An Empirical Comparison of Similarity Measures for Abstract Test Case Prioritization

Rubing Huang*, Yunan Zhou*, Weiwen Zong*, Dave Towey†, Jinfu Chen*
*School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, P.R. China
{rbhuang, zhouyn, vevanzong, jinfuchen}@ujs.edu.cn
†School of Computer Science, The University of Nottingham Ningbo China, Ningbo 315100, P.R. China
dave.towey@nottingham.edu.cn

***Abstract*—Test case prioritization (TCP) attempts to order test cases such that those which are more important, according to some criterion or measurement, are executed earlier. TCP has been applied in many testing situations, including, for example, regression testing. An abstract test case (also called a model input) is an important type of test case, and has been widely used in practice, such as in configurable systems and software product lines. Similarity-based test case prioritization (STCP) has been proven to be cost-effective for abstract test cases (ATCs), but because there are many similarity measures which could be used to evaluate ATCs and to support STCP, we face the following question: How can we choose the similarity measure(s) for prioritizing ATCs that will deliver the most effective results? To address this, we studied fourteen measures and two popular STCP algorithms — local STCP (LSTCP), and global STCP (GSTCP). We also conducted an empirical study of five real-world programs, and investigated the efficacy of each similarity measure, according to the interaction coverage rate and fault detection rate. The results of these studies show that GSTCP outperforms LSTCP — in 61% to 84% of the cases, in terms of interaction coverage rates; and in 76% to 78% of the cases with respect to fault detection rates. Our studies also show that *Overlap*, the simplest similarity measure examined in this study, could obtain the overall best performance for LSTCP; and that *Goodall3* has the best performance for GSTCP.**

***Index Terms*—Software testing, test case prioritization, abstract test case, similarity**

I. Introduction

Because resources are usually limited, when conducting testing in practice (for example in regression testing), the test case execution order can be critical, and more important test cases in a test set should be executed as early as possible. A well-ordered test case execution sequence may be able to identify faults earlier, and thus enable earlier fault characterization, diagnosis and correction. The process of determining the order of test cases in a test set is called *test case prioritization* (TCP) [1], and has been used extensively in many testing situations. For example, two possible scenarios where TCP should be applied are: (1) regression testing; and (2) after a test set is constructed, if only some of the test cases can be executed (due, perhaps, to limited testing resources).

Many TCP algorithms have been proposed to guide the prioritization of different types of test case, such as random test case prioritization, and code-coverage-based prioritization [1]. An *abstract test case* (ATC) [2] — also called a *model input* [3] — is an important type of test case [4], and can be generated based on a model of the object under test. ATCs have been widely used in testing, including for testing highly-configurable systems [5, 6], the category-partition testing method [4], and combinatorial testing [7]. ATC prioritization has also been studied extensively in recent years, especially in the field of combinatorial testing [8–10] and software product line testing [11, 12]. Similarity-based test case prioritization (STCP) is a cost-effective prioritization technique, achieving comparable testing effectiveness to other techniques, but with much greater testing efficiency [11, 12]. Because many different similarity measures exist to evaluate ATCs [13], and guide the STCP process, we are faced with the following question when using the STCP: Is there a *best* similarity measure?

This paper investigated fourteen similarity measures for ATCs, and two widely-used STCP algorithms — *local similarity-based test case prioritization* (LSTCP), and *global similarity-based test case prioritization* (GSTCP) [12]. We also conducted an empirical study involving five real-world programs, each of which has six versions by mutation analysis. The study investigated the testing effectiveness of each measure, according to two quality metrics: the rate of interaction coverage, and the fault detection rate. The results show that the similarity measures performed differently for different STCP algorithms, with better performance for GSTCP than for LSTCP — in 61% to 84% of the cases, in terms of interaction coverage rates; and in 76% to 78% of the cases for fault detection rates. The results also show that *Overlap*, the simplest similarity measure examined, could obtain the overall best performance for LSTCP; and that *Goodall3* was best for GSTCP.

The rest of this paper is organized as follows: Section II explains some background information for ATCs and TCP. Section III explains the methodology and the experimental settings, including the design of the research questions. Section IV describes the results of the experiments, answers the research questions, and examines potential threats to validity. Finally, Section V concludes the paper.

II. Background

In this section, we explain some background information, including abstract test cases (ATCs) and test case prioritization (TCP).

0730-3157/17 $31.00 © 2017 IEEE
DOI 10.1109/COMPSAC.2017.271

A. Abstract Test Cases

A test object is generally influenced by a number of *factors* or *parameters* (such as configurations, features, components, etc.). Typically, each parameter can take a fixed number of possible *values*, or *levels*. Generally, there may also be constraints among parametric values, with some value combinations being infeasible. Based on this, we can define the *input parameter model* [2] used to model the test object as follows:

Definition 1: An *input parameter model*, *Model*$(\{p_1, p_2, \cdots, p_k\}, \{V_1, V_2, \cdots, V_k\}, \mathcal{C})$, represents the information about the test object — $p_1, p_2, \cdots, p_k$, are the k parameters; each V_i is the set of possible values for the i-th parameter (p_i); and $\mathcal{C}$ is the set of value combination constraints.

For instance, consider the following input parameter model with value constraints: *Model*$(\{p_1, p_2, p_3\}, \{\{0, 1\}, \{2, 3, 4\}, \{5, 6, 7\}\}, \mathcal{C} = \{((p_1 = 0) \Rightarrow (p_2 \neq 4)), ((p_1 = 1) \Rightarrow (p_3 = 5))\})$. This model has two constraints, and three parameters, of which the first has two values, and the last two both have three. Because specific values of each parameter have no impact on the model, without loss of generality, we can use the following abbreviated version: *Model*$(|V_1||V_2|\cdots|V_k|, \mathcal{C})$. The example above can therefore be represented as: *Model*$(2^1 3^2, \mathcal{C} = \{(-0-4), (-1+5)\})$.

An input parameter model (if available) can be used to construct *abstract test cases* (ATCs) [2] (also called *model inputs* [3]) for testing the test object. A definition of an ATC is as follows:

Definition 2: An *abstract test case*, denoted $(v_1, v_2, \cdots, v_k)$, is a k-tuple, where $v_i \in V_i$ $(i = 1, 2, \cdots, k)$.

If all the value constraints in $\mathcal{C}$ are satisfied, then an ATC is said to be *valid*, otherwise it is *invalid*. An example of a valid ATC for the previous model is $(0, 2, 5)$; and an example of an invalid one is $(0, 4, 5)$, which violates the constraint $((p_1 = 0) \Rightarrow (p_2 \neq 4))$. In addition, each ATC with size d could cover some tuples with the size x less than d $(1 \leq x < d)$, where the tuple is generally called *x-wise value combination* [14] or *x-wise schema* [7]. For example, an ATC $(1, 2, 5)$ covers three 2-wise value combinations: $(1, 2)$, $(1, 5)$, and $(2, 5)$.

B. Test Case Prioritization

Test case prioritization (TCP) schedules test cases so that those with higher priority, according to some criteria, are executed earlier than those with lower priority. When the execution of all test cases in a test suite is not possible, a well-designed execution order can be very important. The TCP problem can be defined as follows [1]:

Definition 3: 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 test case prioritization problem is to find a prioritized test set $S \in \Omega$ such that:

$$(\forall S')\ (S' \in \Omega)\ (S' \neq S)\ [g(S) \geq g(S')] \quad (1)$$

III. Methodology

A. Research Questions

The objective of this study was to answer the following three research questions:

- **RQ1**. How effective are similarity measures used in *local similarity-based TCP* (LSTCP), in terms of rates of interaction coverage and fault detection? (This research question aims to identify which similarity measure is the most effective for LSTCP.)
- **RQ2**. How effective are similarity measures used in *global similarity-based TCP* (GSTCP), in terms of rates of interaction coverage and fault detection? (This research question aims to identify which similarity measure is the most effective for GSTCP.)
- **RQ3**. How well do the LSTCP techniques compare with GSTCP, according to the interaction coverage and fault detection rates? (The answer to this should enable testers to select between LSTCP and GSTCP.)

TABLE I
Studied Programs

Subject	Input Parameter Model	Test Pool	Information	V0	V1	V2	V3	V4	V5
flex	*Model*$(2^6 3^2 5^1, \mathcal{C}_1)$, $\lvert\mathcal{C}_1\rvert = 32$	500	*Version*	2.4.3 *(1993)*	2.4.7 *(1994)*	2.5.1 *(1995)*	2.5.2 *(1996)*	2.5.3 *(1996)*	2.5.4 *(1997)*
			LOC	8,959	9,470	12,231	12,249	12,370	12,366
			#Faults	-	32	32	20	33	32
grep	*Model*$(2^1 3^3 4^2 5^1 6^1 8^1, \mathcal{C}_2)$, $\lvert\mathcal{C}_2\rvert = 58$	440	*Version*	2.0 *(1996)*	2.2 *(1998)*	2.3 *(1999)*	2.4 *(1999)*	2.5 *(2002)*	2.7 *(2010)*
			LOC	8,163	11,988	12,724	12,826	20,838	58,344
			#Faults	-	56	58	54	58	59
sed	*Model*$(2^7 3^1 4^1 6^1 10^1, \mathcal{C}_3)$, $\lvert\mathcal{C}_3\rvert = 58$	324	*Version*	3.0.1 *(1998)*	3.0.2 *(1998)*	4.0.6 *(2003)*	4.0.8 *(2003)*	4.1.1 *(2004)*	4.2 *(2009)*
			LOC	7,790	7,793	18,545	18,687	21,743	26,466
			#Faults	-	16	18	18	19	22
make	*Model*$(2^{10}, \mathcal{C}_4)$, $\lvert\mathcal{C}_4\rvert = 28$	111	*Version*	3.75 *(1996)*	3.76.1 *(1997)*	3.77 *(1998)*	3.78.1 *(1999)*	3.79 *(2000)*	3.80 *(2002)*
			LOC	17,463	18,568	19,663	20,461	23,125	23,400
			#Faults	-	37	29	28	29	28
gzip	*Model*$(2^{13} 3^1, \mathcal{C}_5)$, $\lvert\mathcal{C}_5\rvert = 69$	156	*Version*	1.0.7 *(1993)*	1.1.2 *(1993)*	1.2.2 *(1993)*	1.2.3 *(1993)*	1.2.4 *(1993)*	1.3 *(1999)*
			LOC	4,324	4,521	5,048	5,059	5,178	5,682
			#Faults	-	8	8	7	7	7

B. Subject Programs

We used five open-source programs (obtained from the GNU FTP server [15]) written in the C language: flex, grep, sed, gzip, and make. The flex program is a lexical analysis generator; grep and sed are widely-used command-line tools for searching and processing text-matching regular expressions; make is a popular utility used to control the compile and build processes of programs; and gzip is a compression utility. These programs have been widely used in the field of test case prioritization [1, 3, 10, 16].

Table I presents details of the subject programs, including version number, year of release, its LOC in uncommented lines of code (measured by `cloc` [17]), and number of mutation faults. The table also includes the input parameter model for each program (from previous work by Petke et al. [10]), and the number of ATCs. These test suites are available from the Software Infrastructure Repository (SIR) [18].

C. Fault Seeding and Detection

For each subject program, the original version does not contain any faults. Although a number of hand-seeded faults are available from the SIR [18], they are easily detected (on average more than 60% of test cases can identify them). In this paper, therefore, we used mutation faults [19] (see Table I). As discussed in previous studies [20, 21], mutation faults are considered appropriate for studying test case prioritization.

We used the same mutation faults as in [3]. A fault is said to be detected by a test case if the execution results are different for the original and seeded versions.

D. The Two Prioritization Algorithms Studied

In this section, we present the two algorithms for similarity-based TCP (STCP) for ATCs: *local STCP* (LSTCP), and *global STCP* (GSTCP). The algorithms were originally proposed by Henard et al. [12], and were modified for this study.

Algorithm 1 formally describes the LSTCP algorithm for prioritizing ATCs. LSTCP iterates over the candidate set of ATCs T, identifying the two ATCs sharing minimum similarity

Algorithm 1: Local STCP algorithm

Input: $T = \{t_1, t_2, \cdots, t_n\}$ ▷ Unordered abstract test cases
Output: S ▷ Prioritized abstract test cases
1: $S \leftarrow []$
2: $T \leftarrow T \setminus \{t_i, t_j\}$
3: **while** $\#T > 0$ **do**
4: **if** $\#T > 1$ **then**
5: Select $t_i, t_j \in T$ where $min(sim(t_i, t_j))$ ▷ take the random one in caseof equality
6: **if** *rand*$(0, 1.0) > 0.5$
7: $S.add(t_i)$
8: $S.add(t_j)$
9: **else**
10: $S.add(t_j)$
11: $S.add(t_i)$
12: **end if**
13: **else** ▷ T contains only one element
14: $S.add(tc_i)$, where $tc_i \in T$
15: $T \leftarrow \emptyset$
16: **end if**
17: **end while**
18: **return** S

Algorithm 2: Global STCP algorithm

Input: $T = \{t_1, t_2, \cdots, t_n\}$ ▷ Unordered abstract test cases
Output: S ▷ Prioritized abstract test cases
1: $S \leftarrow []$
2: Select $t_i, t_j \in T$ where $min(sim(t_i, t_j))$ ▷ take the random one in case of equality
3: **if** *rand*$(0, 1.0) > 0.5$
4: $S.add(t_i)$
5: $S.add(t_j)$
6: **else**
7: $S.add(t_j)$
8: $S.add(t_i)$
9: **end if**
10: $T \leftarrow T \setminus \{t_i, t_j\}$
11: **while** $\#T > 0$ **do**
12: **if** $\#T > 1$ **then**
13: $s \leftarrow size(S)$
14: Select $t_i \in T$, where $min(\sum_{i=1}^{s} sim(t_i, S[j]))$ ▷ take the random one in case of equality
15: $S.add(t_i)$
16: $T \leftarrow T \setminus \{t_i\}$
17: **else** ▷ T contains only one element
18: $S.add(tc_i)$, where $tc_i \in T$
19: $T \leftarrow \emptyset$
20: **end if**
21: **end while**
22: **return** S

(line 4). These are then added to the list S, and removed from T (lines 6 to 12). The process is repeated until all ATCs from T have been moved to S. Compared to the original LSTCP [12], there are two major differences in our algorithm: (1) when facing a tie-breaking situation (with more than one pair of ATCs having the same minimum similarity), the original LSTCP adopts a *first-test-case tie-breaking technique* (i.e., selects the first pair) [22], however our algorithm uses the *random tie-breaking technique* (i.e., selects a random pair); and (2) when selecting the best pair of ATCs from candidates, the original LSTCP adds these two ATCs to S successively, however our algorithm adds them in a random order. Our LSTCP algorithm, therefore, has a more normalized aspect than the original.

The Global STCP (GSTCP) algorithm for ATC prioritization is shown in Algorithm 2. Initially, GSTCP is similar to LSTCP — choosing a pair of ATCs from candidates in T with minimum similarity (line 2), then adding them to S (lines 3 to 9) and removing them from T (line 10). This moving of elements from T to S is repeated such that each moved element achieves minimum similarity to *all* ATCs already in S (lines 11 to 21). More specifically, for each candidate in T, the sum of individual similarity values with elements of S is calculated, and the candidate with the minimum lowest similarity is selected as the next element to be moved to S (line 14). This process is continued until all elements have been moved from T to S. Because GSTCP shares the same differences as LSTCP compared to the original LSTCP [12], it is also more normalized than the original.

Because of the randomization involved in some of the prioritization strategies — due to the random tie-breaking technique or random addition sequencing — we ran each experiment 200 times, recording and analyzing the results for each technique.

E. The 14 Similarity Measures

Various similarity measures can be used to evaluate ATCs, and in this paper we used a number of popular ones [13, 23], which will be presented in this section. For ease of presentation, we first define some terms.

Considering a set of ATCs $T = \{t_1, t_2, \cdots, t_n\}$ containing n ATCs derived from a *Model*$(|V_1||V_2|\ ...|V_k|, \mathcal{C})$ for parameters $P_1, P_2, \cdots, P_k$, we use the following notations:

- $f_i(x)$: The number of times parameter P_i takes the value x in T, where $i = 1, 2, \cdots, k$. Note that if $x \notin V_i$, $f_i(x) = 0$.
- $\hat{p}_i(x)$: The sample probability of parameter P_i taking the value x in T, where $i = 1, 2, \cdots, k$. The sample probability is given by:

$$\hat{p}_i(x) = \frac{f_i(x)}{n} \tag{2}$$

- $p_i^2(x)$: Another probability estimate of parameter P_i taking the value x in T is given by:

$$p_i^2(x) = \frac{f_i(x)(f_i(x)-1)}{n(n-1)} \tag{3}$$

  where $i = 1, 2, \cdots, k$.
- *sim*(X, Y): A generalized similarity measure between two ATCs, $X = (X_1, X_2, \cdots, X_k)$ and $Y = (Y_1, Y_2, \cdots, Y_k)$, where $X, Y \in T$ and $X_i, Y_i \in V_i$ $(i =$

TABLE II
SIMILARITY MEASURES FOR ABSTRACT TEST CASES.

No.	Similarity Name	$sim_i(X_i, Y_i)$	$\omega_i, i = 1, 2, \cdots, k$	Mnemonic
1.	*Overlap*	$= \begin{cases} 1 & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	OVE
2.	*Eskin*	$= \begin{cases} 1 & \text{if } X_i = Y_i \\ \frac{\lvert V_i\rvert^2}{\lvert V_i\rvert^2+2} & \text{otherwise} \end{cases}$	$\frac{1}{k}$	ESK
3.	*IOF*	$= \begin{cases} 1 & \text{if } X_i = Y_i \\ \frac{1}{1+\log f_i(X_i)\times\log f_i(Y_i)} & \text{otherwise} \end{cases}$	$\frac{1}{k}$	IOF
4.	*OF*	$= \begin{cases} 1 & \text{if } X_i = Y_i \\ \frac{1}{1+\log \frac{N}{f_i(X_i)}\times\log \frac{N}{f_i(Y_i)}} & \text{otherwise} \end{cases}$	$\frac{1}{k}$	OF
5.	*Lin*	$= \begin{cases} 2\log \hat{p}_i(X_i) & \text{if } X_i = Y_i \\ 2\log(\hat{p}_i(X_i)+\hat{p}_i(Y_i)) & \text{otherwise} \end{cases}$	$\frac{1}{\sum_{i=1}^{k}\log \hat{p}_i(X_i)+\log \hat{p}_i(Y_i)}$	LIN
6.	*Lin1*	$= \begin{cases} \sum\limits_{q\in Q}\log \hat{p}_k(q) & \text{if } X_i = Y_i \\ 2\log \sum\limits_{q\in Q}\hat{p}_k(q) & \text{otherwise} \end{cases}$	$\frac{1}{\sum_{i=1}^{k}\sum_{q\in Q}\log \hat{p}_i(X_i)}$	LIN1
7.	*Goodall1*	$= \begin{cases} 1-\sum\limits_{q\in Q} p_i^2(q) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	GO1
8.	*Goodall2*	$= \begin{cases} 1-\sum\limits_{q\in Q} p_i^2(q) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	GO2
9.	*Goodall3*	$= \begin{cases} 1-\sum p_i^2(X_i) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	GO3
10.	*Goodall4*	$= \begin{cases} \sum p_i^2(X_i) & \text{if } X_i = Y_i \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{k}$	GO4
11.	*Burnaby*	$= \begin{cases} 1, & \text{if } X_i = Y_i \\ \frac{\sum_{q\in V_i} 2\log(1-\hat{p}_i(q))}{\log \frac{\hat{p}_i(X_i)\hat{p}_i(Y_i)}{(1-\hat{p}_i(X_i))(1-\hat{p}_i(Y_i))}+\sum_{q\in V_i} 2\log(1-\hat{p}_i(q))}, & \text{otherwise} \end{cases}$	$\frac{1}{k}$	BUR
12.	*Smirnov*	$= \begin{cases} 2+\frac{n-f_k(t_{ai})}{f_k(X_i)}+\mathcal{A}, & \text{if } X_i = Y_i \\ \sum_{q\in\{V_i\setminus X_i\}}\frac{f_k(q)}{n-f_k(q)}, & \text{otherwise} \end{cases}$	$\frac{1}{\sum_{i=1}^{k}\lvert V_i\rvert}$	SMI
13.	*Gambaryan*	$= \begin{cases} -[\hat{p}_i(t_{ai})\log_2(\hat{p}_i(t_{ai}))+ & \text{if } t_{ai} = t_{bi} \\ (1-\hat{p}_i(t_{ai}))\log_2(1-\hat{p}_i(t_{ai}))] & \\ 0 & \text{otherwise} \end{cases}$	$\frac{1}{\sum_{i=1}^{k}\lvert V_i\rvert}$	GAM
14.	*Anderberg*	$\frac{\sum_{i\in\{1\le i\le k: X_i=Y_i\}}(\frac{1}{\hat{p}_i(X_i)})^2\frac{2}{\lvert V_i\rvert(\lvert V_i\rvert+1)}}{\sum_{i\in\{1\le i\le k: X_i=Y_i\}}(\frac{1}{\hat{p}_i(X_i)})^2\frac{2}{\lvert V_i\rvert(\lvert V_i\rvert+1)}+\sum_{i\in\{1\le i\le k: X_i\neq Y_i\}}(\frac{1}{2\hat{p}_i(X_i)\hat{p}_i(Y_i)})\frac{2}{\lvert V_i\rvert(\lvert V_i\rvert+1)}}$		AND

Note: For *Lin1*, $Q = \{q | q \in V_i, \min(\hat{p}_i(X_i), \hat{p}_i(Y_i)) \le \hat{p}_i(q) \le \max(\hat{p}_i(X_i), \hat{p}_i(Y_i))\}$.
For *Goodall1*, $Q = \{q | q \in V_i, p_i(q) \ge p_i(X_i)\}$.
For *Goodall2*, $Q = \{q | q \in V_i, p_i(q) \le p_i(X_i)\}$.

$1, 2, \cdots, k$). The definition of $\text{sim}(X, Y)$ is given by:

$$sim(X, Y) = \sum_{i=1}^{k} \omega_i * sim_i(X_i, Y_i) \quad (4)$$

where $sim_i(X_i, Y_i)$ (for $i = 1, 2, \cdots, k$) is the similarity between two values X_i and Y_i of parameter P_i; and ω_i denotes the weight assigned to parameter P_i.

We focus on the definitions of $sim_i(X_i, Y_i)$, and ω_i for each similarity measurement. Table II summarises details of fourteen previously investigated similarity measures [13, 23], including the similarity name, its mnemonic for use with the SCTP algorithms, per-parameter similarity, weighting assignment, and a reference to its original research publication.

F. Metrics

We evaluated the different prioritization techniques from the following two aspects: (a) interaction coverage rate (how quickly each prioritized set of ATCs achieved an interaction coverage); and (b) fault detection rate (how well each prioritized set of ATCs identified faults).

1) Interaction Coverage Rate: The *average percentage of λ-wise combinations covered* (APCC) [10, 24] was used to measure the rate of interaction coverage of strength λ. If $S = \langle tc_1, tc_2, \cdots, tc_N \rangle$ is a prioritized set of ATCs obtained by prioritizing T with size N, then the definition of APCC for S at strength λ ($1 \leq \lambda \leq k$) is:

$$APCC(\lambda, S) = \frac{\sum_{i=1}^{N-1} |\bigcup_{j=1}^{i} CombSet(\lambda, tc_j)|}{N \times |\bigcup_{j=1}^{N} CombSet(\lambda, tc_j)|} \quad (5)$$

where $CombSet(\lambda, tc_j)$ returns the set of λ-wise value combinations covered by tc_j.

Generally, the APCC metric values range from 0.0 to 1.0, with higher values indicating better rates of interaction coverage at a specific strength. Following previous studies [10], in this study, we considered APCC with $\lambda = 1, 2, 3, 4, 5, 6$.

2) Fault Detection Rate: The *average percentage of faults detected* (APFD) was used to assess the fault detection [1], but has the requirement of needing the fault-detection capability details for all executed test cases.

If T is a test set containing N ATCs, and F is a set of m faults revealed by T, then SF_i is the number of ATCs in the prioritized set of T (S) that are executed before detecting fault $F_i \in F$. The APFD for S is calculated as follows [1]:

$$APFD(S) = 1 - \frac{SF_1 + SF_2 + \cdots + SF_m}{N \times m} + \frac{1}{2N} \quad (6)$$

IV. Results

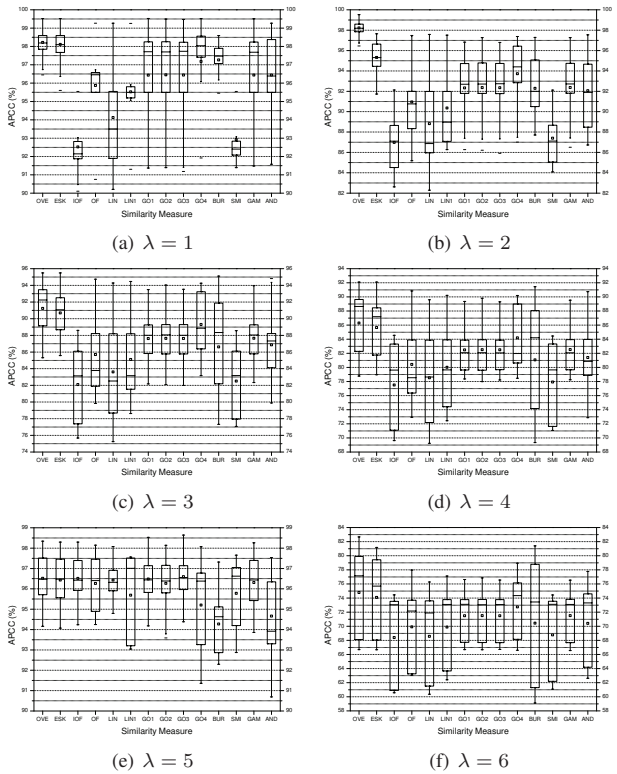
In this section, we present the experimental results. In the plots in each figure, the X-axis lists the various techniques compared, and the Y-axis shows the resulting APCC or APFD values. Each box plot shows the mean (square in the box), median (line in the box), upper and lower quartiles, and min and max APCC or APFD values for the technique.

A. Interaction Coverage Experiments

Figure 1 shows the results of the interaction coverage rates (measured with APCC for $\lambda = 1, 2, 3, 4, 5, 6$) for LSTCP; and Figure 2 presents the results for GSTCP. Figure 3(a) shows the average APCC values for all six λ values for each similarity measure for LSTCP; and Figure 3(b) shows the results for GSTCP. In these figures, each plot shows the distribution of the 1,000 APCC values (200 orderings for each of the five programs) at a given strength value λ ($1 \leq \lambda \leq 6$).

Table III presents the statistical APCC analysis of all pairwise comparisons for the five programs, based on Figure 3, with each cell showing the p-value/$\hat{A}_{12}$ measures. For example, the coordinate (ESK, OVE) shows the p-value/$\hat{A}_{12}$ (of 7.4E-11/0.42) for *Eskin* compared with *Overlap*, which means that: (i) the difference between ESK and OVE is significant (because the p-value is much less than 0.01); and (ii) ESK has a 42% probability of outperforming OVE (i.e., OVE is better than ESK with 58% probability). In this table, the lower triangle (white region) shows results comparing similarity measures in LSTCP; and the upper triangle (gray region) shows the results for GSTCP.

*1) **RQ1: LSTCP Similarity Measures**:* When λ is equal to 1, 2, 3, 4, or 6, OVE performs best, followed by ESK; and IOF and SMI generally perform worst. However, when $\lambda = 5$ (Figure 1(e)), GO3, GO1, OVE, IOF, ESK, and LIN have the best mean APCC values, approximately 96.5%, but LIN1 has



(a) $\lambda = 1$ (b) $\lambda = 2$ (c) $\lambda = 3$ (d) $\lambda = 4$ (e) $\lambda = 5$ (f) $\lambda = 6$

Fig. 1. APCC for each similarity measure in LSTCP.

TABLE III
STATISTICAL APCC ANALYSIS OF ALL PAIRWISE COMPARISONS FOR FIVE PROGRAMS: LSTCP (LOWER TRIANGLE) AND GSTCP (UPPER TRIANGLE)

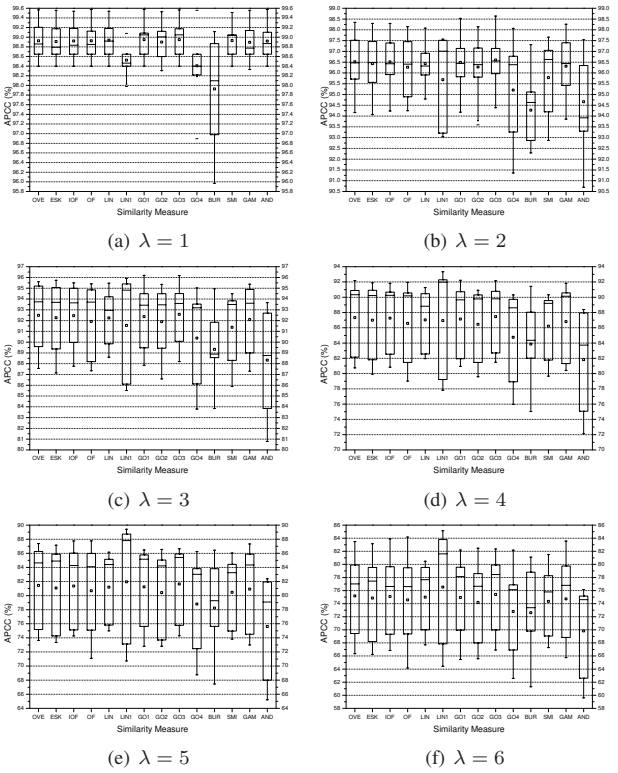
	OVE	ESK	IOF	OF	LIN	LIN1	GO1	GO2	GO3	GO4	BUR	SMI	GAM	AND
OVE	–	2.1E-11/0.59	0.06/0.52	2.6E-12/0.59	4.7E-12/0.55	9.5E-15/0.40	3.6E-04/0.55	0.0*/0.73	1.3E-05/0.44	0.0*/0.75	0.0*/0.68	0.0*/0.74	0.0*/0.64	0.0*/0.76
ESK	7.4E-11/0.42	–	7.5E-04/0.46	0.01/0.53	0.01/0.53	9.5E-15/0.40	0.51/0.49	0.0*/0.70	2.0E-09/0.42	0.0*/0.75	0.0*/0.67	0.0*/0.68	2.6E-09/0.58	0.0*/0.76
IOF	0.0*/0.12	0.0*/0.12	–	1.7E-07/0.57	0.02/0.53	9.5E-15/0.40	0.11/0.52	0.0*/0.69	6.6E-10/0.42	0.0*/0.73	0.0*/0.69	0.0*/0.73	7.1E-14/0.60	0.0*/0.76
OF	0.0*/0.23	0.0*/0.24	0.0*/0.62	–	0.48/0.49	0.0*/0.37	2.1E-04/0.45	7.2E-10/0.58	0.0*/0.39	0.0*/0.71	0.0*/0.67	0.0*/0.64	0.81/0.50	0.0*/0.76
LIN	0.0*/0.17	0.0*/0.22	1.7E-03/0.54	0.0*/0.40	–	9.2E-15/0.40	0.16/0.48	1.1E-14/0.60	2.5E-15/0.40	0.0*/0.67	0.0*/0.69	0.0*/0.68	0.33/0.51	0.0*/0.76
LIN1	0.0*/0.20	0.0*/0.24	0.0*/0.64	1.6E-03/0.46	0.0*/0.62	–	9.5E-15/0.60	9.5E-15/0.60	9.5E-15/0.60	0.0*/0.72	0.0*/0.68	9.5E-15/0.60	9.5E-15/0.60	0.0*/0.76
GO1	0.0*/0.22	0.0*/0.26	0.0*/0.76	0.0*/0.66	0.0*/0.67	0.0*/0.64	–	0.0*/0.65	5.3E-10/0.42	0.0*/0.72	0.0*/0.68	0.0*/0.69	3.1E-07/0.57	0.0*/0.76
GO2	0.0*/0.22	0.0*/0.27	0.0*/0.76	0.0*/0.66	0.0*/0.67	0.0*/0.64	0.89/0.50	–	0.0*/0.29	0.0*/0.68	0.0*/0.65	3.4E-10/0.58	0.0*/0.37	0.0*/0.76
GO3	0.0*/0.22	0.0*/0.26	0.0*/0.76	0.0*/0.66	0.0*/0.66	0.0*/0.64	0.99/0.50	0.89/0.50	–	0.0*/0.74	0.0*/0.72	0.0*/0.74	0.0*/0.62	0.0*/0.76
GO4	0.0*/0.29	0.0*/0.39	0.0*/0.79	0.0*/0.68	0.0*/0.74	0.0*/0.70	0.0*/0.63	0.0*/0.62	0.0*/0.63	–	1.4E-15/0.60	2.1E-10/0.42	0.0*/0.27	0.0*/0.73
BUR	0.0*/0.30	0.0*/0.36	0.0*/0.75	0.07/0.52	0.0*/0.64	5.1E-10/0.58	2.8E-03/0.54	7.0E-03/0.53	2.5E-03/0.54	1.7E-07/0.43	–	0.0*/0.38	0.0*/0.33	3.6E-04/0.55
SMI	0.0*/0.12	0.0*/0.13	2.8E-10/0.58	0.0*/0.38	3.7E-03/0.46	0.0*/0.36	0.0*/0.25	0.0*/0.25	0.0*/0.24	0.0*/0.21	4.3E-12/0.25	–	1.2E-03/0.36	0.0*/0.76
GAM	0.0*/0.22	0.0*/0.27	0.0*/0.76	0.0*/0.66	0.0*/0.67	0.0*/0.64	0.82/0.50	0.94/0.50	0.82/0.50	0.0*/0.38	6.0E-03/0.46	0.0*/0.76	–	0.0*/0.76
AND	0.0*/0.24	0.0*/0.26	0.0*/0.70	5.4E-05/0.55	0.0*/0.66	0.0*/0.61	9.3E-05/0.45	1.5E-04/0.45	1.4E-04/0.45	0.0*/0.36	0.03/0.47	0.0*/0.70	9.7E-05/0.45	–

*p-value less than 2.2E-16.

the best median APCC (97%).

Based on the average APCC over all six λ values (Figure 3(a)), OVE and ESK are the two best prioritization techniques for LSTCP, according to the mean, median, minimum, and maximum values. In addition, according to the mean APCC, GO4 is the next best; but according to the median value, BUR has the next best performance.

As can be seen in Table III, all the p-values for the LSTCP statistical analyses are highly significant for comparisons between OVE and each of the other 13 prioritization techniques. Similarly, the effect size $\hat{A}_{12}$ is lower than 0.30 compared with any other LSTCP technique (other than ESK, whose $\hat{A}_{12}$ is equal to 0.42 against OVE), which means that OVE performs better in more than 70% of the cases (but 58% for ESK).

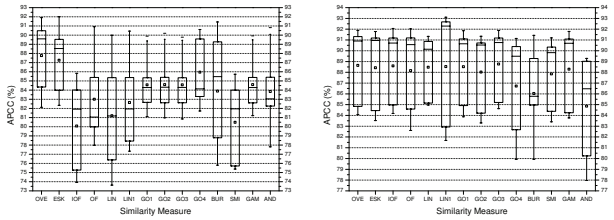


(a) $\lambda = 1$ (b) $\lambda = 2$

(c) $\lambda = 3$ (d) $\lambda = 4$

(e) $\lambda = 5$ (f) $\lambda = 6$

Fig. 2. APCC for each similarity measure in GSTCP.



(a) LSTCP (b) GSTCP

Fig. 3. Average APCC upon $\lambda = 1, 2, 3, 4, 5, 6$ for each similarity measure.

Additionally, when comparing any LSTCP techniques (other than OVE) against ESK, all the p-values are highly significant, and the effect size $\hat{A}_{12}$ is lower than 0.40. The next best techniques are GO4 and BUR, which have high significance against other techniques, and have a corresponding effect size $\hat{A}_{12}$ lower than 0.50. Overall, the statistical analyses confirm the box plots observations that, among the LSTCP techniques, OVE has the highest interaction coverage rates, followed by ESK, GO4, and BUR.

*2) **RQ2: GSTCP Similarity Measures**:* When $\lambda = 1$ (Figure 2(a)), GO1 and GO3 have the best performance, followed by SMI and GO2 — all of which have median APCC values greater than 99.0%, and mean values greater than 99.85%. The three worst techniques are GO4, LIN1, and BUR. When $\lambda \neq 1$ (i.e., $\lambda = 2$, 3, 4, 5, or 6), LIN1 performs best according to the median, for all λ cases; and BUR and AND perform worst. Regarding the average APCC (Figure 3(b)), LIN1 is the best with a median APCC greater than 92.0%; followed by OVE and ESK, with median values approaching approximately 91.0%; and BUR and AND have the worst performance. According to the mean APCC, however, OVE, ESK, IOF, LIN, LIN1, GO1, and GO3 perform best, and are similar to each other (equal to about 88.5%).

The upper triangular area in Table III gives the results of statistical analyses for GSTCP, from which we have the following observations: LIN1 is best for GSTCP, because it achieves high significance against all other prioritization techniques (the p-value is much less than 0.01), and it also has higher effect size $\hat{A}_{12}$ values than others (all $\hat{A}_{12}$ values are greater than or equal to 0.60, which indicates that LIN1 should outperform others in more than 60% of all cases). The

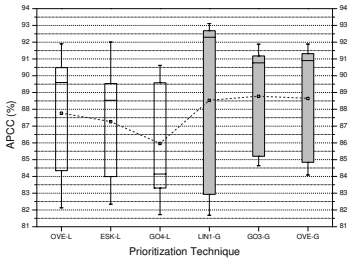


Fig. 4. Average APCC for the three best LSTCP and GSTCP techniques.

second best is GO3, followed by OVE and IOF. Additionally, AND performs worst for GSTCP, followed by BUR. Overall, the statistical analyses support the box plots observations.

As can be seen from the results for both **RQ1** and **RQ2**, the same similarity measure has different performances with different STCP algorithms: For example, BUR is a possible choice for LSTCP but not for GSTCP — which is reasonable, given that different similarity measures have different properties in LSTCP compared with in GSTCP.

*3) **RQ3: Comparing LSTCP and GSTCP**:* To investigate the differences between LSTCP and GSTCP, we compared their three best techniques — OVE, ESK, and GO4 for LSTCP; and LIN1, GO3, and OVE for GSTCP. For ease of description, we abbreviate these techniques as OVE-L, ESK-L, GO4-L, LIN1-G, GO3-G, and OVE-G, respectively.

Figure 4 shows the distribution of the average APCC values for these techniques for all programs. The joined line through the squared points gives the mean APCC values, according to which the GSTCP techniques outperform the LSTCP ones, with the greatest difference between GO3-G and GO4-L (approximately 2.5%), and the smallest difference between LIN1-G and OVE-L (less than 1.0%). The median APCC values (the bars inside each box) also show the GSTCP techniques outperforming the LSTCP techniques, with the greatest difference between LIN1-G and GO4-L (more than 8.0%), and the smallest between GO3-G and OVE-L (about 1.0%).

Table IV presents the statistical analyses for the comparisons among the six techniques. In terms of the p-values, the comparisons are highly significant (much lower than 0.01). From the perspective of the effect size, the GSTCP techniques perform better than LSTCP, with results ranging from 61% ($100\% - 39\%$) to 84% ($100\% - 16\%$).

In summary, the differences between the average APCC values for LSTCP and GSTCP range from 1.0% to 2.5%, with GSTCP performing better overall than LSTCP 61% to 84% of the time. Therefore, when selecting a single technique out of the 28 studied (14 similarity measures for two algorithms), then, according to the APCC metric, the similarity measure LIN1 for GSTCP is the best choice.

TABLE IV
STATISTICAL APCC COMPARISON BETWEEN 3 BEST LSTCP AND GSTCP TECHNIQUES

		GSTCP		
		LIN1-G	**GO3-G**	**OVE-G**
LSTCP	**OVE-L**	0.0*/0.39	0.0*/0.31	0.0*/0.31
	ESK-L	0.0*/0.39	0.0*/0.30	0.0*/0.29
	GO4-L	0.0*/0.37	0.0*/0.16	0.0*/0.16

*p-value lower than 2.2E-16.

B. Fault Detection Experiments

Figure 5 shows the fault detection results (measured with APFD over all five mutant groups) for the five programs (flex, grep, sed, make, and gzip) for LSTCP; and Figure 6 presents the results for GSTCP. Each plot shows the distribution of the 1,000 APFD values (200 orderings for each of the five versions). Figures 5(f) and 6(f) show the distributions of the APFD values over all the versions of all programs, for LSTCP and GSTCP, respectively.

Table V shows the statistical APFD analyses of all pairwise comparisons for the five programs, based on Figures 5(f) and 6(f). Similar to Table III, the lower triangle (white region) in Table V gives the results for LSTCP; and the upper triangle (gray region) shows the results for GSTCP.

*1) **RQ1: LSTCP Similarity Measures**:* As before, it is clear that different prioritization techniques perform differently for different subject programs — which is reasonable given that different programs have different properties and different mutation faults. More specifically, for the programs flex, grep, and sed, OVE, ESK, GO1, GO2, GO3, GO4, and GAM

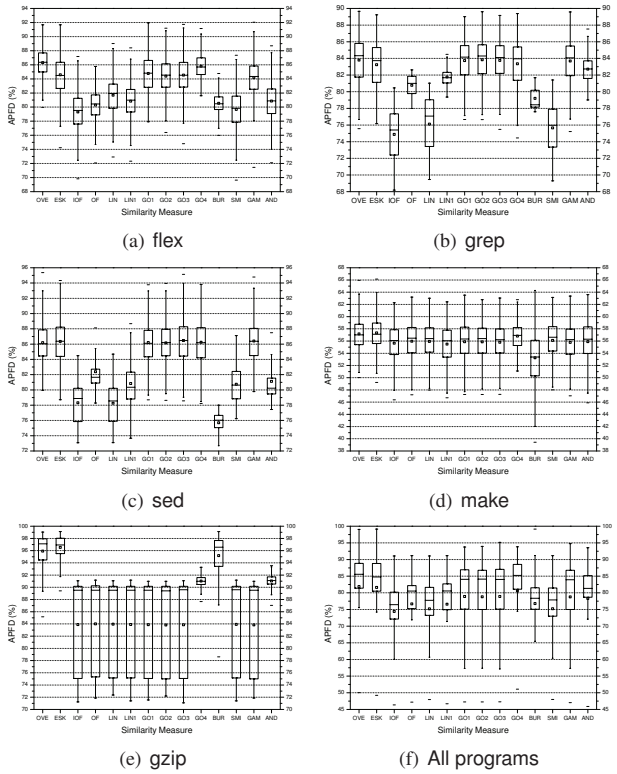


Fig. 5. APFD for each similarity measure in LSTCP on V1 to V5.

TABLE V
STATISTICAL APFD ANALYSIS OF ALL PAIRWISE COMPARISONS FOR FIVE PROGRAMS: LSTCP (LOWER TRIANGLE) AND GSTCP (UPPER TRIANGLE)

	OVE	ESK	IOF	OF	LIN	LIN1	GO1	GO2	GO3	GO4	BUR	SMI	GAM	AND
OVE	–	0.47/0.50	0.71/0.50	0.06/0.51	1.1E-10/0.46	3.8E-15/0.55	7.0E-08/0.53	9.4E-09/0.53	8.5E-09/0.47	0.0*/0.60	0.0*/0.62	0.07/0.51	2.1E-08/0.53	0.0*/0.62
ESK	9.6E-06/0.47	–	0.70/0.50	0.32/0.51	3.7E-12/0.46	3.3E-10/0.54	9.4E-06/0.53	2.5E-06/0.53	5.5E-10/0.46	0.0*/0.59	0.0*/0.61	0.36/0.51	5.2E-06/0.53	0.0*/0.61
IOF	0.0*/0.26	0.0*/0.26	–	0.17/0.51	3.1E-11/0.46	7.8E-14/0.54	4.7E-07/0.53	2.5E-08/0.53	2.1E-09/0.47	0.0*/0.60	0.0*/0.62	0.12/0.51	1.3E-07/0.53	0.0*/0.62
OF	0.0*/0.29	0.0*/0.30	0.0*/0.63	–	1.7E-14/0.46	6.5E-10/0.54	2.8E-04/0.52	1.1E-05/0.53	1.4E-12/0.46	0.0*/0.59	0.0*/0.61	0.75/0.50	1.6E-04/0.52	0.0*/0.61
LIN	0.0*/0.27	0.0*/0.28	8.4E-11/0.54	0.0*/0.42	–	0.0*/0.56	0.0*/0.58	0.0*/0.56	0.33/0.51	0.0*/0.61	0.0*/0.63	1.5E-11/0.54	0.0*/0.56	0.0*/0.63
LIN1	0.0*/0.29	0.0*/0.30	0.0*/0.62	0.73/0.50	0.0*/0.58	–	0.74/0.50	0.30/0.51	0.0*/0.43	0.0*/0.57	0.0*/0.59	1.9E-08/0.47	1.3E-03/0.48	0.0*/0.58
GO1	0.0*/0.40	0.0*/0.43	0.0*/0.68	0.0*/0.63	0.0*/0.66	0.0*/0.63	–	6.1E-05/0.52	0.0*/0.43	0.0*/0.58	0.0*/0.58	7.0E-06/0.47	0.70/0.50	0.0*/0.59
GO2	0.0*/0.40	0.0*/0.43	0.0*/0.68	0.0*/0.63	0.0*/0.66	0.0*/0.63	0.43/0.50	–	0.0*/0.44	0.0*/0.57	0.0*/0.59	7.0E-06/0.47	0.17/0.49	0.0*/0.58
GO3	0.0*/0.41	0.0*/0.43	0.0*/0.68	0.0*/0.63	0.0*/0.66	0.0*/0.63	0.99/0.50	0.43/0.50	–	0.0*/0.61	0.0*/0.63	0.0*/0.54	1.3E-11/0.56	0.0*/0.64
GO4	4.4E-08/0.47	0.22/0.49	0.0*/0.74	0.0*/0.70	0.0*/0.72	0.0*/0.71	0.0*/0.58	0.0*/0.58	0.0*/0.58	–	4.4E-07/0.53	0.0*/0.41	0.0*/0.41	0.01/0.51
BUR	0.0*/0.31	0.0*/0.32	0.0*/0.58	0.0*/0.45	9.2E-14/0.54	0.0*/0.45	0.0*/0.39	0.0*/0.39	0.0*/0.39	0.0*/0.33	–	0.0*/0.39	0.0*/0.40	0.0*/0.49
SMI	0.0*/0.27	0.0*/0.28	5.2E-13/0.54	0.0*/0.42	0.64/0.50	0.0*/0.42	0.0*/0.34	0.0*/0.34	0.0*/0.34	0.0*/0.28	4.3E-12/0.46	–	1.2E-03/0.52	0.0*/0.61
GAM	0.0*/0.40	0.0*/0.43	0.0*/0.68	0.0*/0.63	0.0*/0.66	0.0*/0.63	0.26/0.49	0.73/0.50	0.27/0.49	0.0*/0.42	0.0*/0.61	0.0*/0.66	–	0.0*/0.60
AND	0.0*/0.35	0.0*/0.37	0.0*/0.69	0.0*/0.58	0.0*/0.65	0.0*/0.58	0.0*/0.45	0.0*/0.45	0.0*/0.45	0.0*/0.37	0.0*/0.61	0.0*/0.65	0.0*/0.45	–

*p-value lower than 2.2E-16.

perform similarly in terms of both the mean and median APFD values (within about 2%), and better than other LSTCP techniques. IOF and BUR generally perform worst among all techniques. For make, all techniques other than BUR have only slight differences: their mean and median APFD values range from 55% to 57%). For gzip, however, OVE, ESK, and BUR perform best, followed by AND and GO4, with all other techniques displaying very similar performance. Overall, in terms of APFD, OVE, ESK, and GO4 perform best; and IOF, LIN, BUR, and SMI perform worst.

All the LSTCP p-values for OVE in Table V are highly significant, and all effect size $\hat{A}_{12}$ values are lower than 50% (ranging from 26% to 47%), which means that OVE achieves higher APFD values than other prioritization techniques 53% to 74% of the time. ESK and GO4 have similar performance, and perform better than all techniques other than OVE; and IOF, LIN, BUR, and SMI have the worst performance (according to the p-value/$\hat{A}_{12}$ values). Therefore, the statistical analyses confirm the box plots results.

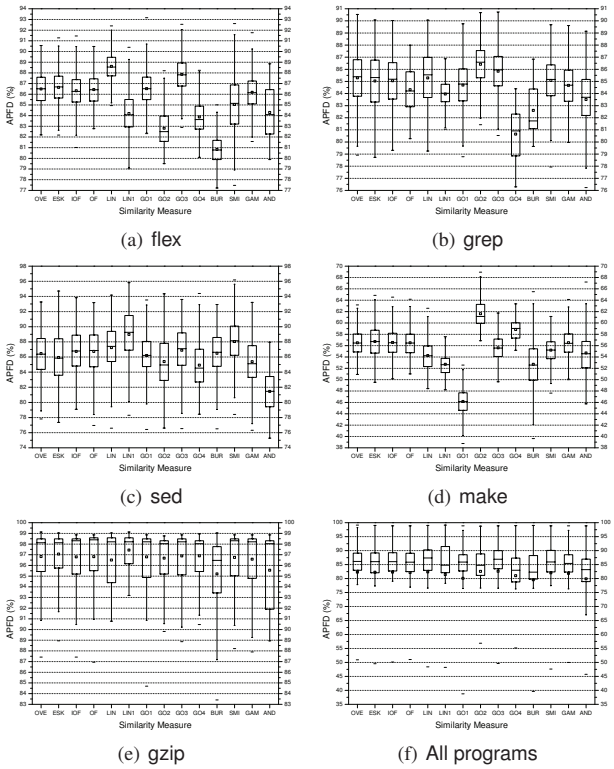
As discussed in the interaction coverage experiments, OVE, ESK, and GO4 have the best APCC values among all LSTCP techniques, which is further confirmed by the fault detection experiments. In other words, for LSTCP, the higher interaction coverage rates relate to higher fault detection rates, and *vice versa*.

*2) **RQ2: GSTCP Similarity Measures**:* Similar to LSTCP, Figure 6 shows that different GSTCP techniques perform differently for different programs.

According to the mean and median APFD values, for the program flex (Figure 6(a)), LIN performs best, followed by GO4; and BUR is worst, followed by GO2. For grep (Figure 6(b)), GO2 and GO3 perform best, and GO4 and BUR perform worst. For sed (Figure 6(c)), the two best techniques are LIN1 and SMI, and the two worst are AND and GO4. GO2 has the best APFD performance for make (Figure 6(d)), followed by GO4; and GO1 performs worst, followed by BUR and LIN1. For gzip (Figure 6(e)), in terms of the mean APFD, LIN1 is best, followed by ESK, with BUR and AND worst; but in terms of the median APFD, all techniques other than BUR perform similarly (ranging from about 98.0% to 98.5%). Considering all programs together (Figure 6(f)), LIN and GO3 perform best in terms of median APFD, with values approaching 87.5%, followed by OVE, ESK, IOF, OF, and GO2; however, in terms of mean APFD, OVE, ESK, IOF, LIN, GO2, and GO3 perform best, with values as high as 82.5%. Overall, LIN and GO3 generally perform best among all GSTCP techniques.

All the GSTCP p-values for LIN and GO3 in Table V are highly significant against other techniques, and the effect size $\hat{A}_{12}$ values range from 53% to 64%. Furthermore, LIN differs only slightly from GO3: their p-value is 0.33 (higher than 0.01), and the $\hat{A}_{12}$ is 0.51. Therefore, it is clear that the statistical analyses support the box plot observations of LIN and GO3 being the two best techniques.

It should be noted that LIN1 can achieve the best interaction coverage rates, but LIN can achieve the best fault detection rates, which means that higher APCC values do not always



(a) flex (b) grep (c) sed (d) make (e) gzip (f) All programs

Fig. 6. APFD for each similarity measure in GSTCP on V1 to V5.

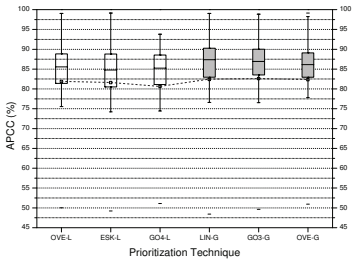


Fig. 7. Average APFD for the three best LSTCP and GSTCP techniques.

imply higher APFD values. Nevertheless, other techniques (for example, GO3) can produce prioritized abstract test cases with both high APCC and high APFD values.

*3) **RQ3: Comparing LSTCP and GSTCP**:* To investigate the differences between LSTCP and GSTCP, we again compared their three best techniques — OVE, ESK, and GO4 for LSTCP; and LIN, GO3, and OVE for GSTCP. As before, we abbreviate these techniques as OVE-L, ESK-L, GO4-L, LIN-G, GO3-G, and OVE-G, respectively.

Figure 7 shows the distribution of the average APFD values for these techniques for all the programs. The joined line through the squared points gives the mean APFD values, according to which the GSTCP techniques are better than the LSTCP ones, with the greatest difference about 1.5%. The median APFD values (the bars inside each box) also show the GSTCP techniques outperforming the LSTCP techniques, with the greatest difference between LIN-G and GO4-L (more than 2.5%), and the smallest difference between OVE-G and OVE-L (approximately 1.0%).

Table VI presents the statistical analyses for the comparisons among the six techniques. In terms of the p-values, the comparisons are highly significant (much lower than 0.01). From the perspective of the effect size $\hat{A}_{12}$, the GSTCP techniques perform much better than LSTCP with results ranging from 76% ($100\% - 24\%$) to 78% ($100\% - 22\%$).

Overall, the differences between the APFD values for L-STCP and GSTCP range from 1.0% to 2.5%, with GSTCP performing better overall than LSTCP 76% to 78% of the time. Therefore, when selecting a single technique out of the 28 studied (14 similarity measures for two algorithms), then, according to the APFD metric, the similarity measures LIN and GO3 for GSTCP may the best choices.

In summary, by combining the APCC and APFD results, we can conclude that the GSTCP algorithm outperforms LSTCP. Furthermore, OVE may be the best choice of similarity measure for LSTCP, and GO3 the best for GSTCP. This conclusion may serve as a guideline for testers, especially when they need to make a selection from many different measures.

C. Threats to Validity

Empirical studies are subject to threats to validity. Although we have attempted to reduce these through our experiment design, we outline some potential threats here.

TABLE VI
STATISTICAL APFD COMPARISON BETWEEN 3 BEST LSTCP AND GSTCP TECHNIQUES

		GSTCP		
		LIN-G	**GO3-G**	**OVE-G**
LSTCP	**OVE-L**	0.0*/0.23	0.0*/0.23	0.0*/0.24
	ESK-L	0.0*/0.23	0.0*/0.23	0.0*/0.23
	GO4-L	0.0*/0.22	0.0*/0.22	0.0*/0.23

*p-value lower than 2.2E-16.

The first potential threat may relate to the five subject programs used in this study. However, we believe that the 30 versions (six versions for each of the five programs) are sufficient to obtain the comparative conclusions for similarity measures in STCP. Additional studies will be conducted in the future using more real-life programs.

Another potential threat relates to the encoding of abstract test cases. In this paper, we focus on the ATCs obtained from the SIR [18] (mainly using the test specification language to obtain the input parameter model and encode ATCs [4]), which is one ATC encoding approach. However, other approaches for encoding ATCs also exist [25], and it will be interesting to investigate the impact of alternative encoding approaches.

V. CONCLUSIONS

Test case prioritization (TCP) aims at scheduling the test cases in test sets such that the speed of fault detection can be improved. TCP is a popular approach, and is often used in practice, including in regression testing [1]. Many TCP algorithms have been proposed, varying according to the different information used. The abstract test case (ATC) is an important type of test case in practical systems such as highly-configurable systems and software product lines. Previous studies have demonstrated that similarity-based prioritization (STCP) is more cost-effective than other prioritization techniques for ATCs [11, 12]. However, because many similarity measures exist to evaluate ATCs, the question of which one to select naturally arises. To answer this question, we conducted a series of empirical studies to investigate the testing effectiveness of a number of similarity measures. The results of these studies show that each similarity measure can perform differently in different STCP algorithms, and that global STCP (GSTCP) is better than local STCP (LSTCP) 61% to 84% of the time, in terms of interaction coverage rates; and 76% to 78% of the time, in terms of fault detection rates. Our studies also show that *Overlap*, the simplest similarity measure listed in this paper, could have the best performance for interaction coverage and fault detection for the local STCP; and that *Goodall3* would be the best choice for the global STCP for testers, especially when they must select only one similarity measure from many choices.

VI. ACKNOWLEDGMENTS

We would like to thank Christopher Henard for providing us the fault data for the five subject programs. This work is supported by the National Natural Science Foundation of China under grant No. 61502205 and 61202110,

the Postdoctoral Science Foundation of China under grant No. 2015M581739 and 2015M571687, the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under grant No. 15KJB520007, and the Senior Personnel Scientific Research Foundation of Jiangsu University under grant No. 14JDG039. This work is also supported by the Young Backbone Teacher Cultivation Project of Jiangsu University.

REFERENCES

[1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[2] M. Grindal, B. Lindström, J. Offutt, and S. F. Andler, "An evaluation of combination strategies for test case selection," *Empirical Software Engineering*, vol. 11, no. 4, pp. 583–611, 2006.

[3] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. L. Traon, "Comparing white-box and black-box test prioritization," in *Proceedings of the 38th Interaction Conference on Software Engineering (ICSE'16)*, 2016, pp. 523–534.

[4] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating functional tests," *Communications of the ACM*, vol. 31, no. 6, pp. 676–686, 1988.

[5] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transaction on Software Engineering*, vol. 30, no. 6, pp. 418–421, 2004.

[6] C. Yilmaz, E. Dumlu, M. B. Cohen, and A. A. Porter, "Reducing masking effects in combinatorial interaction testing: A feedback driven adaptive approach," *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 43–66, January 2014.

[7] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computer Survey*, vol. 43, no. 2, pp. 11:1–11:29, 2011.

[8] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pairwise coverage with seeding and constraints," *Information and Software Technology*, vol. 48, no. 10, pp. 960–970, 2006.

[9] R. Huang, J. Chen, D. Towey, A. Chan, and Y. Lu, "Aggregate-strength interaction test suite prioritization," *Journal of Systems and Software*, vol. 99, pp. 36–51, 2015.

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

[11] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake, "Similarity-based prioritization in software product-line testing," in *Proceedings of 18th International Software Product Line Conference (SPLC'14)*, 2014, pp. 197–206.

[12] 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 Transactions on Software Engineering*, vol. 40, no. 7, pp. 650–670, 2014.

[13] S. Boriah, V. Chandola, and V. Kumar, "Similarity measures for categorical data: A comparative evaluation," in *Proceedings of the SIAM International Conference on Data Mining (SDM'08)*, 2008, pp. 243–254.

[14] Z. Zhang and J. Zhang, "Characterizing failure-causing parameter interactions by adaptive testing," in *Proceedings of the 20th International Symposium on Software Testing and Analysis (ISSTA'11)*, 2011, pp. 331–341.

[15] GNU FTP Server. http://ftp.gnu.org/.

[16] X. Qu, M. B. Cohen, and K. M. Woolf, "Combinatorial interaction regression testing: A study of test case generation and prioritization," in *Proceedings of the 23rd International Conference on Software Maintenance (ICSM'07)*, 2007, pp. 255–264.

[17] cloc: Count Lines of Code. http://cloc.sourceforge.net/.

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

[19] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[20] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.

[21] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 733–752, 2006.

[22] R. Huang, J. Chen, D. Chen, and R. Wang, "How to do tie-breaking in prioritization of interaction test suites?" in *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE'14)*, 2014, pp. 121–125.

[23] T. R. dos Santos and L. E. Zarate, "Categorical data clustering: What similarity measure to recommend?" *Expert Systems with Applications*, vol. 42, no. 3, pp. 1247–1260, 2015.

[24] Z. Wang, L. Chen, B. Xu, and Y. Huang, "Cost-cognizant combinatorial test case prioritization," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 6, pp. 829–854, 2011.

[25] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *Acm Transactions on Software Engineering & Methodology*, vol. 22, no. 1, pp. 139–176, 2013.