\eta)\big)$ (where η is equal to 1 or 2), it is expected that RSLCP would have similar testing efficiency to λLCP when λ is 1 or 2. However, RSLCP should be more efficient than λLCP when λ is 3, 4, 5, or 6.

D. Discussion

This section briefly explains why RSLCP should achieve improvements over λLCP. RSLCP attempts to provide a trade-off between testing effectiveness and efficiency for prioritizing ATCs. The analysis of time complexity showed that the testing efficiency of RSLCP should be similar or better than λLCP, which means that RSLCP is an efficient ATCP technique. The rest of this analysis, therefore, addresses how RSLCP should provide comparable testing effectiveness, comparing RSLCP with λLCP for different λ values.

1) When $1 \leq \lambda \leq 2$: As discussed earlier (Section III-A), RSLCP uses either 1 or 2 when applying λLCP to the prioritization of ATCs, which means that it would cover 1-wise or 2-wise level combinations as quickly as λLCP. This means that RSLCP should have testing effectiveness that is at least similar to λLCP. Furthermore, when 1-wise or 2-wise level combinations have been fully covered by the already selected ATCs, λLCP then randomly prioritizes the remaining ATCs. However, RSLCP repeats the λLCP process to prioritize any remaining ATCs, which should provide better performance than random prioritization (e.g., in terms of the speed of covering higher-strength level combinations).
2) When $3 \leq \lambda \leq 6$: RSLCP should be faster at covering 1-wise or 2-wise level combinations than λLCP, because this is the basic principle of RSLCP. Compared with λLCP, RSLCP may, however, be slower at covering high-strength level combinations. However, because RSLCP repeatedly achieves full 1-wise or 2-wise interaction coverage, it may also be able to quickly (to some extent) cover high-strength level combinations. For example, if a candidate ATC tc covers a set of 1-wise level combinations that have not been covered by previously selected ATCs, tc may also cover a set of λ-wise level combinations. In other words, RSLCP may sometimes provide comparable testing effectiveness to λLCP, when λ is high.

IV. Experimental Setup

In this section, we present the research questions related to the testing effectiveness and efficiency of our proposed techniques, and examine the experiments we conducted to answer them.

A. Research Questions

In the field of TCP, two important issues are: 1) the prioritization effectiveness and 2) the prioritization efficiency. Generally speaking, the prioritization effectiveness is measured by the rate of fault detection. However, due to the characteristics of ATCs, the prioritization effectiveness can be also measured by the rate of interaction coverage. In this paper, therefore, we focus on the rates of interaction coverage and fault detection with respect to the effectiveness. Furthermore, when a new version of the SUT is released, the original prioritized test suite may become less effective: the initial test ordering might no longer be optimal. It would be helpful, therefore, for testers to know how maintainable the fault detection potential (the *robustness*) of a test suite prioritization technique is over multiple releases of the system. The following four research questions were designed to examine the testing effectiveness, prioritization costs, and robustness of RSLCP.

RQ1: How well do the six RSLCP versions perform?

RQ1.1: How well do the five RSLCP-IV algorithms perform?

RQ1.2: How well does the RSLCP-PV algorithm compare with the RSLCP-IV algorithms?

Answering **RQ1** will help testers know which RSLCP technique is the most effective or efficient. The two sub-questions are designed to further investigate the best RSLCP-IV algorithms and the differences between the RSLCP-IV and RSLCP-PV algorithms.

RQ2: How well does RSLCP compare with λLCP?

As discussed, RSLCP attempts to balance the tradeoff between testing effectiveness and efficiency in λLCP. Answering **RQ2** should make it clear whether or not RSLCP can achieve comparable testing effectiveness to current λLCP techniques, which would help clarify whether or not it should be considered as a cost-effective alternative.

RQ3: How does RSLCP compare with other widely used prioritization techniques such as *Incremental-strength LCP* (ILCP), and *Similarity-based Prioritization* (SP)?

The ILCP is another ATCP technique to avoid the selection of prioritization strength existed in λLCP; while the SP has been considered as an efficient prioritization technique. Therefore, answer **RQ3** would enable a better understanding of the testing effectiveness and efficiency of RSLCP (compared with those of ILCP and SP), which would help decide whether it is more cost-effective or not.

RQ4: How robust is RSLCP across multiple releases of the SUT?

Answering **RQ4** will help identify the robustness of RSLCP, and whether or not it degrades over multiple releases of the system.

B. Subject Programs

In our empirical study, we considered five versions of five programs (giving a total of 25 different programs) written in the C programming language. The five programs, which were obtained from the GNU FTP server,[2] were: a tool for lexical analysis (*flex*); two widely used command-line tools for searching and processing text matching regular expressions (*grep* and *sed*); a widely used compression utility (*gzip*); and a popular utility used to control the compile and build processes of the programs (*make*).

These subject programs have been widely used in TCP research [4], [6], [13], [14], [29], [32]–[36]. Table II gives the program details, including the input parameter model,[3] the number of ATCs obtained from the Software-artifact Infrastructure Repository (SIR)[4] [37], the program size excluding comments in lines of code (measured by `cloc`[5]), the program version number, and the number of faults in each version. The ATC set for each program was constructed using the test specification language [38]. Apart from the program *make* (for which some ATCs were removed due to unsuccessful execution), the ATCs used cover all valid level combinations at each strength.

C. The 15 Studied Prioritization Techniques

Table III gives an overview of the 15 prioritization techniques investigated, listing each technique's category, mnemonic,

[2][Online]. Available: http://ftp.gnu.org/

[3]The input parameter model of each program was taken from the previous work by Petke *et al.* [13], [14].

[4][Online]. Available: http://sir.unl.edu/

[5][Online]. Available: http://cloc.sourceforge.net/

TABLE II
SUBJECT PROGRAMS

Subject	ATCs	Input Parameter Model	Size	Version	Faults
flex-v1			9 470	2.4.7	32
flex-v2		*Model*($2^6 3^2 5^1$, $\mathcal{Q}$)	12 231	2.5.1	32
flex-v3	500	$\lvert\mathcal{Q}\rvert = 12$	12 249	2.5.2	20
flex-v4			12 379	2.5.3	33
flex-v5			12 366	2.5.4	32
grep-v1			11 988	2.2	56
grep-v2		*Model*($2^1 3^3 4^2 5^1 6^1 8^1$, $\mathcal{Q}$)	12 724	2.3	58
grep-v3	440	$\lvert\mathcal{Q}\rvert = 83$	12 826	2.4	54
grep-v4			20 838	2.5	58
grep-v5			58 344	2.7	59
gzip-v1			4 521	1.1.2	8
gzip-v2		*Model*($2^{13} 3^1$, $\mathcal{Q}$)	5 048	1.2.2	8
gzip-v3	156	$\lvert\mathcal{Q}\rvert = 61$	5 059	1.2.3	7
gzip-v4			5 178	1.2.4	7
gzip-v5			5 682	1.3	7
make-v1			18 568	3.76.1	37
make-v2		*Model*(2^{10}, $\mathcal{Q}$)	19 663	3.77	29
make-v3	111	$\lvert\mathcal{Q}\rvert = 1$	20 461	3.78.1	28
make-v4			23 125	3.79	29
make-v5			23 400	3.80	28
sed-v1			7 793	3.0.2	16
sed-v2		*Model*($2^7 3^1 4^1 6^1 10^1$, $\mathcal{Q}$)	18 545	4.0.6	18
sed-v3	324	$\lvert\mathcal{Q}\rvert = 50$	18 687	4.0.8	18
sed-v4			21 743	4.1.1	19
sed-v5			26 466	4.2	22

description, prioritization objective, and corresponding reference in the literature. Because RSLCP is a new version of LCP, we also considered another λLCP version, denoted λW, which we investigated for six λ values ($\lambda = 1, 2, 3, 4, 5, 6$), following previous studies [6], [11], [14], [29]–[31]. In addition, we also compared our methods with another two widely used ATCP techniques, ILCP [29], and SP [8]. ILCP makes use of incremental strengths beginning with $\lambda = 1$ to run λLCP. We examined two versions of SBP [8], *global SP* (GSP), and *local SP* (LSP): GSP initially selects two elements as the first two test cases with the minimum similarity, and then iteratively chooses an element as the next test case such that it has the minimum Jaccard similarity against previously selected test cases; LSP, in contrast, iteratively chooses a pair of test cases with the minimum Jaccard similarity until all candidates have been chosen [8].

D. Fault Seeding

For each of the subject programs, the original version contains no seeded-in faults. Although a number of hand-seeded faults are available from the SIR [37], many of these faults are easily detected (on average more than 60% of test cases can reveal them). In this paper, therefore, we used mutation analysis [39] to seed in faults (see Table II). As discussed in previous studies [40], [41], mutation analysis can provide more realistic faults than hand-seeding, and may be more appropriate for studying TCP. Compared with real faults, however, the correlation between mutant killing ability and real fault detection may become weak when the test suite size is kept constant [42]. Nevertheless, the detection of real faults should improve significantly when test suites attain the highest levels of mutant kills [42]. In this paper, each mutant could be killed by the given test suite.

For the five subject programs, we used the same mutation faults[6] as used by Henard *et al.* [35]. More specifically, for each version V_i $(1 \leq i \leq 5)$ of each subject program, the same mutant operators used in Andrews *et al.* [40] were adopted to produce the faulty versions (mutants) for our paper. The operators used were: constant replacement; statement deletion; unary insertion; arithmetic operator replacement; relational operator replacement; logical operator replacement; and bitwise logical operator replacement. As discussed by Henard *et al.* [35], *equivalent*,[7] and *duplicated*[8] mutants were eliminated using the Trivial Compiler Equivalence (TCE) [44] tool, resulting in about one third of the mutants being removed. This was done to reduce interference in the fault detection evaluation of each prioritization technique. Furthermore, as suggested by Papadakis *et al.* [45], *subsumed* mutants [46] (also called *disjoint* mutants [47])[9] were also identified and discarded [35] to avoid biasing the experimental results [43]. The subsumed mutants were removed by executing all ATCs for each mutant, and identifying the failure-causing ATCs.

The performance of several ATCP techniques may depend on the *Failure-Triggering Fault Interaction* (FTFI) value of each mutant in each subject program, i.e., the number of parameters required to detect a failure [15], [48]. Table IV shows the FTFI number distribution of each program.

E. Evaluation Metrics

In this paper, we focused on the testing effectiveness and efficiency of RSLCP, from the perspectives of interaction coverage, fault detection, and prioritization cost.

1) Interaction Coverage Metric: The rate of interaction coverage was used to evaluate the speed of covering level combinations by the prioritized test suite. The *Average Percentage of τ-wise Covering-array Coverage* (APCC) [14], also called *Average Percentage of Combinatorial Coverage* [27], was used to measure the rate of interaction coverage of strength τ achieved by prioritized ATCs. Its definition is given as follows:

Definition 4.1: Average Percentage of τ-wise Covering-array Coverage: Suppose $S = \langle t_1, t_2, \ldots, t_n \rangle$ is a prioritized set of ATCs with size n, the APCC definition of S at strength τ $(1 \leq \tau \leq k)$ is

$$\mathrm{APCC}(\tau, S) = \frac{\sum_{i=1}^{n} \left|\psi(\tau, \bigcup_{j=1}^{i} \{t_j\})\right|}{n \times |\psi(\tau, S)|} - \frac{1}{2n}. \tag{11}$$

The APCC metric values range from 0.0 to 1.0, with higher values indicating better rates of interaction coverage at a specific strength τ. In this paper, following previous studies [14], we considered APCC with $\tau = 1$, 2, 3, 4, 5, and 6.

2) Fault Detection Metric: We used the fault detection rates of each prioritization technique as the fault detection metric.

[6][Online]. Available: https://henard.net/research/regression/ICSE_2016/mutants/

[7]Equivalent mutants are functionally equivalent versions of the original program [43].

[8]Duplicated mutants are equivalent to other mutants, but not to the original program [44].

[9]The mutants are subsumed or disjoint such that they are jointly killed when other mutants are killed.

TABLE III
RSLCP, λLCP, ILCP, AND SBP TECHNIQUES CONSIDERED IN THE EXPERIMENTS

Category	Mnemonic	Description	Prioritization Objective	Reference
RSLCP	IV1	Pure 1-wise RSLCP-IV	Covers the repeated maximum 1-wise level combinations	Our study, and [17]
	IV2	Pure 2-wise RSLCP-IV	Covers the repeated maximum 2-wise level combinations	Our study
	IV3	(1 + 2)-wise RSLCP-IV	Covers the independently-repeated maximum (1 + 2)-wise level combinations	Our study
	IV4	(2 + 1)-wise RSLCP-IV	Covers the independently-repeated maximum (2 + 1)-wise level combinations	Our study
	IV5	Random Assignment RSLCP-IV	Covers the independently-repeated maximum (1 or 2)-wise level combinations	Our study
	PV	RSLCP-PV	Covers the partially-independent-repeated maximum level combinations	Our study
λLCP	1W	LCP at prioritization strength 1	Covers the maximum 1-wise level combinations	[30]
	2W	LCP at prioritization strength 2	Covers the maximum 2-wise level combinations	[11]
	3W	LCP at prioritization strength 3	Covers the maximum 3-wise level combinations	[11]
	4W	LCP at prioritization strength 4	Covers the maximum 4-wise level combinations	[29]
	5W	LCP at prioritization strength 5	Covers the maximum 5-wise level combinations	[31]
	6W	LCP at prioritization strength 6	Covers the maximum 6-wise level combinations	[14]
ILCP	ILCP	Incremental-strength LCP	Covers the maximum level combinations at incremental strengths	[29]
SP	GSP	Global SP	Achieves global maximum distance	[8]
	LSP	Local SP	Achieves local maximum distance	[8]

TABLE IV
FTFI NUMBER DISTRIBUTION

Subject Program	FTFI Number							Total Number
	1	2	3	4	5	6	>6	
flex-v1	2	7	7	4	2	7	3	32
flex-v2	2	7	9	4	1	6	3	32
flex-v3	2	4	6	3	0	4	1	20
flex-v4	2	7	9	3	3	6	3	33
flex-v5	2	8	7	4	2	6	3	32
$\sum$	10	33	38	18	8	29	13	149
grep-v1	2	20	17	8	9	0	0	56
grep-v2	2	19	17	8	11	0	1	58
grep-v3	1	21	14	9	9	0	0	54
grep-v4	1	18	15	16	8	0	0	58
grep-v5	2	16	18	14	9	0	0	59
$\sum$	8	94	81	55	46	0	1	285
gzip-v1	4	2	1	0	1	0	0	8
gzip-v2	4	2	1	0	1	0	0	8
gzip-v3	5	2	0	0	0	0	0	7
gzip-v4	5	2	0	0	0	0	0	7
gzip-v5	5	2	0	0	0	0	0	7
$\sum$	23	10	2	0	2	0	0	37
make-v1	0	0	0	0	1	1	35	37
make-v2	0	0	0	0	0	1	28	29
make-v3	0	0	0	0	2	0	26	28
make-v4	0	0	0	0	1	1	27	29
make-v5	0	0	0	0	1	1	26	28
$\sum$	0	0	0	0	5	4	142	151
sed-v1	0	8	5	3	0	0	0	16
sed-v2	1	9	7	1	0	0	0	18
sed-v3	1	9	7	1	0	0	0	18
sed-v4	1	8	8	2	0	0	0	19
sed-v5	1	12	7	2	0	0	0	22
$\sum$	4	46	34	9	0	0	0	93

A well-known fitness function is the *Average Percentage of Faults Detected* (APFD) [4], which measures the fault detection rate of a given prioritized test suite. Higher APFD values indicate better prioritized test sequences. The APFD is defined as follows:

Definition 4.2: Average Percentage of Faults Detected: Suppose T is a test suite containing n test cases, and F is a set of m faults revealed by T. Let SF_i be the number of test cases in the prioritized test suite S of T that are executed before detecting fault f_i. The APFD of S is calculated using the following equation (from Rothermel *et al.* [4]):

$$\text{APFD}(S) = 1 - \frac{\text{SF}_1 + \text{SF}_2 + \cdots + \text{SF}_m}{n \times m} + \frac{1}{2n}. \qquad (12)$$

3) Efficiency Metric: The prioritization cost measures how quickly each prioritized test suite is constructed, and was used to represent the efficiency of the technique. Obviously, lower prioritization cost means better efficiency.

F. Inferential Statistical Analysis

Because some prioritization strategies involve randomization (due to the random tie-breaking technique [28]), we ran each experiment 1000 times, as suggested in previous studies [49].

As part of the investigation, we wanted to determine the statistical significance of any differences between the APCC or APFD values (used to evaluate each prioritization technique), for which there are many statistical tests, such as the t-test and Wilcoxon–Mann–Whitney test [49]. Because there was no relationship among the 1000 iterations, we used an unpaired test [35]. Furthermore, because no assumptions were made about which prioritization technique was better than the other, a two-tailed test was used [35]. Following previous guidelines on inferential statistical approaches for dealing with randomized algorithms [49], [50], we used the unpaired two-tailed Wilcoxon–Mann–Whitney test to check the statistical significance (at a significance level of 5%).

Since we used multiple statistical prioritization techniques, we report the p-values, which indicate whether or not the differences between two techniques are highly significant. When the p-value between two techniques M_1 and M_2 is less than 5%, the difference between M_1 and M_2 is highly significant; otherwise, it is not significant. As Henard *et al.* [35] explained, however, with an increase of the number of the executions, p will become sufficiently small, which means that there are differences between two algorithms. However, when the p-value is very small, it may be difficult to identify which algorithm is actually the better. We therefore used a different statistical measure, the effect size, which is generally measured by the non-parametric Varia and Delay effect size measure [51], $\hat{\text{A}}_{12}$.

The Varia and Delay effect size measure should provide more useful information when comparing two different algorithms. $\hat{\text{A}}_{12}(M_1, M_2) = 0.50$, for example, would indicate that in the sample, there is no difference between algorithms M_1 and M_2. $\hat{\text{A}}_{12}(M_1, M_2) > 0.50$ would mean that M_1 is superior to M_2; and $\hat{\text{A}}_{12}(M_1, M_2) < 0.50$ would mean that M_2 is superior to M_1. The further the $\hat{\text{A}}_{12}$ value is from 0.50, the larger is the effect size. Based on previous work [51], we classify four categories of the effect size: no-difference ($|\hat{\text{A}}_{12}(M_1, M_2) - 0.50| = 0$);

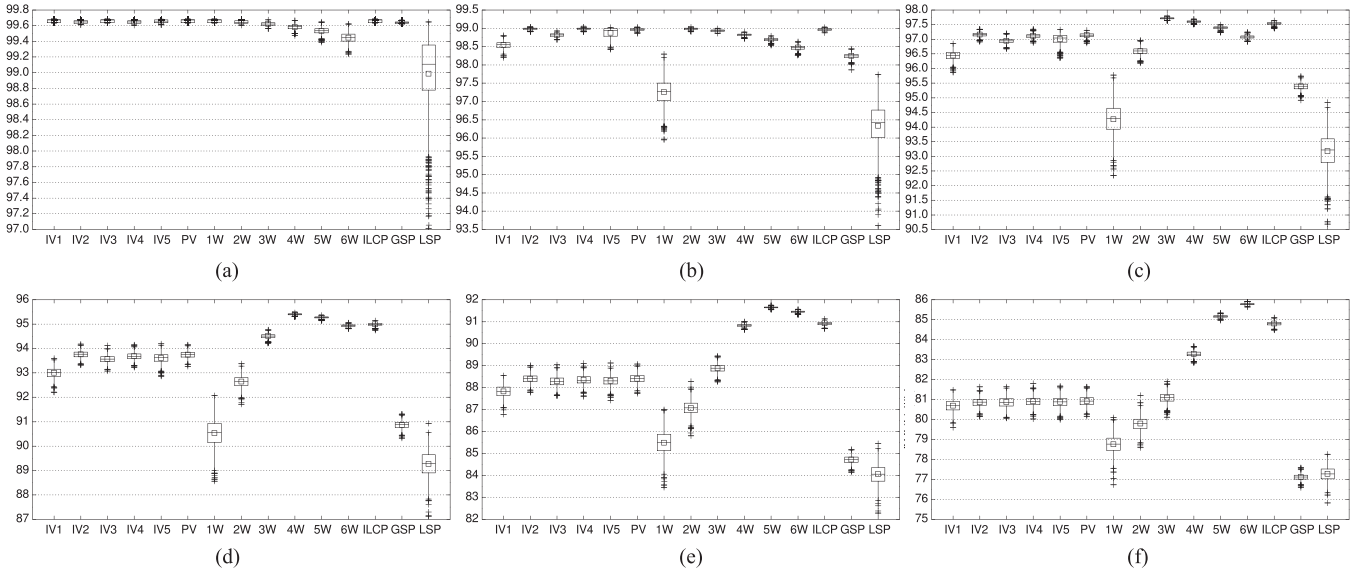


Fig. 3. APCC results for each prioritization technique for the program *flex*. (a) $\tau = 1$. (b) $\tau = 2$. (c) $\tau = 3$. (d) $\tau = 4$. (e) $\tau = 5$. (f) $\tau = 6$.

small ($0 < |\hat{A}_{12}(M_1, M_2) - 0.50| \leq 0.10$); medium ($0.10 < |\hat{A}_{12}(M_1, M_2) - 0.50| \leq 0.17$); and large ($|\hat{A}_{12}(M_1, M_2) - 0.50| > 0.17$).

V. Results

This section presents the results of the experiments conducted, and answers the research questions. In the displayed results, each box plot shows the distribution of the 1000 APCCs or APFDs (averaged over 1000 iterations), listed horizontally across the figure. Each box plot shows the mean (square in the box), median (line in the box), upper and lower quartiles, and minimum and maximum APCC values for the prioritization technique. In addition, a statistical analysis is given for each pairwise APCC or APFD comparison of prioritization techniques. For example, for a comparison between two methods M_1 versus M_2, we use $\times$ to denote that there is no statistical difference between them (i.e., their p-value is greater than 0.05); ✔ to denote that M_1 is significantly better (p-value is less than 0.05, and the effect size $\hat{A}_{12}(M_1, M_2)$ is greater than 0.50); and ✗ to denote that M_2 is significantly better (p-value is less than 0.05, and $\hat{A}_{12}(M_1, M_2)$ is less than 0.50).

A. Interaction Coverage Results

In this section, we answer **RQ1**, **RQ2**, and **RQ3**, from the perspective of the interaction coverage rates. Figs. 3–7 present the APCC results for programs *flex*, *grep*, *gzip*, *make*, and *sed*. Each figure describes different strength values for APCC, i.e., τ is assigned 1, 2, 3, 4, 5, and 6. Tables V and VI show the detailed Wilcoxon test APCC results at the 0.05 significance level for each comparison.

1) RQ1: RSLCP Techniques: Here, we try to answer the subquestions of **RQ1**: **RQ1.1** and **RQ1.2**, according to APCC, and then briefly analyze each observation.

a) **RQ1.1***: RSLCP-IV Techniques:* Based on the experimental results, we can observe the following.

i) When $\tau = 1$, all RSLCP-IV techniques have very similar APCCs for all programs, because their 1-wise APCCs have very similar distributions. According to the statistical analysis (Table V), however, IV1 and IV3 generally have the best performances, followed by IV5, regardless of subject programs (apart from programs *gzip* and *make*).

ii) When $2 \leq \tau \leq 6$, it can be observed that IV2 overall has the best performance for all programs, followed by IV4; while IV1 is worst, followed by IV5 and IV3. The statistical analysis (Table V) also confirms these observations.

The main reason for the first observation is that both IV1 and IV3 initially make use of $\lambda = 1$ for prioritizing ATCs, which is the same mechanism as 1W—the mechanism 1W chooses an element as the next test case such that it covers the largest number of 1-wise level combination that have not been covered by previously selected ATCs. When all 1-wise level combinations have been covered by the selected ATCs, the order of the remaining ATCs does not change the 1-wise APCC value. Therefore, IV1 and IV3 perform very similarly, and have better 1-wise APCCs than other RSLCP-IV techniques. Regarding the difference for programs *gzip* and *make*, a possible reason may be that an element covering the largest number of uncovered 2-wise level combinations (selected as the next test case by IV2, IV4, and IV5), could cover a comparable number of uncovered 1-wise level combinations as IV1 and IV3, due to the characteristics of the input parameter model (i.e., each parameter contains a similar number of levels).

The second observation can be explained as follows: Similar to the case of IV1 and IV3, IV2 and IV4 have the same APCC values at $\tau = 2$. However, IV2 repeatedly covers entire 2-wise level combinations, which may provide the faster speed to cover level combinations $\tau > 2$ than other RSLCP-IV techniques. Similarly, IV1 only repeatedly covers entire 1-wise level combinations, which may not provide higher rates of interaction coverage at higher strengths.

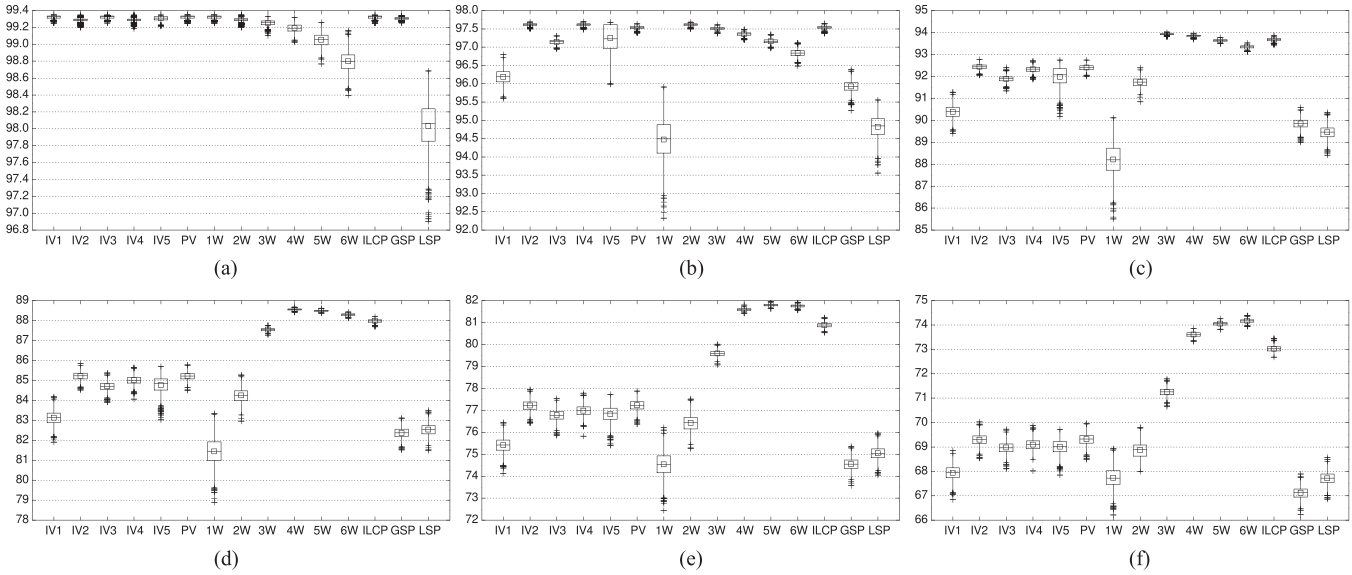


Fig. 4. APCC results for each prioritization technique for the program *grep*. (a) $\tau = 1$. (b) $\tau = 2$. (c) $\tau = 3$. (d) $\tau = 4$. (e) $\tau = 5$. (f) $\tau = 6$.

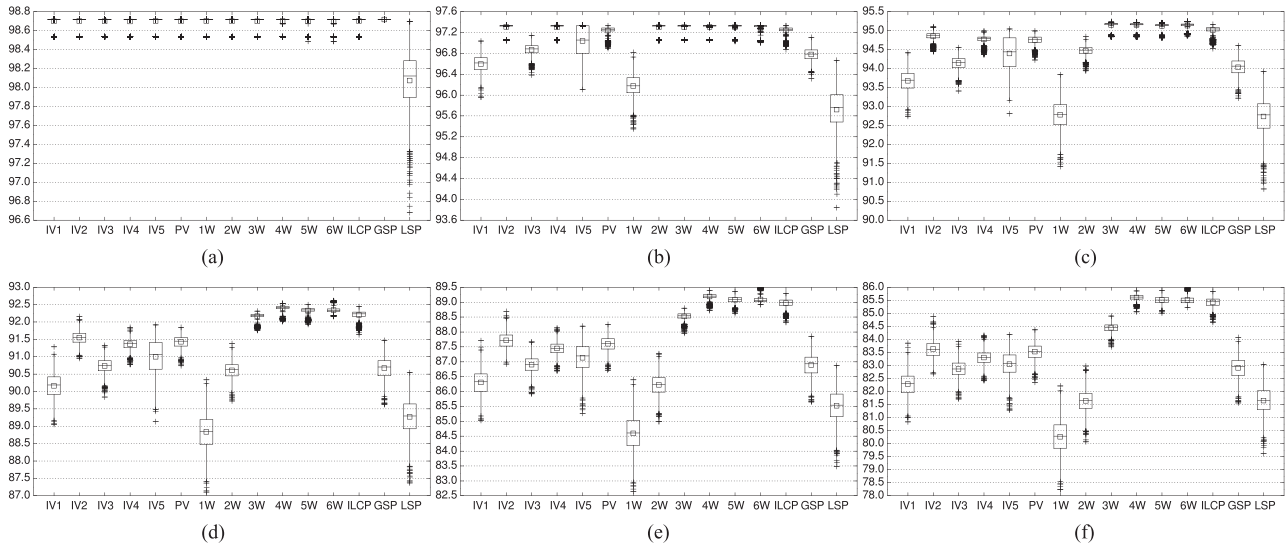


Fig. 5. APCC results for each prioritization technique for the program *gzip*. (a) $\tau = 1$. (b) $\tau = 2$. (c) $\tau = 3$. (d) $\tau = 4$. (e) $\tau = 5$. (f) $\tau = 6$.

From the perspective of the interaction coverage rate, the answer to **RQ1.1** is: Overall, IV2 has the best performance among all RSLCP-IV techniques, followed by IV4; and IV1 is generally the worst.

b) **RQ1.2***: RSLCP-IV versus RSLCP-PV:* Based on the experimental data, we have the following observations:

i) When $\tau = 1$, PV has very similar APCC values to RSLCP-IV techniques for all programs. Apart from programs *gzip* and *make*, however, the statistical analysis shows that compared with IV1 and IV3, there is no highly significant difference compared with PV; while the difference between PV and IV2, IV4, or IV5 is highly significant. In addition, the statistical analysis also shows that PV is similar to IV1 and IV3, but performs better than other RSLCP-IV techniques when $\tau = 1$.

ii) When $2 \leq \tau \leq 6$, PV is worse than IV2 for all programs, but performs better than IV1, IV3, and IV5. The statistical analysis confirms the box plot observations. PV has APCC values comparable to IV4. However, the statistical analysis shows that IV4 has significantly better 2-wise APCCs than PV; but for high τ values, the opposite is true: PV has significantly better τ-wise APCCs than IV4.

A plausible reason to the first observation is that, as was the case for IV1 and IV3, PV uses 1W at the start of the prioritization process, which guarantees that it has similar speeds to IV1 and IV3, but higher speeds than other RSLCP-IV techniques for covering 1-wise level combinations. However, IV2 uses 2W to prioritize ATCs, while PV uses 1W and 2W to guide the prioritization. Therefore, IV2 may provide faster speeds than PV

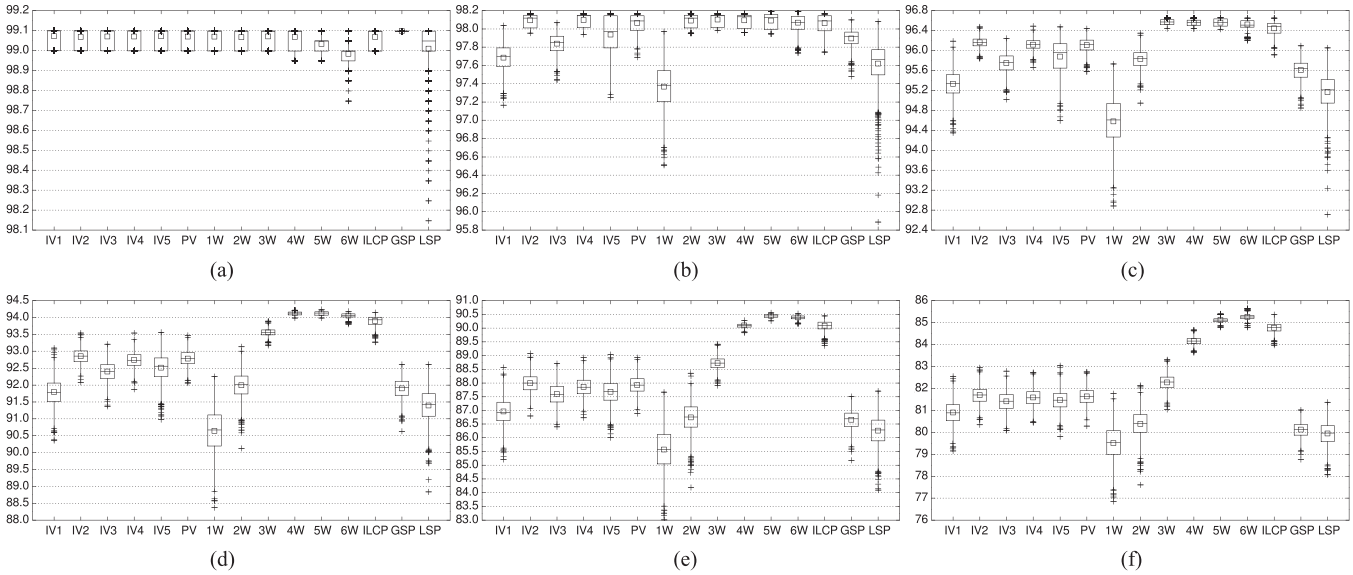


Fig. 6. APCC results for each prioritization technique for the program *make*. (a) $\tau = 1$. (b) $\tau = 2$. (c) $\tau = 3$. (d) $\tau = 4$. (e) $\tau = 5$. (f) $\tau = 6$.

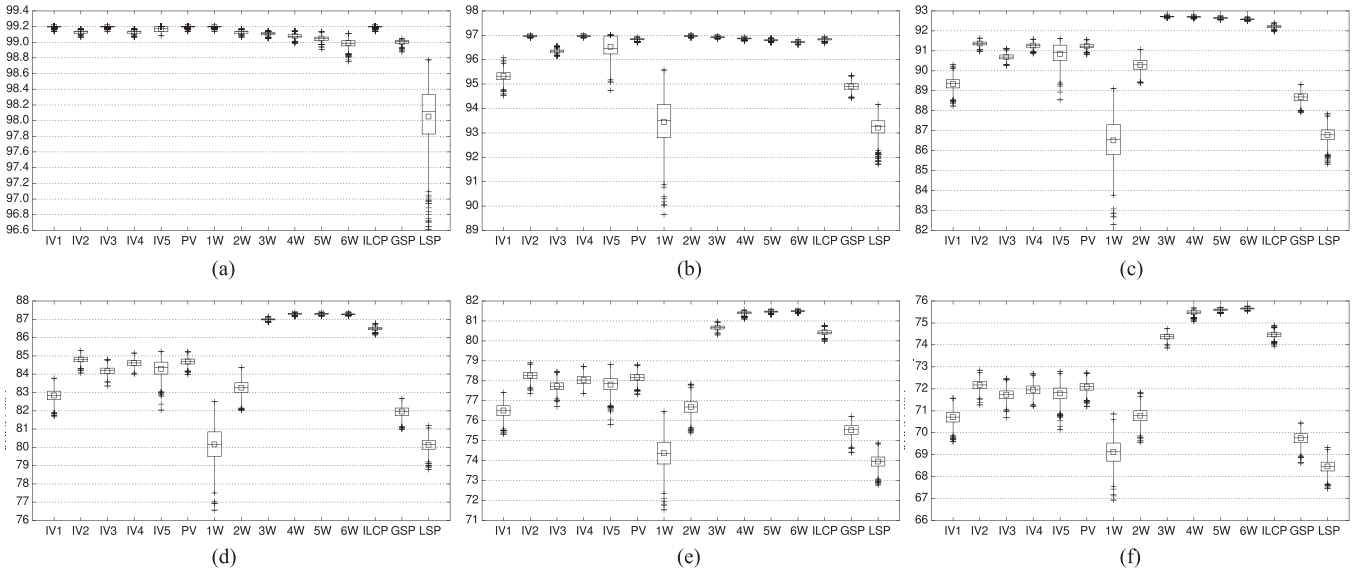


Fig. 7. APCC results for each prioritization technique for the program *sed*. (a) $\tau = 1$. (b) $\tau = 2$. (c) $\tau = 3$. (d) $\tau = 4$. (e) $\tau = 5$. (f) $\tau = 6$.

for covering high τ value level combinations. Similarly, IV4 initially uses 2W for prioritizing ATCs, and then independently chooses 1W to prioritize the remaining ATCs when 2-wise level combinations have been fully covered. This process may provide better 2-wise APCCs than PV. However, when all 1-wise level combinations have been covered by the selected ATCs, PV does not independently use 2W for prioritizing the remaining ATCs, i.e., it considers 2-wise level combinations that have not yet been covered by previously selected ATCs (obtained by 1W). This process may guarantee that PV has higher APCCs than IV4.

With respect to the interaction coverage rate, the answer to **RQ1.2** is: Overall, PV achieves a better performance than other RSLCP-IV techniques.

2) RQ2: RSLCP versus λLCP: Based on experimental results comparing RSLCP and λLCP, we have the following observations:

a) When the prioritization strength λ for λLCP is equal to the strength value τ for APCC (i.e., $\lambda = \tau$), λLCP generally has similar or better APCC values than all RSLCP techniques.
b) Compared with λLCP at low λ values (such as 1W and 2W, i.e., λ is 1 or 2), although all RSLCP techniques have similar or worse λ-wise APCC values when $\tau = \lambda$, overall, they have better APCC values at other strengths (when $\tau \neq \lambda$).
c) Compared with λLCP at high λ values (such as 3W, 4W, 5W, and 6W, i.e., $\lambda \geq 3$), when $\tau \geq 3$, RSLCP has worse

TABLE V
STATISTICAL ANALYSIS FOR PAIRWISE APCC COMPARISONS OF RSLCP TECHNIQUES ($\hat{A}_{12}$ VALUES INCLUDED IN BRACKETS)

Comparison	*flex*						*grep*						*gzip*						*make*						*sed*					
	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6
IV1 *vs* IV2	✓(0.72)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.04)	✗(0.33)	✓(0.92)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.01)	○(0.52)	✗(0.00)	✗(0.00)	✗(0.01)	✗(0.05)	✗(0.12)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV1 *vs* IV3	○(0.51)	✗(0.00)	✗(0.00)	✗(0.01)	✗(0.10)	✗(0.34)	○(0.52)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.09)	✗(0.08)	✗(0.10)	✗(0.13)	✗(0.15)	○(0.51)	✗(0.20)	✗(0.11)	✗(0.12)	✗(0.16)	✗(0.23)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.01)
IV1 *vs* IV4	✓(0.72)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.06)	✗(0.30)	✓(0.91)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.01)	✗(0.03)	○(0.51)	✗(0.00)	✗(0.00)	✗(0.02)	✗(0.07)	✗(0.15)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV1 *vs* IV5	✓(0.62)	✗(0.03)	✗(0.01)	✗(0.02)	✗(0.09)	✗(0.32)	✓(0.71)	✗(0.01)	✗(0.01)	✗(0.00)	✗(0.01)	✗(0.01)	○(0.50)	✗(0.11)	✗(0.10)	✗(0.10)	✗(0.11)	✗(0.13)	○(0.50)	✗(0.16)	✗(0.11)	✗(0.11)	✗(0.14)	✗(0.21)	✓(0.74)	✗(0.01)	✗(0.01)	✗(0.01)	✗(0.01)	✗(0.02)
IV2 *vs* IV3	✗(0.29)	✓(1.00)	✓(0.97)	✓(0.82)	✓(0.66)	○(0.50)	✗(0.09)	✓(1.00)	✓(1.00)	✓(0.95)	✓(0.89)	✓(0.83)	○(0.51)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.98)	✓(0.95)	○(0.49)	✓(0.98)	✓(0.98)	✓(0.88)	✓(0.77)	✓(0.68)	✗(0.00)	✓(1.00)	✓(1.00)	✓(0.98)	✓(0.94)	✓(0.90)
IV2 *vs* IV4	○(0.50)	○(0.49)	✓(0.67)	✓(0.65)	✓(0.58)	✗(0.45)	○(0.49)	○(0.49)	✓(0.74)	✓(0.75)	✓(0.75)	✓(0.73)	○(0.51)	✓(0.52)	✓(0.76)	✓(0.77)	✓(0.77)	✓(0.77)	○(0.49)	○(0.50)	✓(0.61)	✓(0.63)	✓(0.61)	✓(0.58)	○(0.50)	○(0.52)	✓(0.76)	✓(0.75)	✓(0.75)	✓(0.74)
IV2 *vs* IV5	✗(0.40)	✓(0.74)	✓(0.79)	✓(0.72)	✓(0.62)	○(0.48)	✗(0.29)	✓(0.75)	✓(0.85)	✓(0.84)	✓(0.81)	✓(0.78)	○(0.50)	✓(0.77)	✓(0.85)	✓(0.85)	✓(0.85)	✓(0.83)	○(0.48)	✓(0.74)	✓(0.78)	✓(0.76)	✓(0.70)	✓(0.64)	✗(0.24)	✓(0.76)	✓(0.85)	✓(0.85)	✓(0.83)	✓(0.81)
IV3 *vs* IV4	✓(0.71)	✗(0.00)	✗(0.05)	✗(0.29)	✗(0.42)	✗(0.46)	✓(0.91)	✗(0.00)	✗(0.02)	✗(0.16)	✗(0.29)	✗(0.36)	○(0.51)	✗(0.00)	✗(0.00)	✗(0.01)	✗(0.07)	✗(0.16)	○(0.50)	✗(0.02)	✗(0.04)	✗(0.19)	✗(0.31)	✗(0.39)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.08)	✗(0.19)	✗(0.27)
IV3 *vs* IV5	✓(0.61)	✗(0.32)	✗(0.33)	✗(0.42)	✗(0.47)	○(0.48)	✓(0.70)	✗(0.39)	✗(0.37)	✗(0.40)	✗(0.44)	✗(0.46)	○(0.49)	✗(0.33)	✗(0.33)	✗(0.32)	✗(0.33)	✗(0.36)	○(0.49)	✗(0.32)	✗(0.34)	✗(0.39)	✗(0.44)	✗(0.47)	✓(0.74)	✗(0.38)	✗(0.36)	✗(0.37)	✗(0.40)	✗(0.43)
IV4 *vs* IV5	✗(0.40)	✓(0.74)	✓(0.69)	✓(0.60)	✓(0.55)	✓(0.53)	✗(0.30)	✓(0.75)	✓(0.73)	✓(0.67)	✓(0.62)	✓(0.58)	○(0.49)	✓(0.75)	✓(0.71)	✓(0.71)	✓(0.69)	✓(0.65)	○(0.49)	✓(0.74)	✓(0.72)	✓(0.67)	✓(0.61)	✓(0.57)	✗(0.24)	✓(0.75)	✓(0.73)	✓(0.70)	✓(0.66)	✓(0.63)
IV1 *vs* PV	○(0.51)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.04)	✗(0.28)	○(0.52)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.01)	○(0.51)	✗(0.01)	✗(0.01)	✗(0.02)	✗(0.06)	✗(0.14)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV2 *vs* PV	✗(0.29)	✓(0.68)	✓(0.57)	✓(0.53)	○(0.50)	✗(0.43)	✗(0.10)	✓(0.94)	✓(0.59)	○(0.51)	○(0.49)	○(0.48)	○(0.51)	✓(0.92)	✓(0.78)	✓(0.67)	✓(0.61)	✓(0.58)	○(0.49)	✓(0.61)	✓(0.61)	✓(0.58)	✓(0.56)	✓(0.54)	✗(0.00)	✓(1.00)	✓(0.80)	✓(0.66)	✓(0.61)	✓(0.59)
IV3 *vs* PV	○(0.50)	✗(0.00)	✗(0.04)	✗(0.20)	✗(0.35)	✗(0.44)	○(0.51)	✗(0.00)	✗(0.00)	✗(0.05)	✗(0.11)	✗(0.16)	○(0.50)	✗(0.01)	✗(0.00)	✗(0.01)	✗(0.04)	✗(0.08)	○(0.50)	✗(0.06)	✗(0.07)	✗(0.16)	✗(0.27)	✗(0.36)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.04)	✗(0.10)	✗(0.15)
IV4 *vs* PV	✗(0.29)	✓(0.69)	✗(0.40)	✗(0.38)	✗(0.42)	○(0.48)	✗(0.10)	✓(0.94)	✗(0.34)	✗(0.26)	✗(0.25)	✗(0.25)	○(0.49)	✓(0.90)	✓(0.57)	✗(0.39)	✗(0.32)	✗(0.30)	○(0.50)	✓(0.61)	○(0.51)	✗(0.45)	✗(0.44)	✗(0.46)	✗(0.00)	✓(1.00)	✓(0.55)	✗(0.39)	✗(0.35)	✗(0.34)
IV5 *vs* PV	✗(0.39)	✗(0.36)	✗(0.25)	✗(0.30)	✗(0.38)	✗(0.45)	✗(0.31)	✗(0.46)	✗(0.19)	✗(0.17)	✗(0.19)	✗(0.21)	○(0.50)	✗(0.46)	✗(0.31)	✗(0.23)	✗(0.21)	✗(0.21)	○(0.51)	✗(0.33)	✗(0.29)	✗(0.30)	✗(0.34)	✗(0.39)	✗(0.26)	○(0.48)	✗(0.29)	✗(0.23)	✗(0.23)	✗(0.25)

TABLE VI
STATISTICAL ANALYSIS FOR PAIRWISE APCC COMPARISONS OF RSLCP AGAINST OTHER ATCP TECHNIQUES ($\hat{A}_{12}$ VALUES INCLUDED IN BRACKETS)

Comparison	*flex*						*grep*						*gzip*						*make*						*sed*					
	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6	τ=1	τ=2	τ=3	τ=4	τ=5	τ=6
IV1 *vs* 1W	○(0.50)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.52)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.90)	✓(0.65)	○(0.49)	✓(0.93)	✓(0.97)	✓(0.98)	✓(0.99)	✓(1.00)	○(0.50)	✓(0.87)	✓(0.91)	✓(0.93)	✓(0.93)	✓(0.93)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)
IV2 *vs* 1W	✗(0.28)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.09)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.50)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV3 *vs* 1W	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.50)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.50)	✓(0.97)	✓(0.99)	✓(1.00)	✓(0.99)	✓(0.99)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV4 *vs* 1W	✗(0.28)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.10)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.50)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	✗(0.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV5 *vs* 1W	✗(0.38)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.30)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	○(0.50)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.50)	✓(0.97)	✓(0.99)	✓(0.99)	✓(0.99)	✓(0.99)	✗(0.25)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
PV *vs* 1W	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV1 *vs* 2W	✓(0.73)	✗(0.00)	✗(0.23)	✓(0.87)	✓(0.96)	✓(0.97)	✓(0.91)	✗(0.00)	✗(0.00)	✗(0.01)	✗(0.02)	✗(0.02)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.16)	✓(0.56)	✓(0.84)	✓(0.52)	✗(0.00)	✗(0.07)	✗(0.35)	✓(0.60)	✓(0.73)	✓(1.00)	✗(0.00)	✗(0.01)	✗(0.22)	✗(0.37)	✗(0.44)
IV2 *vs* 2W	○(0.51)	○(0.50)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	○(0.48)	○(0.48)	✓(1.00)	✓(0.99)	✓(0.96)	✓(0.86)	○(0.51)	○(0.50)	✓(0.98)	✓(1.00)	✓(1.00)	✓(1.00)	○(0.51)	○(0.50)	✓(0.95)	✓(0.98)	✓(0.98)	✓(0.97)	○(0.49)	○(0.49)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV3 *vs* 2W	✓(0.72)	✗(0.00)	✓(0.99)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.90)	✗(0.00)	✓(0.72)	✓(0.86)	✓(0.77)	✓(0.59)	○(0.50)	✗(0.00)	✗(0.07)	✓(0.65)	✓(0.93)	✓(0.99)	○(0.51)	✗(0.02)	✗(0.38)	✓(0.79)	✓(0.90)	✓(0.91)	✓(1.00)	✗(0.00)	✓(0.87)	✓(0.98)	✓(0.99)	✓(0.99)
IV4 *vs* 2W	○(0.51)	○(0.51)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	○(0.49)	○(0.49)	✓(0.99)	✓(0.97)	✓(0.88)	✓(0.70)	○(0.49)	✗(0.48)	✓(0.94)	✓(0.99)	✓(1.00)	✓(1.00)	○(0.52)	○(0.51)	✓(0.91)	✓(0.96)	✓(0.96)	✓(0.95)	○(0.48)	✗(0.47)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV5 *vs* 2W	✓(0.61)	✗(0.26)	✓(0.96)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.70)	✗(0.24)	✓(0.71)	✓(0.83)	✓(0.77)	✓(0.62)	○(0.51)	✗(0.23)	○(0.50)	✓(0.75)	✓(0.92)	✓(0.98)	✓(0.52)	✗(0.26)	✓(0.59)	✓(0.82)	✓(0.91)	✓(0.92)	✓(0.75)	✗(0.24)	✓(0.82)	✓(0.95)	✓(0.97)	✓(0.97)
PV *vs* 2W	✓(0.72)	✗(0.32)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.89)	✗(0.06)	✓(1.00)	✓(0.99)	✓(0.96)	✓(0.86)	○(0.50)	✗(0.10)	✓(0.93)	✓(0.99)	✓(1.00)	✓(1.00)	○(0.51)	✗(0.40)	✓(0.88)	✓(0.96)	✓(0.97)	✓(0.96)	✓(1.00)	✗(0.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV1 *vs* 3W	✓(0.90)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.16)	✓(0.98)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.02)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV2 *vs* 3W	✓(0.80)	✓(0.92)	✗(0.00)	✗(0.00)	✗(0.04)	✗(0.25)	✓(0.83)	✓(0.98)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✓(0.52)	✗(0.06)	✗(0.00)	✗(0.01)	✗(0.02)	○(0.48)	○(0.48)	✗(0.00)	✗(0.00)	✗(0.04)	✗(0.15)	✓(0.76)	✓(0.96)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV3 *vs* 3W	✓(0.90)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.02)	✗(0.27)	✓(0.98)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.01)	✗(0.07)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV4 *vs* 3W	✓(0.80)	✓(0.93)	✗(0.00)	✗(0.00)	✗(0.03)	✗(0.30)	✓(0.83)	✓(0.98)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	○(0.50)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.02)	✗(0.11)	✓(0.76)	✓(0.95)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV5 *vs* 3W	✓(0.85)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.03)	✗(0.28)	✓(0.91)	○(0.48)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.25)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.25)	✗(0.00)	✗(0.00)	✗(0.02)	✗(0.09)	✓(0.88)	✗(0.46)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
PV *vs* 3W	✓(0.90)	✓(0.80)	✗(0.00)	✗(0.00)	✗(0.05)	✗(0.31)	✓(0.98)	✓(0.66)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.09)	✗(0.02)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	✗(0.37)	✗(0.00)	✗(0.00)	✗(0.03)	✗(0.12)	✓(1.00)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV1 *vs* 4W	✓(0.99)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV2 *vs* 4W	✓(0.97)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.97)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✓(0.53)	✗(0.05)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✓(0.53)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.95)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV3 *vs* 4W	✓(0.99)	✗(0.46)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.02)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV4 *vs* 4W	✓(0.97)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.97)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	○(0.51)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✓(0.54)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.95)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV5 *vs* 4W	✓(0.98)	✓(0.67)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.99)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.25)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✗(0.28)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.98)	○(0.48)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
PV *vs* 4W	✓(0.99)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.08)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.40)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.25)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV1 *vs* 5W	✓(1.00)	✗(0.05)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.74)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV2 *vs* 5W	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✓(0.55)	✗(0.06)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.72)	○(0.48)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.99)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV3 *vs* 5W	✓(1.00)	✓(0.99)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.43)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.73)	✗(0.02)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV4 *vs* 5W	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	✓(0.53)	✗(0.02)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.73)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.99)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV5 *vs* 5W	✓(1.00)	✓(0.88)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(0.59)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✗(0.26)	✗(0.02)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.74)	✗(0.25)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	○(0.48)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
PV *vs* 5W	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.10)	✗(0.02)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.73)	✗(0.39)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(0.82)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV1 *vs* 6W	✓(1.00)	✓(0.75)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.51)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.88)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV2 *vs* 6W	✓(1.00)	✓(1.00)	✓(0.81)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.52)	✓(0.57)	✗(0.04)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.87)	✓(0.60)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV3 *vs* 6W	✓(1.00)	✓(1.00)	✗(0.08)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.88)	✗(0.04)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV4 *vs* 6W	✓(1.00)	✓(1.00)	✓(0.66)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✓(0.55)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.88)	✓(0.61)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV5 *vs* 6W	✓(1.00)	✓(0.99)	✗(0.40)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(0.81)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.52)	✗(0.28)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.88)	✗(0.32)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	○(0.48)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
PV *vs* 6W	✓(1.00)	✓(1.00)	✓(0.75)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.52)	✗(0.15)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	✓(0.88)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✓(0.98)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV1 *vs* ILCP	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV2 *vs* ILCP	✗(0.28)	✓(0.68)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.08)	✓(0.93)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✓(0.92)	✗(0.11)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	✓(0.61)	✗(0.06)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV3 *vs* ILCP	○(0.49)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.01)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.06)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV4 *vs* ILCP	✗(0.28)	✓(0.69)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.08)	✓(0.93)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.49)	✓(0.89)	✗(0.07)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✓(0.61)	✗(0.04)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✓(1.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV5 *vs* ILCP	✗(0.38)	✗(0.36)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.29)	✗(0.45)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	✗(0.46)	✗(0.04)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	✗(0.33)	✗(0.02)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.26)	○(0.48)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
PV *vs* ILCP	○(0.49)	○(0.50)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.48)	✗(0.47)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	○(0.49)	✗(0.06)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.50)	○(0.50)	✗(0.04)	✗(0.00)	✗(0.00)	✗(0.00)	○(0.51)	○(0.51)	✗(0.00)	✗(0.00)	✗(0.00)	✗(0.00)
IV1 *vs* GSP	✓(0.79)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.81)	✓(0.85)	✓(0.92)	✓(0.95)	✓(0.97)	✓(0.98)	✗(0.45)	✗(0.19)	✗(0.16)	✗(0.15)	✗(0.15)	✗(0.16)	✗(0.36)	✗(0.10)	✗(0.22)	✗(0.41)	✓(0.68)	✓(0.89)	✓(1.00)	✓(0.94)	✓(0.96)	✓(0.97)	✓(0.98)	✓(0.99)
IV2 *vs* GSP	✓(0.59)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.23)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.46)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.97)	✓(0.92)	✗(0.34)	✓(0.98)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV3 *vs* GSP	✓(0.78)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.80)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.45)	✓(0.71)	✓(0.64)	✓(0.56)	○(0.49)	✗(0.46)	✗(0.35)	✗(0.34)	✓(0.69)	✓(0.87)	✓(0.96)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV4 *vs* GSP	✓(0.60)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.25)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.44)	✓(1.00)	✓(1.00)	✓(0.98)	✓(0.90)	✓(0.78)	✗(0.35)	✓(0.97)	✓(0.99)	✓(0.99)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV5 *vs* GSP	✓(0.68)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.53)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.45)	✓(0.76)	✓(0.74)	✓(0.71)	✓(0.66)	✓(0.60)	✗(0.36)	✓(0.61)	✓(0.76)	✓(0.88)	✓(0.96)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
PV *vs* GSP	✓(0.79)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.79)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✗(0.45)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.95)	✓(0.89)	✗(0.35)	✓(0.90)	✓(0.98)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV1 *vs* LSP	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.89)	✓(0.78)	✓(0.71)	✓(1.00)	✓(0.99)	✓(0.96)	✓(0.92)	✓(0.87)	✓(0.81)	✓(0.69)	✓(0.57)	✓(0.63)	✓(0.73)	✓(0.82)	✓(0.89)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV2 *vs* LSP	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.67)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV3 *vs* LSP	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.98)	✓(0.68)	✓(0.82)	✓(0.93)	✓(0.96)	✓(0.97)	✓(0.98)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV4 *vs* LSP	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.68)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
IV5 *vs* LSP	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	✓(0.98)	✓(0.97)	✓(0.69)	✓(0.86)	✓(0.92)	✓(0.96)	✓(0.97)	✓(0.98)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)
PV *vs* LSP	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.68)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(0.99)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)	✓(1.00)

APCC values, irrespective of subject program. However, when $1 \leq \tau < 3$, RSLCP has comparable, or even better, performance.

d) The statistical analysis validates above three observations overall.

The first observation can be easily explained: λLCP chooses the next test case with the largest number of uncovered λ-wise level combinations, leading to highest rate of interaction coverage. The second observation can be explained by the fact that RSLCP repeatedly achieves full 1-wise or 2-wise level combination coverage, which may provide comparable level combination coverage at high τ values; however, once 1-wise (or 2-wise) level combinations have been full covered by 1W (or 2W), the remaining ATCs are randomly prioritized.

The third observation can be explained as follows: The RSLCP techniques focus on the prioritization strength equal to either 1 or 2, which guarantees better APCCs at $\tau = 1$ or $\tau = 2$. Although RSLCP could cover level combinations at high τ values, its speed may be lower than that of λLCP at high λ values. Additionally, λLCP also achieves a faster speed of covering level combinations at low τ values, which may explain why λLCP could sometimes achieve better APCCs at low τ values (e.g., 3W, 4W, and 5W have higher 2-wise APCCs than IV1, IV3, IV5, and PV for the programs *gzip* and *make*).

From the perspective of the interaction coverage rate, the answer to **RQ2** is: RSLCP generally has better τ-wise APCC performances than λLCP at low λ values for $\tau \neq \lambda$; while it has comparable (and even better) τ-wise APCCs for $1 \leq \tau < 3$, compared with 3W, 4W, 5W, and 6W.

3) RQ3: RSLCP versus ILCP and SP: As illustrated in Figs. 3–7 and Table VI, it can be observed that:

a) ILCP, IV1, IV3, and IV5 have comparable APCCs when $\tau = 1$; while IV2 and IV4 achieve better APCCs when $\tau = 2$. Similarly, PV achieves comparable 1-wise and 2-wise APCCs to ILCP. However, for all other τ values (i.e., $3 \leq \tau \leq 6$), ILCP performs significantly better than all RSLCP techniques.

b) Compared with GSP, different programs have different observations. More specifically, RSLCP and GSP have very similar 1-wise APCCs for the programs *flex* and *grep*; GSP has higher 1-wise APCCs than RSLCP for the programs *gzip* and *make*. For the program *sed*, however, the 1-wise

APCC performances of RSLCP are better than those of GSP. In addition, the statistical analysis shows that, overall, RSLCP performs significantly better than GSP for the programs *flex*, *grep*, and *sed*; while the opposite is true for *gzip* and *make*. Nevertheless, when τ is greater than 1 (i.e., $2 \leq \tau \leq 6$), all RSLCP techniques apart from IV1 have significantly better performances than GSP, for all programs. IV1 performs better than GSP for the programs *flex*, *grep*, and *sed*, but has worse performance for *gzip*. Regarding the program *make*, IV1 outperforms GSP for τ equal to 2, 3, and 4, but GSP is better than IV1 for τ equal to 5 and 6.

c) Compared with LSP, all RSLCP techniques have higher APCCs, irrespective of subject programs and τ values. The statistical results confirm that the differences between the APCCs of RSLCP and LSP are highly significant.

The first observation can be explained as follows: On the one hand, ILCP first adopts the same procedure as PV (it initially uses 1W for prioritizing ATCs until all 1-wise level combinations have been covered by the selected ATCs), and then uses 2W for prioritizing the remaining ATCs to cover uncovered 2-wise level combinations as quickly as possible. Therefore, ILCP has comparable 1-wise APCCs to IV1, IV3, IV5, and PV, and also has similar 2-wise APCCs to PV. However, since IV2 and IV4 initially use 2W to guide the prioritization, it is understandable that their 2-wise APCCs are better than ILCPs. On the other hand, when all 2-wise level combinations have been covered by previously selected ATCs, ILCP makes use of 3W to prioritize the remaining ATCs, and attempts to cover uncovered 3-wise level combinations as quickly as possible. ILCP iteratively repeats this process, incrementing by 1 each time, until all ATCs have been chosen, which may guarantee that ILCP has higher APCCs at high τ values than RSLCP.

The second observation (that RSLCP generally has better APCCs than GSP) can be explained as follows: RSLCP makes use of the information of interaction coverage to guide the prioritization of ATCs, but GSP uses the information of similarity between each candidate and previously selected ATCs. Therefore, RSLCP provides faster speed of covering level combinations than GSP. Regarding why GSP has higher 1-wise APCCs than RSLCP for the programs *make* and *gzip*: GSP initially selects two elements from candidates as the first two ATCs with the minimum Jaccard similarity, indicating that these two ATCs cover the largest number of 1-wise level combinations. Although GSP may choose a different pair of elements as the first two ATCs due to the tie-breaking [28], each pair of ATCs cover the same maximum number of 1-wise level combinations. However, RSLCP randomly chooses an element as the first ATC tc_1, and then chooses the second ATC tc_2 such that it may cover the largest number of uncovered 1-wise level combinations by tc_1. In other words, it is not guaranteed that tc_1 and tc_2 can cover the maximum number of 1-wise level combinations among all pairs of ATCs. As shown in Table II, the input parameter models for both *make* and *gzip* contain nearly all parameters with the same number of levels, which means that it is difficult for RSLCP to choose two ATCs with the minimum Jaccard similarity. Therefore, after choosing two ATCs, RSLCP may cover fewer 1-wise level combinations than GSP, resulting in the lower 1-wise APCC.

For the final observation, the main reason can be described as follows: LSP iteratively chooses a pair of elements with the minimum Jaccard similarity as the next two ATCs until all candidates have been selected, which indicates that each pair of ATCs is constructed independently. In other words, it is possible that two successive pairs of ATCs cover very similar level combinations, leading to low interaction coverage rates.

With respect to the rate of interaction coverage, the answer to **RQ3** is: RSLCP generally achieves lower APCCs than ILCP, in most cases, but always has better performance than SBP, and often also for GSP.

B. Fault Detection Results

In this section, we answer **RQ1**, **RQ2**, and **RQ3**, in terms of the fault detection rates. Figs. 8–12 present the APFD results for programs *flex*, *grep*, *gzip*, *make*, and *sed*, respectively. Each figure describes different versions for APFD. Tables VII and VIII show the detailed Wilcoxon test APFD results at the 0.05 significance level for each comparison.

1) RQ1: RSLCP Techniques: Here, we attempt to answer the two sub-questions of **RQ1**, **RQ1.1**, and **RQ1.2**, according to the APFD results.

a) **RQ1.1***: RSLCP-IV Techniques:* Based on the experimental data, we have the following observations:

i) Among the five RSLCP-IV techniques, IV1 generally performs worst, regardless of subject programs with different versions. However, the APFD difference between IV1 and the other RSLCP-IV techniques (IV2, IV3, IV4, and IV5) seems small, in terms of both *median* and *mean* values: the differences between the mean values of IV1 and each of the other four RSLCP-IV techniques are less than 3%; and the differences between the median values range from 0% to 3%—program *gzip* with versions *v3*, *v4*, and *v5*, for example, seems to have no difference in the median values for each comparison. As shown in Table VII, it can be observed that, apart from the comparison of IV1 and IV3 for the program *gzip* with the last three versions (i.e., *v3*, *v4*, and *v5*), the differences when comparing any IVx $(2 \leq x \leq 5)$ technique against IV1 are highly significant—the corresponding p-values are less than 0.05. Additionally, the effect size measure $\hat{\mathrm{A}}_{12}(\mathrm{IV1}, \mathrm{IV}x)$ values are less than 0.50, which means that IVx outperforms IV1 more than 50% of the time for each version of each program.

ii) Regarding the comparisons among IV2, IV3, IV4, and IV5, the greatest difference is less than 1% among all programs, for both mean and median APFD values. In other words, other than IV1, the RSLCP-IV techniques all have very similar performance (according to the fault detection rates). Regarding the comparison between IVx and IVy $(2 \leq x \neq y \leq 5)$, most p-values are greater than 0.05, indicating that any differences between them are not significant. However, the effect size measure $\hat{\mathrm{A}}_{12}(\mathrm{IV}x, \mathrm{IV}y)$ results show that IV2 is the best, followed

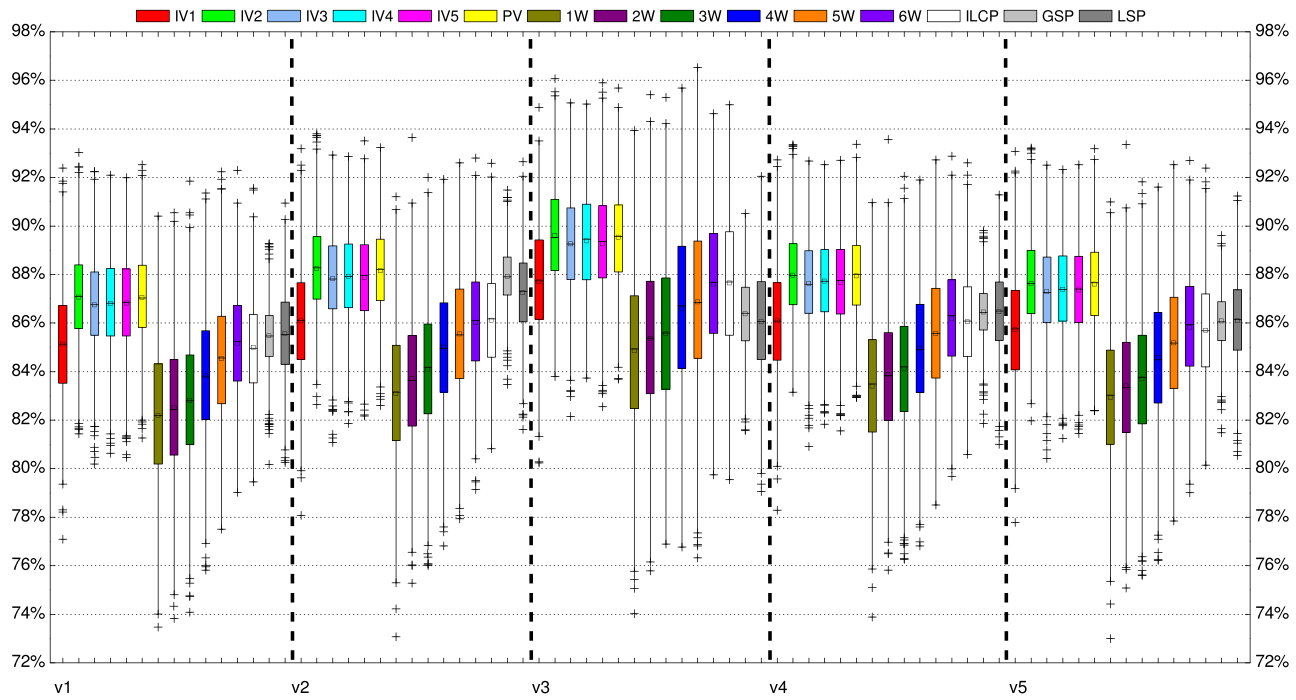


Fig. 8. APFD results for each prioritization technique for the program *flex*.

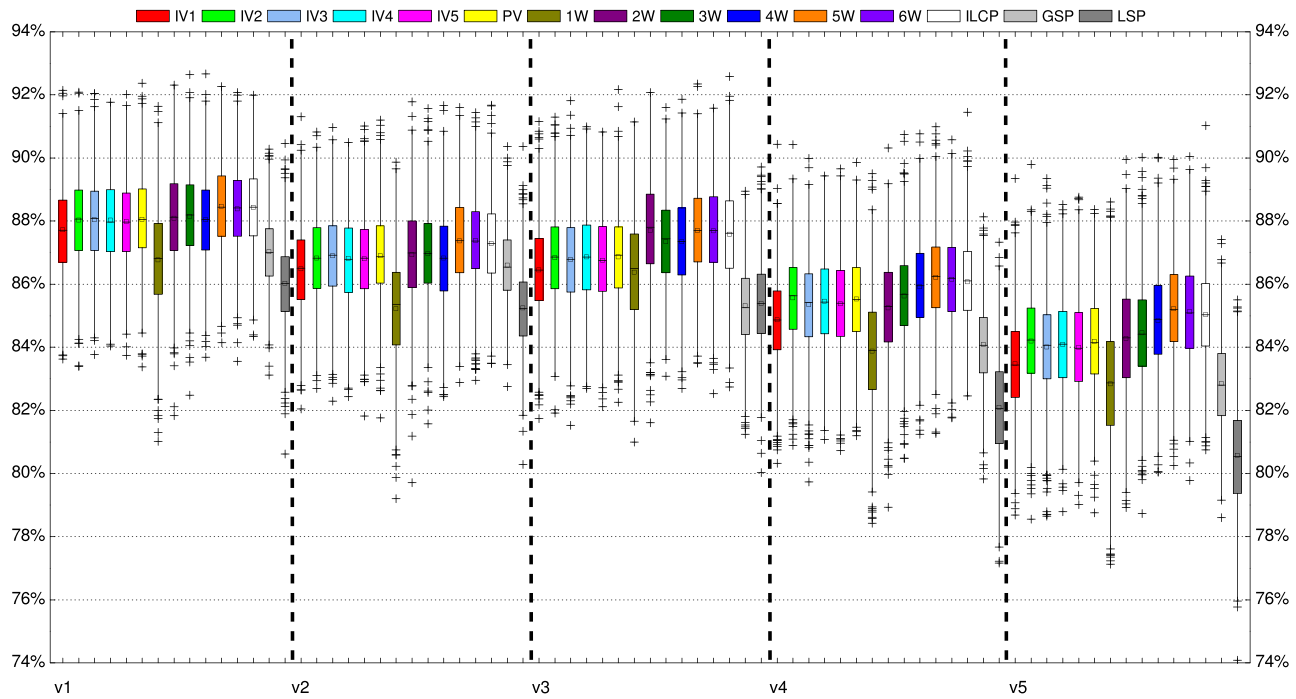


Fig. 9. APFD results for each prioritization technique for the program *grep*.

by IV4 and IV5. Overall, the statistical analysis confirms the box plots observations.

From the perspective of fault detection rate, the answer to **RQ1.1** is: IV1 has the worst fault detection rates, while other RSLCP-IV techniques are similarly effective.

b) **RQ1.2***: RSLCP-IV versus RSLCP-PV:* Based on the experimental results, we can observe the following:

Based on the box plots results, PV achieves higher APFDs than IV1, and has comparable performance to the other RSLCP-IV techniques, regardless of subject programs. According to the statistical analysis, however, we have the following observations.

i) Regarding the p-values, all of the IV1 versus PV comparisons are highly significant, except for the programs

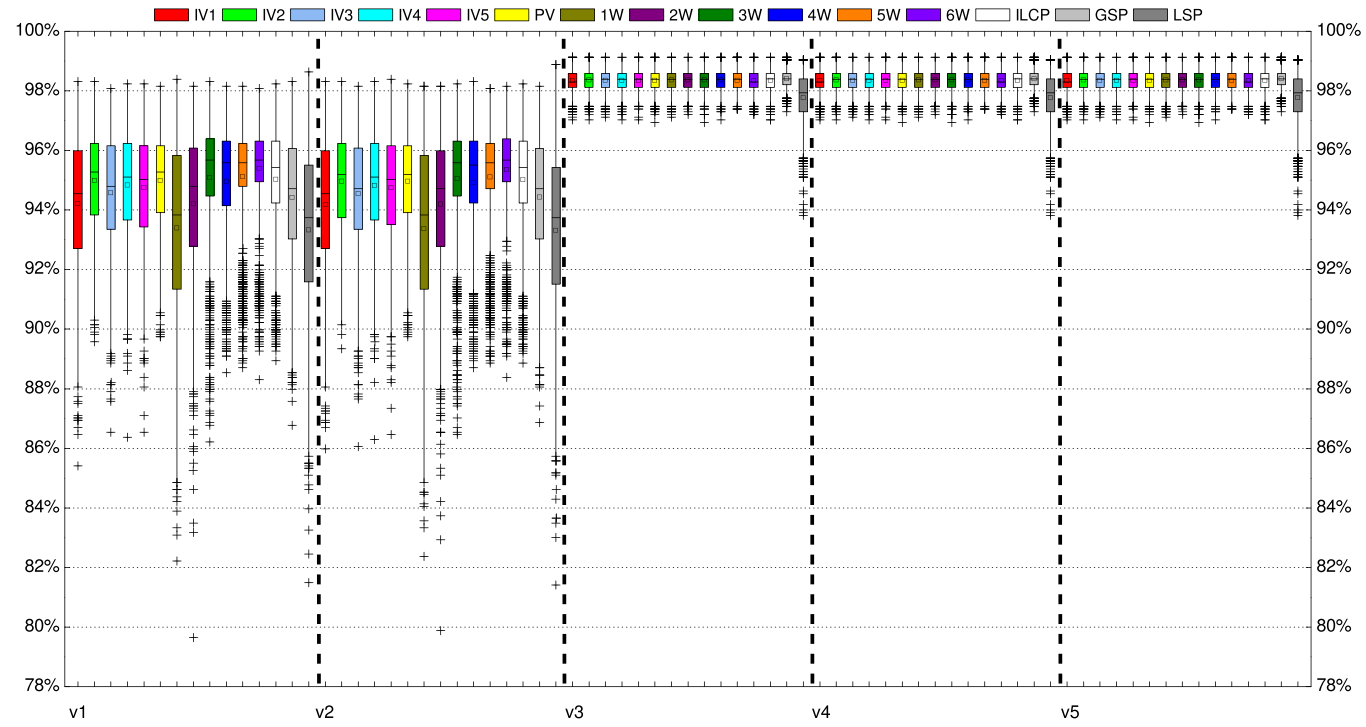


Fig. 10. APFD results for each prioritization technique for the program *gzip*.

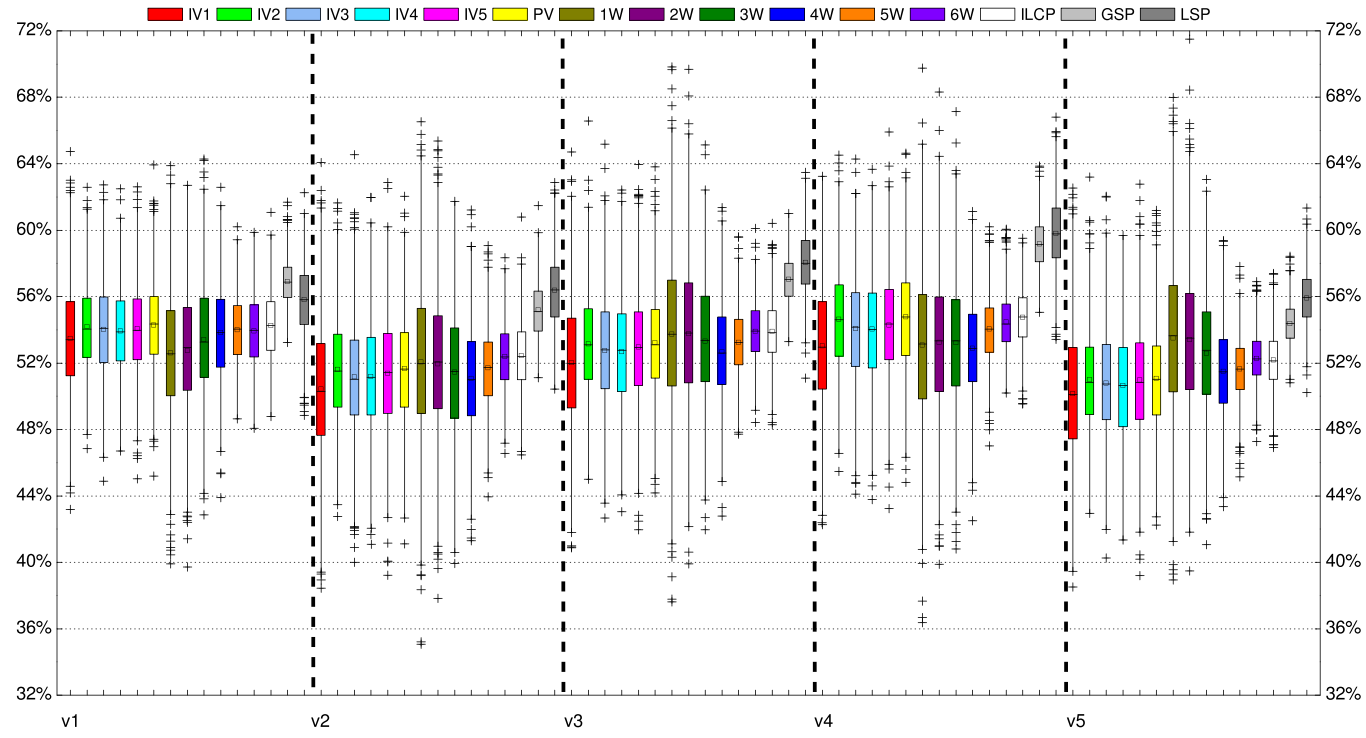


Fig. 11. APFD results for each prioritization technique for the program *make*.

gzip-v3, *gzip-v4*, and *gzip-v5*. This means that the APFDs are very different when comparing IV1 and PV. The effect size measure $\hat{A}_{12}(\text{IV1}, \text{PV})$ values have different ranges for different programs: for example, there are large differences for the programs *flex* and *sed*, regardless of versions, because the $\hat{A}_{12}(\text{IV1}, \text{PV})$ values range from 0.24 to 0.33 (except for the program *sed-v1*). However, for the programs *grep*, *gzip*, and *make*, the $\hat{A}_{12}(\text{IV1}, \text{PV})$

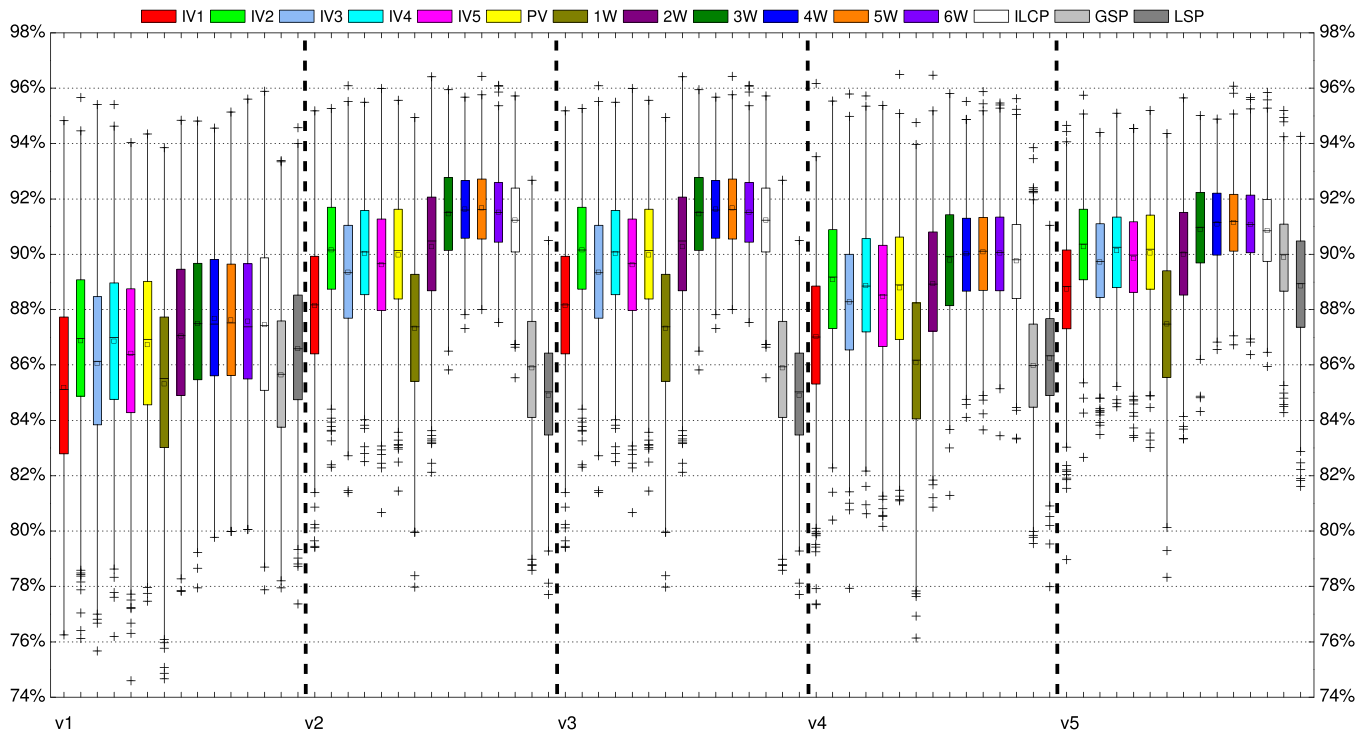


Fig. 12. APFD results for each prioritization technique for the program *sed*.

TABLE VII
STATISTICAL ANALYSIS FOR PAIRWISE APFD COMPARISONS OF RSLCP TECHNIQUES ($\hat{A}_{12}$ VALUES INCLUDED IN BRACKETS)

Comparison	*flex*					*grep*					*gzip*					*make*					*sed*				
	v1	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*
IV1 *vs* IV2	✗(0.26)	✗(0.23)	✗(0.28)	✗(0.26)	✗(0.26)	✗(0.43)	✗(0.43)	✗(0.43)	✗(0.36)	✗(0.37)	✗(0.41)	✗(0.41)	○(0.48)	○(0.48)	○(0.48)	✗(0.43)	✗(0.41)	✗(0.41)	✗(0.38)	✗(0.43)	✗(0.36)	✗(0.28)	✗(0.28)	✗(0.29)	✗(0.30)
IV1 *vs* IV3	✗(0.30)	✗(0.28)	✗(0.32)	✗(0.30)	✗(0.30)	✗(0.43)	✗(0.42)	✗(0.44)	✗(0.40)	✗(0.40)	✗(0.46)	✗(0.46)	○(0.48)	○(0.48)	○(0.48)	✗(0.45)	✗(0.44)	✗(0.44)	✗(0.42)	✗(0.45)	✗(0.43)	✗(0.37)	✗(0.37)	✗(0.37)	✗(0.37)
IV1 *vs* IV4	✗(0.29)	✗(0.27)	✗(0.31)	✗(0.29)	✗(0.29)	✗(0.44)	✗(0.44)	✗(0.42)	✗(0.39)	✗(0.39)	✗(0.43)	✗(0.43)	○(0.48)	○(0.48)	○(0.48)	✗(0.46)	✗(0.44)	✗(0.45)	✗(0.42)	✗(0.46)	✗(0.36)	✗(0.29)	✗(0.29)	✗(0.31)	✗(0.32)
IV1 *vs* IV5	✗(0.29)	✗(0.27)	✗(0.32)	✗(0.29)	✗(0.30)	✗(0.44)	✗(0.43)	✗(0.44)	✗(0.40)	✗(0.41)	✗(0.44)	✗(0.44)	○(0.48)	○(0.48)	○(0.48)	✗(0.44)	✗(0.43)	✗(0.43)	✗(0.40)	✗(0.44)	✗(0.39)	✗(0.34)	✗(0.34)	✗(0.35)	✗(0.35)
IV2 *vs* IV3	✓(0.54)	✓(0.56)	✓(0.54)	✓(0.55)	✓(0.55)	○(0.50)	○(0.49)	○(0.51)	✓(0.54)	✓(0.53)	✓(0.55)	✓(0.55)	○(0.51)	○(0.51)	○(0.51)	○(0.51)	✓(0.53)	✓(0.53)	✓(0.55)	○(0.52)	✓(0.58)	✓(0.60)	✓(0.60)	✓(0.59)	✓(0.58)
IV2 *vs* IV4	✓(0.54)	✓(0.55)	✓(0.53)	✓(0.53)	✓(0.53)	○(0.50)	○(0.51)	○(0.50)	○(0.52)	○(0.52)	○(0.52)	○(0.52)	○(0.50)	○(0.50)	○(0.50)	○(0.52)	✓(0.54)	✓(0.54)	✓(0.55)	✓(0.53)	○(0.50)	○(0.51)	○(0.51)	✓(0.53)	○(0.52)
IV2 *vs* IV5	✓(0.53)	✓(0.55)	✓(0.54)	✓(0.54)	✓(0.54)	○(0.51)	○(0.51)	○(0.51)	✓(0.54)	✓(0.54)	✓(0.53)	✓(0.53)	○(0.50)	○(0.50)	○(0.50)	○(0.51)	○(0.52)	○(0.52)	✓(0.53)	○(0.50)	✓(0.54)	✓(0.57)	✓(0.57)	✓(0.57)	✓(0.56)
IV3 *vs* IV4	○(0.49)	○(0.49)	○(0.48)	○(0.49)	○(0.49)	○(0.51)	○(0.52)	○(0.48)	○(0.48)	○(0.49)	✗(0.46)	✗(0.46)	○(0.49)	○(0.49)	○(0.49)	○(0.51)	○(0.50)	○(0.51)	○(0.50)	○(0.51)	✗(0.43)	✗(0.42)	✗(0.42)	✗(0.44)	✗(0.44)
IV3 *vs* IV5	○(0.49)	○(0.49)	○(0.50)	○(0.49)	○(0.49)	○(0.51)	○(0.52)	○(0.50)	○(0.50)	○(0.50)	○(0.48)	✗(0.47)	○(0.49)	○(0.49)	○(0.49)	○(0.50)	○(0.48)	○(0.49)	○(0.48)	○(0.49)	✗(0.47)	✗(0.47)	✗(0.47)	○(0.48)	○(0.48)
IV4 *vs* IV5	○(0.50)	○(0.51)	○(0.51)	○(0.50)	○(0.50)	○(0.51)	○(0.50)	○(0.52)	○(0.51)	○(0.52)	○(0.51)	○(0.51)	○(0.50)	○(0.50)	○(0.50)	○(0.49)	○(0.48)	○(0.48)	○(0.48)	✗(0.47)	✓(0.54)	✓(0.55)	✓(0.55)	✓(0.54)	✓(0.54)
IV1 *vs* PV	✗(0.26)	✗(0.24)	✗(0.29)	✗(0.26)	✗(0.26)	✗(0.43)	✗(0.42)	✗(0.42)	✗(0.37)	✗(0.37)	✗(0.41)	✗(0.41)	○(0.49)	○(0.49)	○(0.49)	✗(0.42)	✗(0.41)	✗(0.41)	✗(0.37)	✗(0.43)	✗(0.37)	✗(0.30)	✗(0.30)	✗(0.32)	✗(0.33)
IV2 *vs* PV	○(0.50)	○(0.51)	○(0.51)	○(0.50)	○(0.50)	○(0.49)	○(0.49)	○(0.49)	○(0.51)	○(0.50)	○(0.50)	○(0.50)	○(0.51)	○(0.51)	○(0.51)	○(0.48)	○(0.49)	○(0.50)	○(0.49)	○(0.49)	○(0.51)	○(0.52)	○(0.52)	✓(0.53)	✓(0.53)
IV3 *vs* PV	✗(0.46)	✗(0.45)	✗(0.46)	✗(0.45)	✗(0.45)	○(0.50)	○(0.50)	○(0.48)	✗(0.47)	✗(0.47)	✗(0.45)	✗(0.45)	○(0.50)	○(0.50)	○(0.50)	✗(0.47)	✗(0.46)	✗(0.46)	✗(0.44)	○(0.48)	✗(0.44)	✗(0.42)	✗(0.42)	✗(0.44)	✗(0.45)
IV4 *vs* PV	✗(0.46)	✗(0.46)	○(0.48)	✗(0.47)	✗(0.47)	○(0.49)	○(0.48)	○(0.50)	○(0.49)	○(0.48)	○(0.49)	○(0.49)	○(0.51)	○(0.51)	○(0.51)	✗(0.46)	✗(0.46)	✗(0.46)	✗(0.44)	✗(0.47)	○(0.51)	○(0.50)	○(0.50)	○(0.51)	○(0.51)
IV5 *vs* PV	✗(0.47)	✗(0.46)	✗(0.47)	✗(0.46)	✗(0.46)	○(0.48)	○(0.48)	○(0.48)	✗(0.47)	✗(0.47)	✗(0.47)	○(0.47)	○(0.51)	○(0.51)	○(0.51)	○(0.48)	○(0.48)	○(0.48)	✗(0.46)	○(0.49)	✗(0.47)	✗(0.45)	✗(0.45)	✗(0.47)	✗(0.47)

values range from 0.37 to 0.49, which means that their differences are relatively medium or small.

ii) When comparing IVx $(2 \leq x \leq 5)$ with PV, different IVx techniques have different observations. More specifically, apart from the programs *sed-v4* and *sed-v5*, there is no significant differences between IV2 and PV. Except for *grep* with the first three versions, *gzip* with the last three versions, and *make-v5*, the differences between IV3 and PV are highly significant. In addition, the APFD differences between IV4 and PV are highly significant for the programs *flex* (except for *flex-v3*) and *make*, but not for the programs *grep*, *gzip*, and *sed*, for all versions. Similarly, the differences between IV5 and PV are highly significant for all versions of the programs *flex* and *sed*, and some versions of other programs. Nevertheless, the effect size measure $\hat{A}_{12}(\text{IV}x, \text{PV})$ only ranges from 0.42 to 0.53, which indicates that the differences between them are small. As a consequence, we can conclude that, overall, the statistical analysis confirms the box plots results.

With respect to the rate of fault detection, the answer to **RQ1.2** is: RSLCP-PV performs significantly better than IV1, but has very similar performances to other RSLCP-IV techniques.

2) RQ2: RSLCP versus λLCP: Based on the experimental data comparing RSLCP and λLCP, we have the following observations:

a) The box plots distributions show that the differences between RSLCP and λLCP are relatively small, regardless of subject program. Both in terms of mean and median values, this maximum difference between RSLCP and λLCP is only approximately 3.5% to 5.0%.

b) Different programs may have different observations. For example, for all five versions of the program *flex*, all RSLCP techniques perform better than all λLCP

TABLE VIII
STATISTICAL ANALYSIS FOR PAIRWISE APFD COMPARISONS OF RSLCP AGAINST OTHER ATCP TECHNIQUES ($\hat{A}_{12}$ VALUES INCLUDED IN BRACKETS)

Comparison	*flex*					*grep*					*gzip*					*make*					*sed*				
	v1	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*	*v1*	*v2*	*v3*	*v4*	*v5*
IV1 *vs* 1W	✓(0.79)	✓(0.79)	✓(0.75)	✓(0.78)	✓(0.78)	✓(0.66)	✓(0.71)	○(0.51)	✓(0.66)	✓(0.60)	✓(0.57)	✓(0.57)	✗(0.47)	✗(0.47)	✗(0.47)	✓(0.57)	✗(0.40)	✗(0.39)	○(0.50)	✗(0.29)	○(0.49)	✓(0.58)	✓(0.58)	✓(0.59)	✓(0.64)
IV2 *vs* 1W	✓(0.92)	✓(0.94)	✓(0.89)	✓(0.92)	✓(0.92)	✓(0.72)	✓(0.76)	✓(0.57)	✓(0.76)	✓(0.71)	✓(0.65)	✓(0.65)	○(0.50)	○(0.50)	○(0.50)	✓(0.63)	✗(0.47)	✗(0.46)	✓(0.61)	✗(0.33)	✓(0.63)	✓(0.78)	✓(0.78)	✓(0.78)	✓(0.80)
IV3 *vs* 1W	✓(0.91)	✓(0.92)	✓(0.87)	✓(0.90)	✓(0.90)	✓(0.72)	✓(0.77)	✓(0.56)	✓(0.74)	✓(0.68)	✓(0.61)	✓(0.61)	○(0.49)	○(0.49)	○(0.49)	✓(0.61)	✗(0.44)	✗(0.43)	✓(0.57)	✗(0.32)	✓(0.56)	✓(0.70)	✓(0.70)	✓(0.71)	✓(0.74)
IV4 *vs* 1W	✓(0.91)	✓(0.92)	✓(0.87)	✓(0.91)	✓(0.90)	✓(0.72)	✓(0.76)	✓(0.58)	✓(0.75)	✓(0.69)	✓(0.64)	✓(0.64)	○(0.49)	○(0.49)	○(0.49)	✓(0.61)	✗(0.44)	✗(0.43)	✓(0.56)	✗(0.31)	✓(0.63)	✓(0.77)	✓(0.77)	✓(0.76)	✓(0.78)
IV5 *vs* 1W	✓(0.91)	✓(0.92)	✓(0.87)	✓(0.90)	✓(0.90)	✓(0.71)	✓(0.76)	✓(0.56)	✓(0.74)	✓(0.67)	✓(0.63)	✓(0.63)	○(0.49)	○(0.49)	○(0.49)	✓(0.62)	✗(0.46)	✗(0.44)	✓(0.58)	✗(0.33)	✓(0.59)	✓(0.73)	✓(0.73)	✓(0.72)	✓(0.75)
PV *vs* 1W	✓(0.92)	✓(0.94)	✓(0.88)	✓(0.92)	✓(0.92)	✓(0.72)	✓(0.78)	✓(0.58)	✓(0.76)	✓(0.70)	✓(0.65)	✓(0.65)	○(0.49)	○(0.49)	○(0.49)	✓(0.64)	✗(0.47)	✗(0.46)	✓(0.61)	✗(0.33)	✓(0.62)	✓(0.76)	✓(0.76)	✓(0.75)	✓(0.77)
IV1 *vs* 2W	✓(0.76)	✓(0.75)	✓(0.71)	✓(0.74)	✓(0.74)	✗(0.42)	✗(0.41)	✗(0.28)	✗(0.43)	✗(0.37)	○(0.49)	○(0.49)	○(0.48)	○(0.48)	○(0.48)	✓(0.55)	✗(0.39)	✗(0.38)	○(0.48)	✗(0.29)	✗(0.35)	✗(0.27)	✗(0.27)	✗(0.30)	✗(0.34)
IV2 *vs* 2W	✓(0.90)	✓(0.91)	✓(0.86)	✓(0.89)	✓(0.89)	○(0.49)	○(0.47)	✗(0.34)	✓(0.56)	○(0.49)	✓(0.57)	✓(0.57)	○(0.50)	○(0.50)	○(0.50)	✓(0.62)	✗(0.47)	✗(0.45)	✓(0.60)	✗(0.32)	○(0.49)	○(0.48)	○(0.48)	○(0.52)	✓(0.54)
IV3 *vs* 2W	✓(0.88)	✓(0.89)	✓(0.84)	✓(0.87)	✓(0.87)	○(0.49)	○(0.49)	✗(0.33)	○(0.52)	✗(0.46)	○(0.52)	○(0.52)	○(0.49)	○(0.49)	○(0.49)	✓(0.60)	✗(0.44)	✗(0.42)	✓(0.56)	✗(0.31)	✗(0.41)	✗(0.39)	✗(0.39)	✗(0.43)	✗(0.46)
IV4 *vs* 2W	✓(0.89)	✓(0.89)	✓(0.85)	✓(0.88)	✓(0.88)	○(0.48)	✗(0.47)	✗(0.35)	✓(0.53)	✗(0.47)	✓(0.56)	✓(0.56)	○(0.50)	○(0.50)	○(0.50)	✓(0.59)	✗(0.44)	✗(0.42)	✓(0.56)	✗(0.30)	○(0.48)	✗(0.46)	✗(0.46)	○(0.49)	○(0.51)
IV5 *vs* 2W	✓(0.89)	✓(0.89)	✓(0.84)	✓(0.88)	✓(0.88)	○(0.47)	✗(0.47)	✗(0.33)	○(0.52)	✗(0.45)	✓(0.54)	✓(0.55)	○(0.50)	○(0.50)	○(0.50)	✓(0.61)	✗(0.46)	✗(0.44)	✓(0.58)	✗(0.32)	✗(0.45)	✗(0.42)	✗(0.42)	✗(0.45)	○(0.48)
PV *vs* 2W	✓(0.90)	✓(0.91)	✓(0.86)	✓(0.89)	✓(0.89)	○(0.49)	○(0.49)	✗(0.35)	✓(0.55)	○(0.48)	✓(0.57)	✓(0.57)	○(0.49)	○(0.49)	○(0.49)	✓(0.63)	✗(0.47)	✗(0.45)	✓(0.61)	✗(0.33)	✗(0.47)	✗(0.46)	✗(0.46)	○(0.48)	○(0.51)
IV1 *vs* 3W	✓(0.74)	✓(0.71)	✓(0.70)	✓(0.72)	✓(0.72)	✗(0.41)	✗(0.40)	✗(0.33)	✗(0.36)	✗(0.33)	✗(0.38)	✗(0.38)	○(0.47)	○(0.47)	○(0.47)	○(0.50)	✗(0.43)	✗(0.40)	○(0.48)	✗(0.32)	✗(0.31)	✗(0.15)	✗(0.15)	✗(0.22)	✗(0.23)
IV2 *vs* 3W	✓(0.90)	✓(0.90)	✓(0.85)	✓(0.89)	✓(0.88)	✗(0.47)	✗(0.47)	✗(0.40)	○(0.49)	✗(0.46)	✗(0.46)	✗(0.46)	○(0.50)	○(0.50)	○(0.50)	✓(0.57)	○(0.51)	○(0.48)	✓(0.60)	✗(0.36)	✗(0.45)	✗(0.33)	✗(0.33)	✗(0.42)	✗(0.41)
IV3 *vs* 3W	✓(0.88)	✓(0.87)	✓(0.82)	✓(0.86)	✓(0.86)	○(0.48)	○(0.48)	✗(0.39)	✗(0.45)	✗(0.42)	✗(0.41)	✗(0.41)	○(0.49)	○(0.49)	○(0.49)	✓(0.55)	○(0.48)	✗(0.45)	✓(0.56)	✗(0.36)	✗(0.38)	✗(0.24)	✗(0.24)	✗(0.33)	✗(0.33)
IV4 *vs* 3W	✓(0.88)	✓(0.88)	✓(0.83)	✓(0.87)	✓(0.87)	✗(0.47)	✗(0.47)	✗(0.41)	✗(0.47)	✗(0.44)	✗(0.44)	✗(0.44)	○(0.50)	○(0.50)	○(0.50)	✓(0.54)	○(0.48)	✗(0.45)	✓(0.56)	✗(0.34)	✗(0.45)	✗(0.32)	✗(0.32)	✗(0.39)	✗(0.39)
IV5 *vs* 3W	✓(0.88)	✓(0.88)	✓(0.82)	✓(0.87)	✓(0.86)	✗(0.46)	✗(0.47)	✗(0.39)	✗(0.45)	✗(0.42)	✗(0.43)	✗(0.43)	○(0.50)	○(0.50)	○(0.50)	✓(0.56)	○(0.49)	✗(0.47)	✓(0.58)	✗(0.37)	✗(0.41)	✗(0.27)	✗(0.27)	✗(0.36)	✗(0.35)
PV *vs* 3W	✓(0.90)	✓(0.90)	✓(0.84)	✓(0.89)	✓(0.88)	○(0.48)	○(0.48)	✗(0.41)	○(0.48)	✗(0.45)	✗(0.45)	✗(0.45)	○(0.49)	○(0.49)	○(0.49)	✓(0.58)	○(0.51)	○(0.49)	✓(0.61)	✗(0.37)	✗(0.44)	✗(0.32)	✗(0.32)	✗(0.39)	✗(0.38)
IV1 *vs* 4W	✓(0.64)	✓(0.62)	✓(0.59)	✓(0.63)	✓(0.63)	✗(0.43)	✗(0.43)	✗(0.34)	✗(0.30)	✗(0.27)	✗(0.40)	✗(0.40)	○(0.48)	○(0.48)	○(0.48)	✗(0.47)	✗(0.45)	✗(0.45)	○(0.51)	✗(0.39)	✗(0.29)	✗(0.12)	✗(0.12)	✗(0.18)	✗(0.19)
IV2 *vs* 4W	✓(0.83)	✓(0.84)	✓(0.76)	✓(0.82)	✓(0.82)	○(0.50)	○(0.50)	✗(0.40)	✗(0.43)	✗(0.38)	○(0.48)	○(0.48)	○(0.50)	○(0.50)	○(0.50)	✓(0.53)	✓(0.54)	✓(0.54)	✓(0.65)	✗(0.45)	✗(0.43)	✗(0.29)	✗(0.29)	✗(0.39)	✗(0.37)
IV3 *vs* 4W	✓(0.81)	✓(0.80)	✓(0.73)	✓(0.79)	✓(0.79)	○(0.50)	○(0.51)	✗(0.39)	✗(0.40)	✗(0.35)	✗(0.43)	✗(0.43)	○(0.49)	○(0.49)	○(0.49)	○(0.52)	○(0.51)	○(0.51)	✓(0.60)	✗(0.44)	✗(0.36)	✗(0.21)	✗(0.21)	✗(0.30)	✗(0.30)
IV4 *vs* 4W	✓(0.81)	✓(0.81)	✓(0.74)	✓(0.80)	✓(0.80)	○(0.49)	○(0.50)	✗(0.41)	✗(0.41)	✗(0.37)	✗(0.46)	✗(0.46)	○(0.50)	○(0.50)	○(0.50)	○(0.51)	○(0.51)	○(0.50)	✓(0.60)	✗(0.42)	✗(0.43)	✗(0.29)	✗(0.29)	✗(0.36)	✗(0.35)
IV5 *vs* 4W	✓(0.81)	✓(0.80)	✓(0.73)	✓(0.80)	✓(0.79)	○(0.49)	○(0.50)	✗(0.39)	✗(0.40)	✗(0.35)	✗(0.45)	✗(0.45)	○(0.50)	○(0.50)	○(0.50)	○(0.52)	✓(0.53)	○(0.52)	✓(0.63)	✗(0.45)	✗(0.39)	✗(0.24)	✗(0.24)	✗(0.32)	✗(0.31)
PV *vs* 4W	✓(0.83)	✓(0.83)	✓(0.76)	✓(0.82)	✓(0.82)	○(0.50)	○(0.51)	✗(0.41)	✗(0.43)	✗(0.38)	✗(0.47)	✗(0.47)	○(0.49)	○(0.49)	○(0.49)	✓(0.55)	✓(0.55)	✓(0.54)	✓(0.66)	✗(0.45)	✗(0.42)	✗(0.29)	✗(0.29)	✗(0.36)	✗(0.35)
IV1 *vs* 5W	✓(0.57)	✓(0.56)	✓(0.57)	✓(0.56)	✓(0.56)	✗(0.35)	✗(0.33)	✗(0.28)	✗(0.25)	✗(0.21)	✗(0.38)	✗(0.38)	○(0.49)	○(0.49)	○(0.49)	✗(0.44)	✗(0.39)	✗(0.39)	✗(0.41)	✗(0.37)	✗(0.30)	✗(0.12)	✗(0.12)	✗(0.18)	✗(0.18)
IV2 *vs* 5W	✓(0.78)	✓(0.79)	✓(0.74)	✓(0.77)	✓(0.77)	✗(0.41)	✗(0.40)	✗(0.34)	✗(0.38)	✗(0.32)	✗(0.45)	✗(0.45)	○(0.51)	○(0.51)	○(0.51)	○(0.52)	○(0.48)	○(0.49)	✓(0.56)	✗(0.42)	✗(0.44)	✗(0.29)	✗(0.29)	✗(0.39)	✗(0.37)
IV3 *vs* 5W	✓(0.75)	✓(0.75)	✓(0.71)	✓(0.74)	✓(0.74)	✗(0.42)	✗(0.41)	✗(0.33)	✗(0.34)	✗(0.29)	✗(0.41)	✗(0.40)	○(0.50)	○(0.50)	○(0.50)	○(0.50)	✗(0.45)	✗(0.45)	○(0.50)	✗(0.41)	✗(0.36)	✗(0.21)	✗(0.21)	✗(0.29)	✗(0.29)
IV4 *vs* 5W	✓(0.76)	✓(0.76)	✓(0.72)	✓(0.75)	✓(0.74)	✗(0.41)	✗(0.39)	✗(0.35)	✗(0.36)	✗(0.31)	✗(0.44)	✗(0.44)	○(0.51)	○(0.51)	○(0.51)	○(0.49)	✗(0.45)	✗(0.44)	○(0.50)	✗(0.40)	✗(0.44)	✗(0.28)	✗(0.28)	✗(0.36)	✗(0.34)
IV5 *vs* 5W	✓(0.76)	✓(0.76)	✓(0.71)	✓(0.74)	✓(0.74)	✗(0.40)	✗(0.39)	✗(0.33)	✗(0.34)	✗(0.29)	✗(0.43)	✗(0.43)	○(0.51)	○(0.51)	○(0.51)	○(0.51)	✗(0.47)	✗(0.46)	✓(0.53)	✗(0.42)	✗(0.40)	✗(0.24)	✗(0.24)	✗(0.32)	✗(0.30)
PV *vs* 5W	✓(0.78)	✓(0.79)	✓(0.74)	✓(0.77)	✓(0.77)	✗(0.42)	✗(0.41)	✗(0.35)	✗(0.37)	✗(0.32)	✗(0.45)	✗(0.44)	○(0.50)	○(0.50)	○(0.50)	✓(0.53)	○(0.49)	○(0.49)	✓(0.57)	✗(0.43)	✗(0.43)	✗(0.28)	✗(0.28)	✗(0.35)	✗(0.34)
IV1 *vs* 6W	○(0.50)	○(0.51)	○(0.51)	○(0.48)	○(0.49)	✗(0.36)	✗(0.32)	✗(0.28)	✗(0.27)	✗(0.23)	✗(0.34)	✗(0.34)	○(0.50)	○(0.50)	○(0.50)	✗(0.45)	✗(0.33)	✗(0.33)	✗(0.37)	✗(0.31)	✗(0.30)	✗(0.13)	✗(0.13)	✗(0.18)	✗(0.19)
IV2 *vs* 6W	✓(0.74)	✓(0.77)	✓(0.71)	✓(0.72)	✓(0.72)	✗(0.43)	✗(0.39)	✗(0.34)	✗(0.39)	✗(0.34)	✗(0.42)	✗(0.42)	○(0.52)	○(0.52)	○(0.52)	○(0.52)	✗(0.41)	✗(0.41)	○(0.52)	✗(0.35)	✗(0.44)	✗(0.31)	✗(0.31)	✗(0.39)	✗(0.38)
IV3 *vs* 6W	✓(0.70)	✓(0.72)	✓(0.67)	✓(0.68)	✓(0.68)	✗(0.43)	✗(0.40)	✗(0.34)	✗(0.35)	✗(0.31)	✗(0.37)	✗(0.37)	○(0.51)	○(0.51)	○(0.51)	○(0.51)	✗(0.38)	✗(0.38)	✗(0.46)	✗(0.35)	✗(0.37)	✗(0.23)	✗(0.23)	✗(0.30)	✗(0.30)
IV4 *vs* 6W	✓(0.71)	✓(0.73)	✓(0.69)	✓(0.70)	✓(0.69)	✗(0.42)	✗(0.39)	✗(0.35)	✗(0.37)	✗(0.32)	✗(0.40)	✗(0.41)	○(0.52)	○(0.52)	○(0.52)	○(0.50)	✗(0.38)	✗(0.37)	✗(0.45)	✗(0.33)	✗(0.44)	✗(0.30)	✗(0.30)	✗(0.36)	✗(0.36)
IV5 *vs* 6W	✓(0.71)	✓(0.73)	✓(0.67)	✓(0.69)	✓(0.69)	✗(0.41)	✗(0.38)	✗(0.33)	✗(0.36)	✗(0.31)	✗(0.39)	✗(0.39)	○(0.52)	○(0.52)	○(0.52)	○(0.51)	✗(0.40)	✗(0.39)	○(0.49)	✗(0.36)	✗(0.40)	✗(0.25)	✗(0.25)	✗(0.32)	✗(0.32)
PV *vs* 6W	✓(0.74)	✓(0.76)	✓(0.70)	✓(0.72)	✓(0.72)	✗(0.43)	✗(0.40)	✗(0.35)	✗(0.38)	✗(0.34)	✗(0.41)	✗(0.41)	○(0.51)	○(0.51)	○(0.51)	✓(0.54)	✗(0.42)	✗(0.42)	✓(0.53)	✗(0.36)	✗(0.43)	✗(0.30)	✗(0.30)	✗(0.36)	✗(0.35)
IV1 *vs* ILCP	○(0.52)	○(0.50)	○(0.51)	○(0.51)	○(0.50)	✗(0.35)	✗(0.34)	✗(0.30)	✗(0.27)	✗(0.24)	✗(0.40)	✗(0.40)	○(0.48)	○(0.48)	○(0.48)	✗(0.42)	✗(0.33)	✗(0.33)	✗(0.34)	✗(0.32)	✗(0.32)	✗(0.16)	✗(0.16)	✗(0.21)	✗(0.22)
IV2 *vs* ILCP	✓(0.77)	✓(0.77)	✓(0.70)	✓(0.75)	✓(0.75)	✗(0.42)	✗(0.41)	✗(0.36)	✗(0.40)	✗(0.35)	○(0.48)	○(0.48)	○(0.51)	○(0.51)	○(0.51)	○(0.49)	✗(0.41)	✗(0.41)	○(0.48)	✗(0.36)	✗(0.45)	✗(0.35)	✗(0.35)	✗(0.42)	✗(0.42)
IV3 *vs* ILCP	✓(0.73)	✓(0.72)	✓(0.67)	✓(0.71)	✓(0.71)	✗(0.42)	✗(0.42)	✗(0.35)	✗(0.37)	✗(0.32)	✗(0.43)	✗(0.43)	○(0.50)	○(0.50)	○(0.50)	○(0.48)	✗(0.38)	✗(0.39)	✗(0.43)	✗(0.36)	✗(0.38)	✗(0.26)	✗(0.26)	✗(0.33)	✗(0.33)
IV4 *vs* ILCP	✓(0.74)	✓(0.73)	✓(0.68)	✓(0.72)	✓(0.72)	✗(0.41)	✗(0.41)	✗(0.37)	✗(0.38)	✗(0.33)	✗(0.46)	✗(0.46)	○(0.50)	○(0.50)	○(0.50)	✗(0.46)	✗(0.38)	✗(0.38)	✗(0.42)	✗(0.34)	✗(0.45)	✗(0.34)	✗(0.34)	✗(0.39)	✗(0.39)
IV5 *vs* ILCP	✓(0.74)	✓(0.73)	✓(0.67)	✓(0.72)	✓(0.71)	✗(0.40)	✗(0.40)	✗(0.35)	✗(0.37)	✗(0.32)	✗(0.45)	✗(0.45)	○(0.51)	○(0.51)	○(0.51)	○(0.48)	✗(0.40)	✗(0.40)	✗(0.45)	✗(0.37)	✗(0.41)	✗(0.29)	✗(0.29)	✗(0.35)	✗(0.35)
PV *vs* ILCP	✓(0.77)	✓(0.76)	✓(0.69)	✓(0.75)	✓(0.75)	✗(0.42)	✗(0.42)	✗(0.37)	✗(0.39)	✗(0.35)	✗(0.47)	✗(0.47)	○(0.50)	○(0.50)	○(0.50)	○(0.50)	✗(0.42)	✗(0.42)	○(0.50)	✗(0.37)	✗(0.44)	✗(0.34)	✗(0.34)	✗(0.39)	✗(0.39)
IV1 *vs* GSP	✗(0.45)	✗(0.24)	✓(0.67)	✗(0.45)	✗(0.45)	✓(0.65)	○(0.48)	✓(0.72)	✓(0.66)	✓(0.62)	○(0.48)	✗(0.47)	✗(0.44)	✗(0.44)	✗(0.44)	✗(0.17)	✗(0.14)	✗(0.12)	✗(0.07)	✗(0.16)	✗(0.46)	✓(0.73)	✓(0.73)	✓(0.62)	✗(0.34)
IV2 *vs* GSP	✓(0.75)	✓(0.56)	✓(0.89)	✓(0.76)	✓(0.75)	✓(0.71)	✓(0.55)	✓(0.78)	✓(0.78)	✓(0.75)	✓(0.57)	✓(0.56)	✗(0.47)	✗(0.47)	✗(0.47)	✗(0.18)	✗(0.16)	✗(0.13)	✗(0.10)	✗(0.16)	✓(0.62)	✓(0.90)	✓(0.90)	✓(0.82)	✓(0.56)
IV3 *vs* GSP	✓(0.71)	○(0.49)	✓(0.86)	✓(0.71)	✓(0.71)	✓(0.72)	✓(0.57)	✓(0.77)	✓(0.74)	✓(0.71)	○(0.52)	○(0.51)	✗(0.46)	✗(0.46)	✗(0.46)	✗(0.18)	✗(0.15)	✗(0.12)	✗(0.08)	✗(0.16)	✓(0.54)	✓(0.84)	✓(0.84)	✓(0.75)	○(0.48)
IV4 *vs* GSP	✓(0.72)	○(0.50)	✓(0.87)	✓(0.72)	✓(0.72)	✓(0.71)	✓(0.55)	✓(0.78)	✓(0.76)	✓(0.72)	✓(0.55)	✓(0.55)	✗(0.46)	✗(0.46)	✗(0.46)	✗(0.17)	✗(0.15)	✗(0.11)	✗(0.08)	✗(0.15)	✓(0.62)	✓(0.89)	✓(0.89)	✓(0.80)	✓(0.54)
IV5 *vs* GSP	✓(0.71)	○(0.50)	✓(0.85)	✓(0.71)	✓(0.71)	✓(0.71)	✓(0.55)	✓(0.77)	✓(0.74)	✓(0.71)	✓(0.54)	✓(0.53)	✗(0.46)	✗(0.46)	✗(0.46)	✗(0.18)	✗(0.16)	✗(0.13)	✗(0.08)	✗(0.17)	✓(0.58)	✓(0.86)	✓(0.86)	✓(0.77)	○(0.49)
PV *vs* GSP	✓(0.76)	✓(0.55)	✓(0.88)	✓(0.76)	✓(0.75)	✓(0.72)	✓(0.57)	✓(0.78)	✓(0.77)	✓(0.74)	✓(0.57)	✓(0.56)	✗(0.46)	✗(0.46)	✗(0.46)	✗(0.18)	✗(0.18)	✗(0.13)	✗(0.11)	✗(0.16)	✓(0.61)	✓(0.88)	✓(0.88)	✓(0.79)	✓(0.53)
IV1 *vs* LSP	✗(0.44)	✗(0.34)	✓(0.69)	✗(0.45)	✗(0.45)	✓(0.81)	✓(0.74)	✓(0.70)	✓(0.89)	✓(0.89)	✓(0.59)	✓(0.59)	✓(0.70)	✓(0.70)	✓(0.70)	✗(0.27)	✗(0.10)	✗(0.09)	✗(0.06)	✗(0.09)	✗(0.38)	✓(0.83)	✓(0.83)	✓(0.59)	○(0.49)
IV2 *vs* LSP	✓(0.71)	✓(0.65)	✓(0.87)	✓(0.72)	✓(0.72)	✓(0.85)	✓(0.79)	✓(0.76)	✓(0.94)	✓(0.94)	✓(0.67)	✓(0.67)	✓(0.72)	✓(0.72)	✓(0.72)	✗(0.31)	✗(0.11)	✗(0.09)	✗(0.09)	✗(0.09)	✓(0.53)	✓(0.95)	✓(0.95)	✓(0.81)	✓(0.69)
IV3 *vs* LSP	✓(0.67)	✓(0.59)	✓(0.85)	✓(0.68)	✓(0.68)	✓(0.85)	✓(0.80)	✓(0.75)	✓(0.93)	✓(0.93)	✓(0.63)	✓(0.62)	✓(0.71)	✓(0.71)	✓(0.71)	✗(0.31)	✗(0.10)	✗(0.09)	✗(0.08)	✗(0.09)	✗(0.45)	✓(0.92)	✓(0.92)	✓(0.73)	✓(0.61)
IV4 *vs* LSP	✓(0.68)	✓(0.60)	✓(0.85)	✓(0.69)	✓(0.69)	✓(0.85)	✓(0.79)	✓(0.76)	✓(0.94)	✓(0.93)	✓(0.66)	✓(0.66)	✓(0.71)	✓(0.71)	✓(0.71)	✗(0.29)	✗(0.10)	✗(0.08)	✗(0.07)	✗(0.08)	✓(0.53)	✓(0.95)	✓(0.95)	✓(0.79)	✓(0.66)
IV5 *vs* LSP	✓(0.68)	✓(0.60)	✓(0.84)	✓(0.68)	✓(0.68)	✓(0.85)	✓(0.79)	✓(0.75)	✓(0.93)	✓(0.92)	✓(0.65)	✓(0.65)	✓(0.71)	✓(0.71)	✓(0.71)	✗(0.30)	✗(0.11)	✗(0.09)	✗(0.07)	✗(0.10)	○(0.49)	✓(0.93)	✓(0.93)	✓(0.75)	✓(0.63)
PV *vs* LSP	✓(0.71)	✓(0.64)	✓(0.87)	✓(0.72)	✓(0.72)	✓(0.86)	✓(0.81)	✓(0.76)	✓(0.94)	✓(0.94)	✓(0.67)	✓(0.67)	✓(0.71)	✓(0.71)	✓(0.71)	✗(0.32)	✗(0.12)	✗(0.10)	✗(0.10)	✗(0.09)	○(0.52)	✓(0.94)	✓(0.94)	✓(0.78)	✓(0.66)

techniques; however, for the program *sed*, all λLCP techniques (except 1W and 2W for some cases) have higher mean and median APFD values than the RSLCP techniques. Nevertheless, the RSLCP techniques generally have comparable or better APFD performances than 1W and 2W, and comparable performances to 3W and 4W. Compared with 5W and 6W, overall RSLCP has the worse rates of fault detection; however, it can sometimes have significantly better performances.

c) In general, the statistical analysis supports the observations above. It should be noted that for the program *flex*, the difference between RSLCP and λLCP has a large effect size in most cases (except for the comparisons of IV1 with 4W, 5W, and 6W)—with the effect size values ranging from 67% to 94%. However, for the program *sed*, the case is generally the opposite: apart from 1W and 2W, all other λLCP techniques generally have significantly better performances than RSLCP, either with a large or medium effect size.

From the perspective of the fault detection rate, the answer to **RQ2** is: Although RSLCP and λLCP have similar APFD observations, overall, RSLCP is comparable or more effective than λLCP with low λ values (1 or 2), and has comparable (and sometimes superior) APFDs to λLCP with medium λ values such as 3 and 4. In addition, RSLCP is worse than λLCP with high λ values such as 5 and 6, but it sometimes has much better fault detection rates.

3) RQ3: RSLCP versus ILCP and SP: Based on the experimental data comparing RSLCP, ILCP, and SP, it can be seen that:

a) Compared with ILCP, although RSLCP has relative similar mean and median APFD values, it overall has better rates of fault detection for the program *flex* but worse performances for the other four programs, irrespective of versions. The statistical analysis supports these box plot observations.

b) Compared with GSP, except for the program *make*, and some versions of the program *gzip*, RSLCP generally has higher APFDs, both in terms of mean and median values. The statistical analysis shows that the differences between GSP and RSLCP are highly significant, because their p-values are generally less than 0.05; and the effect size $\hat{A}_{12}$ values are within a high or medium degree (except for the program *gzip*).

c) Compared with LSP, except for the program *make*, RSLCP generally has better performances in most cases, regardless of mean and median APFD values. This observation is supported by the statistical analysis, i.e., nearly all the p values are less than 0.05; and most effect size $\hat{A}_{12}$ values are within a high degree.

TABLE IX
PRIORITIZATION TIME (IN MILLISECONDS) REPRESENTED BY MEAN/STANDARD DEVIATION (μ/σ)

Subject	RSLCP						λLCP						ILCP	SP	
	IV1	IV2	IV3	IV4	IV5	PV	1W	2W	3W	4W	5W	6W		GSP	LSP
flex	40/8	100/8	111/6	111/5	111/5	109/6	39/11	97/14	236/27	399/11	460/12	327/17	1,602/36	1,321/19	1,299/20
grep	32/4	86/8	94/5	94/5	93/6	92/6	29/5	81/6	201/7	370/22	472/12	382/23	1,357/25	1,012/14	999/22
gzip	15/9	68/15	63/7	59/7	61/7	61/6	15/9	72/13	248/8	819/10	1822/17	2,962/37	15,540/93	175/7	176/10
make	6/8	20/7	35/7	35/6	34/6	34/6	6/8	20/7	64/6	126/6	165/8	158/12	589/13	69/8	69/8
sed	23/8	79/4	87/8	88/8	87/8	86/8	21/8	77/6	264/8	618/15	967/15	1,043/11	3,691/146	627/9	624/8
$\sum$	116/–	353/–	390/–	387/–	386/–	382/–	110/–	347/–	1,013/–	2,332/–	3,886/–	4,872/–	22,779/–	3,204/–	3,167/–

From the perspective of fault detection rate, the answer to **RQ3** is: RSLCP generally performs worse than ILCP, but can sometimes achieve a better performance. Compared with SP (both GSP and LSP), RSLCP is generally more effective, although it may be less effective in some cases.

As discussed in Section V-A, RSLCP is faster at covering 1-wise or 2-wise level combinations than λLCP. In addition, as shown in Table IV, the proportion of faults with the FTFI number being 1 or 2 is about 27% to 30% for the program *flex*; approximately 30% to 40% for *grep*; 75% to 100% for *gzip*; and 45% to 60% for the program *sed*. In other words, RSLCP may have better or at least similar fault detection rates to λLCP for these four programs. However, as discussed in the APFD results, RSLCP is more effective than λLCP for the program *flex*, but overall less effective for the programs *grep*, *gzip*, and *sed*. This phenomenon may be explained as follows: 1) λLCP could achieve the rates of covering 1-wise and 2-wise level combinations to some extent, which may detect faults with the FTFI number 1 and 2 as quickly as RSLCP. 2) Even though two faults may have the same FTFI number, they may have very different properties. For example, consider two faults f_1 and f_2 with the FTFI number equal to 2, which means that both of them could be caused by two parameters p_i and p_j from the input parameter model $Model(P, L, \mathcal{Q})$, i.e., $p_i, p_j \in P$, and the corresponding levels are $L_i \in L$ and $L_j \in L$. The fault f_1 may be triggered by the combination of $(p_i = l_i)$ and $(p_j = l_j)$, where $l_i \in L_i$ and $l_j \in L_j$; while the fault may be caused by the combination of $(p_i \neq l_i)$ and $(p_j \neq l_j)$. Therefore, the probability of detecting f_1 is equal to $\frac{1}{|L_i|\times|L_j|}$; and the probability of detecting f_2 is $\frac{(|L_i|-1)\times(|L_j|-1)}{|L_i|\times|L_j|}$. In other words, the number of ATCs required to detect f_1 is much fewer than that to detect f_2, especially when the sizes of L_i and L_j are large. As a consequence, a higher rate of covering 1-wise or 2-wise level combinations does not always imply a faster detection of 1-wise or 2-wise faults.

A possible reason to explain why RSLCP has significantly worse APFD performance than SP for the program *make* may be: As discussed in Section V-A3, SP first chooses the *best* pair of elements as the first two ATCs that cover the largest number of 1-wise level combinations. When there is only one *best* pair (i.e., not a tie-breaking situation), the first two ATCs are deterministic. If two such ATCs could detect many faults, then SP could certainly achieve higher rates of fault detection than RSLCP. Manual inspection of the ATC set and mutants for the program *make* has confirmed this to be the case.

C. Prioritization Cost Results

In this section, we answer **RQ1**, **RQ2**, and **RQ3**, from the perspective of the prioritization cost. Table IX shows the prioritization time, in milliseconds, for the RSLCP, λLCP, ILCP, and SP techniques. The table presents the mean prioritization time (μ) and the standard deviation (σ) over the 1000 independent runs performed per technique.

1) RQ1: RSLCP Techniques: From the table, it can be clearly observed that among the six RSLCP techniques, IV1 has the fastest prioritization, with the other RSLCP techniques all showing similar prioritization costs. Therefore, the answer to **RQ1.1** is that IV1 is most efficient among RSLCP-IV techniques; while others are comparable to each other. Similarly, the answer to **RQ1.2** is: IV1 is more efficient than PV; while other RSLCP-IV techniques are similar to RSLCP-PV.

2) RQ2: RSLCP versus λLCP: Compared with λLCP, IV1 requires a very similar time to 1W, with the other RSLCP techniques (IV2, IV3, IV4, IV5, and PV) taking a very similar amount of time to 2W. In addition, all the RSLCP techniques require considerably less time than 3W, 4W, 5W, and 6W. From the perspective of prioritization cost, therefore, the answer to **RQ2** is: RSLCP has similarly efficiency to λLCP when λ is small (such as 1 and 2), and requires much less time when λ is larger (such as 3, 4, 5, and 6).

3) RQ3: RSLCP versus ILCP and SP: Compared with ILCP, each RSLCP technique requires significantly less prioritization time, i.e., the mean (μ) prioritization time of ILCP ranges from 589 to 15 540 ms; however, the mean time of RSLCP ranges from only 6 to 111 ms. Similarly, RSLCP also needs much less time than SP (both GSP and LSP) to prioritize ATCs. Therefore, the answer to **RQ3** is: The RSLCP techniques are much more efficient than ILCP and SP.

D. Summary

By combining and summarizing the observations from the experimental results of interaction coverage rates, the fault detection rates, and the prioritization time, we have the following conclusions:

1) Compared with λLCP with low λ values (such as 1 and 2), RSLCP achieves superior or at least comparable testing effectiveness, measured with APCC and APFD, while maintaining testing efficiency.
2) Compared with λLCP with medium λ values (such as 3 and 4), RSLCP achieves comparable fault detection

TABLE X
MEAN AND MEDIAN APFD VALUES OVER VERSIONS FOR EACH RSLCP TECHNIQUE

Subject	Version	Mean						Median					
		IV1	IV2	IV3	IV4	IV5	PV	IV1	IV2	IV3	IV4	IV5	PV
flex	*v1*	85.13%	87.06%	86.74%	86.80%	86.82%	87.04%	85.13%	87.10%	86.76%	86.81%	86.85%	87.04%
	v2	86.08%	88.25%	87.84%	87.93%	87.88%	88.16%	86.12%	88.28%	87.81%	87.93%	87.96%	88.22%
	v3	87.71%	89.60%	89.25%	89.39%	89.26%	89.53%	87.75%	89.53%	89.26%	89.46%	89.37%	89.58%
	v4	86.08%	87.96%	87.63%	87.72%	87.68%	87.94%	86.09%	87.97%	87.60%	87.73%	87.76%	88.01%
	v5	85.73%	87.62%	87.30%	87.38%	87.35%	87.61%	85.76%	87.63%	87.27%	87.39%	87.42%	87.67%
Max - Min		2.58%	2.54%	2.51%	2.59%	2.44%	2.49%	2.62%	2.43%	2.50%	2.65%	2.52%	2.54%
grep	*v1*	87.72%	88.03%	88.04%	88.02%	87.98%	88.06%	87.72%	88.06%	88.08%	87.97%	87.98%	88.06%
	v2	86.49%	86.83%	86.90%	86.81%	86.81%	86.90%	86.51%	86.83%	86.93%	86.79%	86.83%	86.87%
	v3	86.45%	86.84%	86.78%	86.87%	86.75%	86.87%	86.45%	86.86%	86.80%	86.86%	86.76%	86.93%
	v4	84.86%	85.56%	85.36%	85.46%	85.37%	85.53%	84.87%	85.64%	85.43%	85.44%	85.41%	85.52%
	v5	83.48%	84.19%	84.01%	84.09%	83.99%	84.18%	83.45%	84.24%	84.07%	84.10%	83.97%	84.15%
Max - Min		4.24%	3.84%	4.03%	3.93%	3.99%	3.88%	4.27%	3.82%	4.01%	3.87%	4.01%	3.91%
gzip	*v1*	94.22%	94.99%	94.58%	94.84%	94.76%	94.99%	94.55%	95.27%	94.79%	95.15%	95.03%	95.27%
	v2	94.18%	94.96%	94.55%	94.82%	94.74%	94.96%	94.55%	95.19%	94.71%	95.11%	95.03%	95.19%
	v3	98.33%	98.36%	98.35%	98.36%	98.36%	98.35%	98.31%	98.40%	98.40%	98.40%	98.40%	98.40%
	v4	98.33%	98.36%	98.35%	98.36%	98.36%	98.35%	98.31%	98.40%	98.40%	98.40%	98.40%	98.40%
	v5	98.33%	98.36%	98.35%	98.36%	98.36%	98.35%	98.31%	98.40%	98.40%	98.40%	98.40%	98.40%
Max - Min		4.15%	3.40%	3.80%	3.54%	3.62%	3.39%	3.76%	3.21%	3.69%	3.29%	3.37%	3.21%
make	*v1*	53.49%	54.18%	54.01%	53.92%	54.06%	54.28%	53.42%	54.04%	54.09%	53.88%	53.98%	54.36%
	v2	50.43%	51.60%	51.16%	51.18%	51.38%	51.66%	50.28%	51.54%	51.04%	51.15%	51.43%	51.65%
	v3	52.01%	53.16%	52.75%	52.67%	52.95%	53.21%	51.96%	53.09%	52.82%	52.78%	52.93%	53.11%
	v4	53.04%	54.62%	54.08%	54.04%	54.29%	54.76%	52.98%	54.61%	54.16%	54.01%	54.35%	54.78%
	v5	50.14%	50.99%	50.78%	50.64%	50.97%	51.08%	50.10%	50.84%	50.74%	50.64%	50.84%	51.03%
Max - Min		3.35%	3.63%	3.30%	3.40%	3.32%	3.68%	3.32%	3.77%	3.42%	3.37%	3.51%	3.75%
sed	*v1*	85.19%	86.86%	86.05%	86.86%	86.42%	86.73%	85.12%	86.97%	86.13%	87.01%	86.39%	86.92%
	v2	88.14%	90.15%	89.34%	90.03%	89.62%	89.97%	88.17%	90.16%	89.34%	90.09%	89.68%	90.14%
	v3	88.14%	90.15%	89.34%	90.03%	89.62%	89.97%	88.17%	90.16%	89.34%	90.09%	89.68%	90.14%
	v4	87.03%	89.08%	88.28%	88.86%	88.47%	88.78%	87.01%	89.17%	88.28%	88.86%	88.52%	88.90%
	v5	88.75%	90.27%	89.72%	90.13%	89.84%	90.05%	88.83%	90.37%	89.75%	90.25%	89.97%	90.18%
Max - Min		3.56%	3.41%	3.67%	3.27%	3.42%	3.32%	3.71%	3.40%	3.62%	3.24%	3.58%	3.26%

rates but has worse interaction coverage rates. However, RSLCP also requires less prioritization time.

3) Compared with λLCP with high λ values (such as 5 and 6), RSLCP usually achieves worse (though sometimes comparable or better) rates of interaction coverage and fault detection, while maintaining much better testing efficiency.
4) Compared with ILCP, similar to λLCP with high λ values, the testing effectiveness of RSLCP is generally worse than that of ILCP (although sometimes RSLCP is better or comparable to ILCP), but RSLCP is much more efficient than ILCP.
5) Compared with SP, RSLCP not only provides better testing effectiveness in most cases, but is also more efficient.

As expected based on the discussion in Section III-D, overall, RSLCP provides a good tradeoff between testing effectiveness and efficiency.

E. Robustness Across Versions

This section attempts to answer **RQ4** about the robustness of each RSLCP technique across different versions.

As shown in Figs. 3–7, the APFD distributions for each RSLCP technique are similar over the five versions of each subject program. In addition, Table X presents the mean and median APFDs of RSLCP over the five versions, from which we can have the following observations:

1) In terms of the mean APFD, even in the worst case, it varies only slightly: from *v1* to *v5*, for example, varying less than 4.30% for IV1 and IV3 for all subject programs; and less than 4.00% for other RSLCP techniques (IV2, IV4, IV5, and PV).
2) With respect to the median APFD, the case is very similar to that of the mean APFD: each RSLCP technique achieves the median value varying less than 4.00% over the five versions of each program, in the worst case (except for a few cases, such as for *flex*), where the median values vary approximately 4.27%, 4.01%, and 4.01% for IV1, IV3, and IV5, respectively).

Overall, it can be observed that the APFD is quite robust over the five versions, for all RSLCP techniques, irrespective of subject programs. The answer to **RQ4**, therefore, is: Each RSLCP technique can remain robust over multiple releases of the SUT.

F. Threats to Validity

In this section, we examine some potential threats to the validity of our paper.

1) Construct Validity: In this paper, we have focused on the testing effectiveness and efficiency, measured using the rate of fault detection and the prioritization time, respectively. The APFD metric has been commonly adopted in the study of TCP [4], [6]; while the APCC metric has been widely used in the field of combinatorial interaction testing [13], [14]. Nevertheless, we acknowledge that there may be other metrics which are also pertinent to this paper, for example, *Average Percentage of Statement Coverage* [18], *Average Percentage of Decision Coverage* [18], and *Average Percentage of Method Coverage* [52].

2) Internal Validity: Any potential threat to internal validity would be mainly about the implementation of our algorithms.

We have used C++ to implement the algorithms, and have carefully tested the implementation to minimize this threat as much as possible.

3) External Validity: With respect to the external validity, the main threat is the generalization of our results. We have used only five subject programs, written in the C language, all of which are of a relatively medium size. However, we believe that the 25 versions studied here (five versions of each of the five programs) are sufficient to support the conclusions. Nevertheless, more larger size programs, or programs written in different languages, will be required to further validate our techniques.

A second potential threat to external validity relates to the input parameter models for the subject programs, which we adopted from previous studies [13], [14]. Based on these models, we also availed of the widely used sets of ATCs provided by SIR [37] in our experiments. However, different ATC sets may lead to different findings, and therefore different models (for constrained or unconstrained environments) and other sets of ATCs are still required to help generalize our findings.

4) Conclusion Validity: The main potential threat to conclusion validity relates to the randomized computation in our algorithms. To minimize this threat, all algorithms were repeated 1000 times, and inferential statistics was applied to the comparisons of results.

VI. Related Work

In this section, we introduce some related work about ATCP, which has been widely researched in several different fields, including in combinatorial testing [1], software product lines testing [2], and highly configurable systems testing [3].

According to Qu *et al.* [6], ATCP can be divided into two categories: (1) *Regeneration Prioritization Strategy* (RPS)—considering TCP during ATC generation and (2) *Pure Prioritization Strategy* (PPS)—re-ordering a given set of ATCs. Our proposed RSLCP method belongs to the second category.

A. Regeneration Prioritization Strategy

RPS has generally been applied to combinatorial testing, which attempts to generate prioritized *Covering Arrays* (CAs)—CAs are special sets of ATCs satisfying certain criteria [53]. Two CA construction strategies exist: *greedy* and *meta-heuristic search*.

1) Greedy Strategy: Bryce and Colbourn [54], [55] first proposed an RPS to generate the prioritized CAs by assigning test case weight to interaction coverage. Their strategy defines a weight for each parameter, calculates the weight for each parameter interaction (called *generation strength*) at strength 2, and then uses a greedy algorithm to construct pairwise prioritized CAs. Qu *et al.* [6], [56] proposed different weighting assignments for parameters and levels, including three weighting methods based on code coverage and one based on specification, and applied them to Bryce and Colbourn's method [54], [55], later extending this to configurable systems [57], and then also using the installation and generation cost of new configurations to improve the method [58].

2) Meta-Heuristic Search Strategy: Chen *et al.* [59] used ant colony optimization to build 2-wise prioritized CAs, using the same weighting calculations as Bryce and Colbourn [54], [55]. Similarly, Lopez-Herrejon *et al.* [60] used a genetic algorithm to generate the prioritized pairwise CAs, applying them to software product lines.

B. Pure Prioritization Strategy

Most prioritization studies have focused on interaction coverage information or test case dissimilarity to guide the prioritization of test cases. To clearly describe these strategies, we classify them into two categories of prioritization methods: *level-combination coverage-based prioritization* and *similarity-based prioritization*.

1) Level-Combination Coverage-Based Prioritization: Level-Combination Coverage-Based Prioritization (LCP) greedily chooses an element from the candidates as the next test case such that it covers the largest number of level combinations that have not already been covered by previously selected test cases. According to the prioritization strength(s) adopted, LCP can be categorized as *Fixed-strength LCP* (FLCP, i.e., λLCP), *Incremental-strength LCP* (ILCP), or *Mixed-strength LCP* (MLCP). FLCP uses a fixed prioritization strength λ throughout the entire prioritization process, whereas ILCP and MLCP can use different strength values. When choosing each element from the candidates as the next ATC in the test sequence, both FLCP and ILCP use a single prioritization strength (even though ILCP may change the prioritization strength in later steps). MLCP, on the other hand, uses more than one prioritization strength.

Regarding FLCP, Bryce and Memon [11] first proposed λLCP for prioritizing existing test suites for GUI-based programs using $\lambda = 2, 3$. Sampath *et al.* [61] applied FLCP with $\lambda = 1, 2$ to reorder user-session test cases for web applications. Bryce *et al.* [62] proposed a single model to define generic prioritization criteria (including FLCP with $\lambda = 1, 2$) that are applicable to event-driven software, such as GUI and Web applications. Qu *et al.* [6], [56] used λLCP with $\lambda = 2, 3$ to prioritize CAs by assigning a weighting for each parameter, and then obtaining the weighting of each test case. Bryce *et al.* [63] extended their λLCP (using $\lambda = 2$) technique by defining the test case length as its prioritization cost, suggesting that longer test cases would require more execution time. Wang *et al.* [27] combined test case weight and cost with Bryce and Memon's λLCP technique [11], and proposed two heuristic prioritization methods to reorder CAs by considering total and additional techniques. They also proposed a series of evaluation metrics based on test case weight and cost, and used them to assess prioritized CAs. Huang *et al.* [7] extended the strategy of adaptive random prioritization [32] to CAs, proposing a method which, by replacing Bryce and Memon's [11] λLCP technique (with $\lambda = 2, 3, 4$), attempts to reduce time costs, while maintaining testing effectiveness. Huang *et al.* [17] proposed a new λLCP technique using repeated base-choice coverage. This belongs to the same category as our proposed RSLCP method IV1. Recently, Petke *et al.* [13], [14] conducted an extensive study to investigate the testing effectiveness of λLCP with $\lambda = 2, 3, 4, 5, 6$, showing that λLCP

with small λ values could achieve comparable performance to that with high λ values.

For ILCP, Huang *et al.* [31] applied Bryce and Memon's [11] λLCP technique to the prioritization of *Variable-strength CAs* (VCAs) [64], and proposed two new VCA prioritization algorithms. The technique first uses the *main-strength* to prioritize VCAs using λLCP, and when all level combinations with the *main-strength* are covered by the selected test cases, the technique then uses the *sub-strength* to order the remaining candidate test cases. Huang *et al.* [29] used incremental interaction coverage to guide CA prioritization, by applying λLCP [11] with incremental prioritization strength values. This approach starts with strength $\lambda = 1$, and then increments the λ value when all possible λ-wise parameter-level combinations have been covered by the selected test cases.

Regarding MLCP, Huang *et al.* [33] proposed a new dissimilarity measure based on the aggregate-strength interaction coverage, and presented a new greedy PPS algorithm to prioritize CAs. This method combines the prioritization strength values $1, 2, \ldots, \tau$ (where τ is the generation strength of the given CA) to guide the selection of each test case from the candidates.

According to the classifications above, different versions of RSLCP belong to different categories. IV1 and IV2 belong the category of FLCP, and IV3 and PV belong to the category of ILCP. However, IV4 and IV5 do not fully belong the categories above, because IV4 adopts decreasing strengths, and IV5 may adopt either incremental or decreasing strengths. Nevertheless, when choosing an element from the candidates as the next ATC each time, all RSLCP techniques have the same mechanism as FLCP. Moreover, RSLCP has the same aim as ILCP—overcoming the biased selection of prioritization strength for λLCP. In order to cover all 1-wise and 2-wise level combinations as quickly as possible, PV uses the same process as ILCP. After that, PV repeats the process, ignoring the previously selected ATCs. In contrast, ILCP increments λ to 3 to prioritize the remaining ATCs, considering the already selected ATCs—it checks the λ-wise level combinations that have not been covered by the previously selected ATCs.

2) Similarity-Based Prioritization: Wu *et al.* [65] used Srikanth *et al.*'s [58] switching cost between two test cases as the similarity measure to prioritize CAs, proposing two greedy algorithms and a graph-based algorithm. Their extended work [66] proposed single-objective algorithms to reduce the switching cost, and also proposed hybrid and multiobjective algorithms to balance the tradeoff between high level-combination coverage and low switching cost. Henard *et al.* [8] introduced another similarity measure, and proposed two greedy PPSs for software product lines—*local maximum distance prioritization* and *global maximum distance prioritization*. Recently, Huang *et al.* [36] investigated 14 similarity measures for two similarity-based prioritization algorithms proposed by Henard *et al.* [8], attempting to identify the best similarity measure for each algorithm. Al-Hajjaji *et al.* [9] also proposed a new similarity measure, and applied it to the greedy prioritization algorithm for software product lines.

Huang *et al.* [5] have also recently reported on an empirical examination of 16 popular ATCP techniques, from the perspective of fault detection effectiveness.

VII. Conclusion

ATCP has been widely used in different testing situations, including combinatorial testing [1]. λLCP is perhaps the most well-known ATCP technique [11]. λLCP requires to set a fixed prioritization strength λ before testing. There is a tradeoff between testing effectiveness and efficiency for λLCP: When λ is higher, λLCP has higher testing effectiveness but lower testing efficiency; however, when λ is lower, it has lower testing effectiveness but higher testing efficiency. In this paper, we proposed a new method that attempts to balance the tradeoff between testing effectiveness and efficiency for λLCP, namely RSLCP. RSLCP has the same advantages as λLCP—it is simple, and is a static black-box technique (which means that it neither requires information about the source codes, nor is test execution necessary). However, RSLCP also has some additional advantages compared with λLCP, including that it does not face the requirement of assigning a value to λ in advance of testing. We conducted empirical studies to compare RSLCP with λLCP, ILCP, and SP, in terms of interaction coverage rate, fault detection rate, and prioritization cost. Based on these empirical studies, we have the following findings:

1) Among the six RSLCP techniques, IV1 generally has the worst testing effectiveness, while the other five techniques all have very similar performance.
2) Compared with λLCP and ILCP, RSLCP could provide a good tradeoff between testing effectiveness and efficiency.
3) Compared with SP, RSLCP not only provides better testing rates of interaction coverage and fault detection in most cases, but also requires less prioritization time.
4) All of the six RSLCP techniques are robust over multiple releases of the software-systems under test.

Because of RSLCP's potential for prioritizing ATCs, in the future we would like to develop and examine more cost-effective algorithms to achieve better tradeoff between testing effectiveness and efficiency. Multiobjective algorithms, for example, will be considered in ATCP. We will also conduct more empirical studies to further compare with other ATCP techniques, and evaluate our techniques in different applications (such as GUI and web applications), especially in larger and more complex systems.

Acknowledgment

The authors would like to thank the anonymous reviewers for their many constructive comments. The authors would also like to thank Christopher Henard for providing us the fault data for the five subject programs.

References

[1] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surveys*, vol. 43, no. 2, pp. 11:1–11:29, 2011.
[2] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing," *Inf. Softw. Technol.*, vol. 53, no. 5, pp. 407–423, 2011.
[3] X. Qu, "Testing of configurable systems," *Adv. Comput.*, vol. 89, pp. 141–162, 2013.
[4] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001.

[5] R. Huang, W. Zong, D. Towey, Y. Zhou, and J. Chen, "An empirical examination of abstract test case prioritization techniques," in *Proc. 39th Int. IEEE/ACM Conf. Softw. Eng. Companion (ICSE-C)*, Buenos Aires, Argentina, May 2017.

[6] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: A study of test case generation and prioritization," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Paris, France, Oct. 2007.

[7] R. Huang, J. Chen, Z. Li, R. Wang, and Y. Lu, "Adaptive random prioritization for interaction test suites," in *Proc. 29th Annu. ACM Symp. Appl. Comput. (SAC'14)*, 2014, pp. 1058–1063.

[8] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon, "Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 650–670, Jul. 2014.

[9] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake, "Similarity-based prioritization in software product-line testing," in *Proc. 18th Int. Softw. Product Line Conf. (SPLC'14)*, 2014, pp. 197–206.

[10] H. Wu, C. Nie, F.-C. Kuo, H. K. N. Leung, and C. J. Colbourn, "A discrete particle swarm optimization for covering array generation," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 575–591, Aug. 2015.

[11] R. C. Bryce and A. M. Memon, "Test suite prioritization by interaction coverage," in *Proc. Workshop Domain Specific Approaches Softw. Test Autom. (DoSTA'07)*, 2007, pp. 1–7.

[12] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Softw. Eng.*, vol. 19, no. 1, pp. 182–212, 2014.

[13] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing," in *Proc. 12th Joint Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Foundations Softw. Eng. (ESEC/FSE'13)*, 2013, pp. 26–36.

[14] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection," *IEEE Trans. Softw. Eng.*, vol. 41, no. 9, pp. 901–924, Sep. 2015.

[15] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Trans. Softw. Eng.*, vol. 30, no. 6, pp. 418–421, Jun. 2004.

[16] D. R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *IT Professional*, vol. 10, no. 3, pp. 19–23, May/Jun. 2008.

[17] R. Huang, W. Zong, J. Chen, D. Towey, Y. Zhou, and D. Chen, "Prioritizing interaction test suite using repeated base choice coverage," in *Proc. 40th Annu. IEEE Comput. Softw. Appl. Conf. (COMPSAC'16)*, Atlanta, GA, USA, Jun. 2016.

[18] Z. Li, M. Harman, and R. Hierons, "Search algorithms for regression test case prioritization," *IEEE Trans. Softw. Eng.*, vol. 33, no. 4, pp. 225–237, Apr. 2007.

[19] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proc. 23rd Int. Conf. Softw. Eng. (ICSE'01)*, Toronto, Ontario, Canada, May 2001.

[20] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE'13)*, 2013, pp. 192–201.

[21] Y. Lu *et al.*, "How does regression test prioritization perform in real-world software evolution?," in *Proc. 38th Int. Conf. Softw. Eng. (ICSE'16)*, 2016, pp. 535–546.

[22] L. Zhang, S.-S. Hou, C. Guo, T. Xie, and H. Mei, "Time-aware test-case prioritization using integer linear programming," in *Proc. 18th Int. Symp. Softw. Testing Anal. (ISSTA'09)*, 2009, pp. 213–224.

[23] H. Yoon and B. Choi, "A test case prioritization based on degree of risk exposure and its empirical study," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 21, no. 2, pp. 191–209, 2011.

[24] Y. Huang, K. Peng, and C. Huang, "A history-based cost-cognizant test case prioritization technique in regression testing," *J. Syst. Softw.*, vol. 85, no. 3, pp. 626–637, 2012.

[25] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry, "An information retrieval approach for regression test prioritization based on program changes," in *Proc. 37th Int. IEEE/ACM Conf. Softw. Eng. (ICSE'15)*, Florence, Italy, May 2015.

[26] A. Marchetto, M. M. Islam, W. Asghar, A. Susi, and G. Scanniello, "A multiobjective technique to prioritize test cases," *IEEE Trans. Softw. Eng.*, vol. 42, no. 10, pp. 918–940, Oct. 2016.

[27] Z. Wang, L. Chen, B. Xu, and Y. Huang, "Cost-cognizant combinatorial test case prioritization," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 21, no. 6, pp. 829–854, 2011.

[28] R. Huang, J. Chen, D. Chen, and R. Wang, "How to do tie-breaking in prioritization of interaction test suites?," in *Proc. 26th Int. Conf. Softw. Eng. Knowl. Eng. (SEKE'14)*, 2014, pp. 121–125.

[29] R. Huang, X. Xie, D. Towey, T. Y. Chen, Y. Lu, and J. Chen, "Prioritization of combinatorial test cases by incremental interaction coverage," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 23, no. 10, pp. 1427–1457, 2013.

[30] R. C. Bryce, S. Sampath, and A. M. Memon, "Developing a single model and test prioritization strategies for event-driven software," *IEEE Trans. Softw. Eng.*, vol. 37, no. 1, pp. 48–64, Jan./Feb. 2011.

[31] R. Huang, J. Chen, T. Zhang, R. Wang, and Y. Lu, "Prioritizing variable-strength covering array," in *Proc. 37th Annu. IEEE Comput. Softw. Appl. Conf. (COMPSAC'13)*, Kyoto, Japan, Jul. 2013.

[32] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE'09)*, Auckland, New Zealand, Nov. 2009.

[33] R. Huang, J. Chen, D. Towey, A. Chan, and Y. Lu, "Aggregate-strength interaction test suite prioritization," *J. Syst. Softw.*, vol. 99, pp. 36–51, 2015.

[34] B. Jiang and W. K. Chan, "Input-based adaptive randomized test case prioritization: A local beam search approach," *J. Syst. Softw.*, vol. 105, pp. 91–106, 2015.

[35] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. L. Traon, "Comparing white-box and black-box test prioritization," in *Proc. 38th Int. IEEE/ACM Conf. Softw. Eng. (ICSE'16)*, Austin, TX, USA, May 2016.

[36] R. Huang, Y. Zhou, W. Zong, D. Towey, and J. Chen, "An empirical comparison of similarity measures for abstract test case prioritization," in *Proc. 41st Annu. IEEE Comput. Softw. Appl. Conf. (COMPSAC'17)*, Turin, Italy, Jul. 2017.

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

[38] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Commun. ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[39] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep./Oct. 2011.

[40] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, Aug. 2006.

[41] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 733–752, Sep. 2006.

[42] M. Papadakis, D. Shin, S. Yoo, and D.-H. Bae, "Are mutation scores correlated with real fault detection?: A large scale empirical study on the relationship between mutants and real faults," in *Proc. 40th Int. Conf. Softw. Eng. (ICSE'18)*, 2018, pp. 537–548.

[43] M. Papadakis, C. Henard, M. Harman, Y. Jia, and Y. Le Traon, "Threats to the validity of mutation-based test assessment," in *Proc. 25th Int. Symp. Softw. Testing Anal. (ISSTA'16)*, 2016, pp. 354–365.

[44] M. Papadakis, Y. Jia, M. Harman, and Y. Le Traon, "Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique," in *Proc. 37th Int. IEEE/ACM Conf. Softw. Eng. (ICSE'15)*, Florence, Italy, May 2015.

[45] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Mutation testing advances: An analysis and survey," *Adv. Comput.*, vol. 112, pp. 275–378, 2019.

[46] Y. Jia and M. Harman, "Higher order mutation testing," *Inf. Softw. Technol.*, vol. 51, no. 10, pp. 1379–1393, 2009.

[47] M. Kintis, M. Papadakis, and N. Malevris, "Evaluating mutation testing alternatives: A collateral experiment," in *Proc. Asia-Pacific Softw. Eng. Conf. (APSEC'10)*, Sydney, NSW, Australia, Nov.-Dec. 2010.

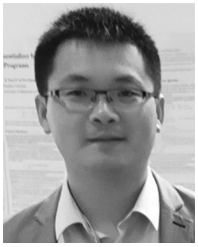
[48] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Proc. 27th Annu. NASA Goddard/IEEE Softw. Eng. Workshop (SEW-27'02)*, Greenbelt, MD, USA, USA, Dec. 2002.

[49] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Testing, Verification Rel.*, vol. 24, no. 3, pp. 219–250, 2014.

[50] M. Harman, P. McMinn, J. Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," *Empirical Softw. Eng. Verification*, pp. 1–59, 2012.

[51] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *J. Educ. Behavioral Statist.*, vol. 25, no. 2, pp. 101–132, 2000.

[52] D. Hao, L. Zhang, L. Zang, Y. Wang, X. Wu, and T. Xie, "To be optimal or not in test-case prioritization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 5, pp. 490–505, May 2016.
[53] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in *Proc. 25th Int. Conf. Softw. Eng. (ICSE'03)*, Portland, OR, USA, May 2003.
[54] R. C. Bryce and C. J. Colbourn, "Test prioritization for pairwise interaction coverage," in *Proc. 1st Int. Workshop Adv. Model-Based Testing (A-MOST'05)*, 2005, pp. 1–7.
[55] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pairwise coverage with seeding and constraints," *Inf. Softw. Technol.*, vol. 48, no. 10, pp. 960–970, 2006.
[56] X. Qu and M. B. Cohen, "A study in prioritization for higher strength combinatorial testing," in *Proc. 6th Int. IEEE Conf. Softw. Testing, Verification Validation Workshops*, Luxembourg, Luxembourg, Mar. 2013.
[57] X. Qu, M. B. Cohen, and G. Rothermel, "Configuration-aware regression testing: An empirical study of sampling and prioritization," in *Proc. Int. Symp. Softw. Testing Anal. (ISSTA'08)*, 2008, pp. 75–86.
[58] H. Srikanth, M. B. Cohen, and X. Qu, "Reducing field failures in system configurable software: Cost-based prioritization," in *Proc. 20th Int. Symp. Softw. Rel. Eng. (ISSRE'09)*, Mysuru, Karnataka, India, Nov. 2009.
[59] X. Chen, Q. Gu, X. Zhang, and D. Chen, "Building prioritized pairwise interaction test suites with ant colony optimization," in *Proc. 9th Int. Conf. Quality Softw. (QSIC'09)*, Jeju, South Korea, Aug. 2009.
[60] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba, "A parallel evolutionary algorithm for prioritized pairwise testing of software product lines," in *Proc. Int. Conf. Genetic Evol. Comput. (GECCO'14)*, 2014, pp. 1255–1262.
[61] S. Sampath, R. C. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru, "Prioritizing user-session-based test cases for web applications testing," in *Proc. 1st Int. Conf. Softw. Testing, Verification Validation (ICST'08)*, Lillehammer, Norway, Apr. 2008.
[62] R. C. Bryce, C. J. Colbourn, and D. R. Kuhn, "Finding interaction faults adaptively using distance-based strategies," in *Proc. 18th Int. Conf. Workshops Eng. Comput.-Based Syst. (ECBS'11)*, Las Vegas, NV, USA, Apr. 2011.
[63] R. C. Bryce, S. Sampath, J. B. Pedersen, and S. Manchester, "Test suite prioritization by cost-based combinatorial interaction coverage," *Int. J. Syst. Assurance Eng. Manage.*, vol. 2, no. 2, pp. 126–134, 2011.
[64] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, and J. S. Collofello, "Variable strength interaction testing of components," in *Proc. 27th Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC'03)*, Dallas, TX, USA, Nov. 2003.
[65] H. Wu, C. Nie, and F.-C. Kuo, "Test suite prioritization by switching cost," in *Proc. 7th Int. IEEE Conf. Softw. Testing, Verification Validation Workshops*, Cleveland, OH, USA, Mar./Apr. 2014.
[66] H. Wu, C. Nie, and F.-C. Kuo, "The optimal testing order in the presence of switching cost," *Inf. Softw. Technol.*, vol. 80, pp. 57–72, 2016.

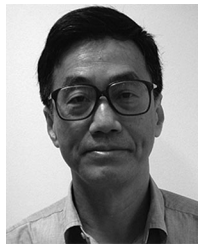


Rubing Huang (M'12) received the Ph.D. degree in computer science and technology from the Huazhong University of Science and Technology, China, in 2013.

He is an Associate Professor in the Department of Software Engineering, School of Computer Science and Communication Engineering, Jiangsu University, China. He has more than 40 publications in journals and proceedings, including in ICSE, IEEE-TR, JSS, IST, IET Software, IJSEKE, SCN, COMPSAC, SEKE, and SAC. He has served as the program committee member of SEKE14-19, SAC17-19, CTA17-19, and AI Testing 2019. His current research interests include software testing and software maintenance, especially adaptive random testing, random testing, combinatorial testing, and regression testing.

Weifeng Sun received the B.Eng. degree in computer science and technology in 2018 from Jiangsu University, Zhenjiang, China, where he is currently working toward the M.Eng. degree with the School of Computer Science and Communication Engineering.

His current research interests include software testing.



Tsong Yueh Chen (SM'03) received the B.Sc. and M.Phil. degrees from the University of Hong Kong, China, the M.Sc. and D.I.C. degrees from the Imperial College of Science and Technology, London, U.K., and the Ph.D. degree from the University of Melbourne, Australia.

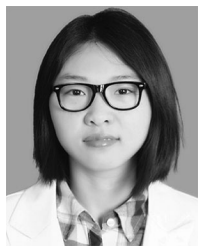
He is currently a Professor of software engineering in the Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia. He is the inventor of metamorphic testing and adaptive random testing. His main research interest is in software testing.

Dave Towey (M'03) received the B.A. and M.A. degrees in computer science, linguistics, and languages from the University of Dublin, Trinity College, Ireland, the M.Ed. degree in education leadership from the University of Bristol, U.K., and the Ph.D. degree in computer science from the University of Hong Kong, Hong Kong.

He is an Associate Professor at the University of Nottingham Ningbo China (UNNC), in Zhejiang, China, where he serves as the Director of teaching and learning, and Deputy Head of the School of Computer Science. He is also the Deputy Director of the International Doctoral Innovation Centre at UNNC. He is a member of the UNNC Artificial Intelligence and Optimization research group. He cofounded the ICSE International Workshop on Metamorphic Testing in 2016. His current research interests include software testing and technology enhanced teaching and learning.

Jinfu Chen (M'13) received the B.Eng. degree from Nanchang Hangkong University, Nanchang, China, in 2004, and the Ph.D. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2009, both in computer science and technology.

He is a Full Professor in computer science and technology in the School of Computer Science and Communication Engineering, Jiangsu University, China. His research interests include software engineering, services computing, and information security.



Weiwen Zong received the B.Eng. degree in computer science and technology in 2016 from Jiangsu University, Zhenjiang, China, where she is currently pursuing M.Eng. degree with the School of Computer Science and Communication Engineering.

Her current research interests include software testing.

Yunan Zhou received the B.Eng. degree in computer science and technology in 2016 from Jiangsu University, Zhenjiang, China, where she is currently pursuing M.Eng. degree with the School of Computer Science and Communication Engineering.

Her current research interests include software testing.