Prioritising abstract test cases: An empirical study

Rubing Huang, Weiwen Zong, Tsongyueh Chen, Dave Towey, Yunan Zhou and Jinfu Chen

University of Nottingham Ningbo China, Taikang East Road, Ningbo, Zhenjiang, 315100, China

First published 2019

Prioritizing Abstract Test Cases: An Empirical Study

R. Huang[1]* W. Zong[1] T. Y. Chen[2] D. Towey[3] Y. Zhou[1] J. Chen[1]

[1] School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, Jiangsu 212013, P.R. China
[2] Department of Computer Science and Software Engineering, Swinburne University of Technology, Hawthorn, 3122, Australia
[3] School of Computer Science, University of Nottingham Ningbo China, Ningbo, Zhejiang 315100, P.R. China
* E-mail: rbhuang@ujs.edu.cn

Abstract: Test case prioritization (TCP) attempts to schedule the order of test case execution such that faults can be detected as quickly as possible. TCP has been widely applied in many testing scenarios, such as regression testing, and fault localization. Abstract test cases (ATCs) are derived from models of the system under test, and have been applied to many testing environments, such as model based testing, and combinatorial interaction testing. Although various empirical and analytical comparisons for some ATC prioritization (ATCP) techniques have been conducted, to the best of our knowledge, no comparative study focusing on the most current techniques has yet been reported. In this study, we investigated 18 ATCP techniques, categorized into four classes. We conducted a comprehensive empirical study to compare 16 of the 18 ATCP techniques in terms of their testing effectiveness and efficiency. We found that different ATCP techniques could be cost-effective in different testing scenarios, allowing us to present recommendations and guidelines for which techniques to use under what conditions.

1 Introduction

The order of test case execution in a given test set can be very important, especially when testing resources are limited. The main reason is that a well-prioritized execution order of test cases may be able to trigger failures more quickly, and thus allow the follow-up processes to be conducted earlier (including fault localization, diagnosis and correction). The process of scheduling the execution order of test cases is called test case prioritization (TCP) [1], and it has been applied in various testing environments, including regression testing [2].

To date, many TCP algorithms have been designed to prioritize different test case types according to different criteria, including code coverage based prioritization [1, 3], search based prioritization [4, 5], adaptive random prioritization [6–9], and similarity based prioritization [10, 11] (the interested reader is referred to two survey papers for more details [12, 13]). An abstract test case (ATC) [14] (model input [15]) is an important test case type that can be extracted from a designed model of the system under test (SUT) [16]. In combinatorial interaction testing [17], for example, an SUT may be impacted by different parameters (or factors), each of which may contain a finite number of values (or levels). In this case, ATCs can be created by assigning a value for each parameter. ATCs have been widely used in many testing approaches including model based testing [18], and category-partition testing [16]. Abstract test case prioritization (ATCP) has also been widely studied in different fields, especially in combinatorial interaction testing [19–21], and software product line testing [22, 23].

Although there have been empirical and analytical comparisons of individual or several ATCP techniques [15, 21, 24], to the best of our best knowledge there has not yet been a comprehensive comparative study focusing on the most current techniques. In our study, we investigated 18 ATCP techniques, grouped into four categories: non-information-guided prioritization (NIGP); interaction coverage based prioritization (ICBP); input-model mutation based prioritization (IMBP); and similarity based prioritization (SBP). We conducted a comprehensive empirical study using five subject programs (written in the C language), each of which had six versions. In the study, based on mutation analysis, the testing effectiveness and efficiency of each ATCP technique were investigated.

We believe that this is the most extensive and inclusive empirical study comparing ATCP techniques so far reported in the litera-ture. Based on the experimental results, some empirical findings are provided, and some recommendations and guidelines are given for testers when choosing ATCP techniques in different testing sce-narios. In summary, the main contributions of this work are as follows:

(1) We selected 18 ATCP techniques from the literature, and divided them into four categories, in terms of the different information used to guide the prioritization process.
(2) We conducted empirical studies to compare 16 of the 18 ATCP techniques, according to three quality evaluation measures: interac-tion coverage rate, fault detection rate, and prioritization cost.
(3) We present empirical findings comparing ATCP techniques among each category and between different categories.
(4) We provide recommendations and guidelines for testers to help select ATCP techniques in different testing scenarios.

The structure of the rest of this paper is organized as following: Section 2 introduces some preliminary information and background details. Section 3 provides the details about the experimental set-tings, and Section 4 presents the experimental results to answer the research questions. Finally, Section 5 concludes the paper, and discusses potential future work.

2 Preliminaries and Background

Some preliminary information is presented in this section, includ-ing details about abstract test cases, and test case prioritization (TCP). ATCP techniques are described, the strength and weakness of each technique are summarized, and previous empirical work is also discussed.

2.1 Preliminaries

2.1.1 Abstract Test Case:
A system under test (SUT) is gen-erally influenced by different parameters or factors (for example, configurations, features, components, etc), with each parameter hav-ing a certain number of possible values or levels. In general, most

Table 1 An example for input model

Factor	p1: **OS**	p2: **Browser**	p3: **Access**	p4: **Proxy**
Level	Windows (0)	IE (2)	ISDL (4)	No Proxy (7)
	Mac OS X (1)	Safari (3)	Modem (5)	HTTP (8)
			VPN (6)	SOCKS5 (9)

Browser = "IE" ! **OS** = "Windows", i.e., p2 = "2" ! p1 = "0".
Browser = "Safari" ! **OS** = "Mac OS X", i.e., p2 = "3" ! p1 = "1".

SUTs may have constraints among parameter values: that is, some value combinations are not feasible. Based on this, we present the following definition of an input model [25] (or input parameter model [14]) used for modeling the SUT.

Definition 1. An input model, Model({p1, p2, · · · , pk}, {L1, L2, · · · , Lk }, C), is the information about the parameters and the values of each parameter of the SUT (with k parameters), a set of values Li for the i-th parameter pi, and a set of value combination constraints C.

As shown in Table 1, for example, an input model with value combination constraints is used for a web application such as a browser game, where four parameters are included, of which the first has two values, and the last three all have three. Since the browser "IE" is developed for the OS "Windows", and the browser "Safari" is developed for the OS "Mac OS X", two value combination constraints are obtained. To simplify the problem, each parameter is denoted by pi (i = 1, 2, 3, 4), and each value is labelled by an integer, beginning with 0 and incrementing by 1, from p1 to p4 (see Table 1).

Therefore, the model for above example can be represented by Model({p1, p2, p3, p4}, {{"0", "1"}, {"2", "3"}, {"4", "5", "6"}, {"7", "8", "9"}}, C = {p2 = "2" ! p1 = "0", p2 = "3" ! p1 = "1"}, containing two value combination constraints, and four parameters, of which the first two parameters have two values, and another two parameters have three values. Since the detailed values of each parameter provide no influence on the model, without loss of generality, we adopt an abbreviated version in this paper: Model(|L1||L2| · · · |2k|2 C . Accordingly, the above example can be described as Model(2 3 , C = {"2" ! "0", "3" ! "1"}).

When an input model is available, construction of abstract test cases (ATCs) [14] (or model inputs [15]) for testing the SUT is possible. The definition of the abstract test case is given as follows.

Definition 2. An abstract test case, (v1, v2, · · · , vk), is a k-tuple, where vi 2 Li (i = 1, 2, · · · , k).

An ATC is valid if C is satisfied, otherwise it is invalid. For instance, in the previous example, a valid ATC is (0, 2, 5, 8); and an invalid one is (0, 3, 4, 8) — due to violation of the constraint ((p2 = 3) ! (p1 = 1)). Intuitively speaking, each ATC with size □ can cover λ-tuples 1 λ □ , where such a tuple is called a λ-wise value combination [26] or a λ-wise schema [17]. For example, an ATC (1, 3, 5, 9) covers six 2-wise value combinations: (1, 3), (1, 5), (1, 9), (3, 5), (3, 9), and (5, 9).

The ATCs have been used in many applications such as configuration-aware systems [27, 28], and software product lines [29]. Many testing methods have focused on the generation and construction of ATCs, such as category-partition testing [16], combinatorial testing [17], and random testing [30].

2.1.2 Test Case Prioritization: Test case prioritization seeks to schedule test cases such that those with the highest significance, in terms of some criteria, are run earlier than those with lower significance. When testing resources are limited or insufficient for the execution of a complete test suite, then a good order of test case execution can be very important. The problem of test case prioritization can be defined as follows [1].

Definition 3. Given a test suite T to be prioritized, □ being the set of all possible orders of test cases by permuting T , and f being a fitness function to evaluate each permutation, the problem of test case prioritization is to identify a permutation S 2 □ such that:

(8S⁰) (S⁰ 2 □) (S⁰ 6= S) [f(S) f(S⁰)] (1)

2.2 ATCP Techniques

Depending on the type of information used, as with other testing approaches, ATCP can be considered either black-box or white-box testing [15]. ATCP approaches using models of the SUT, for example, would be considered black-box, because no access to source code is necessary. In this paper we focus on black-box ATCP techniques (interested readers may refer to work by Rothermel et al. [1] or Zhang et al. [31] for discussion of white-box approaches). According to the information used to guide the prioritization process, the ATC prioritization techniques (ATCP) are mainly classified into the following four categories.

2.2.1 Non-Information-Guided Prioritization (NIGP): The NIGP strategies discussed in this section can be used for not only abstract test cases but for all types of test cases, because this category does not use additional information to support the prioritization process.

• Test-case-generation prioritization (TCGP): TCGP prioritizes ACTs using the order in which the test cases were generated.
• Reverse test-case-generation prioritization (RTCGP): RTCGP prioritizes ACTs by reversing the generation order.
• Random test case prioritization (RTCP): RTCP randomly orders ACTs, according to uniform distribution.

2.2.2 Interaction Coverage Based Prioritization (ICBP): The ICBP strategy makes use of the information of coverage information to support the process of ATCP. By using different levels of interaction coverage, the following three ATCP techniques are considered: fixed-strength ICBP (FICBP), incremental-strength ICBP (IICBP), and aggregate-strength ICBP (AICBP).

• Fixed-strength ICBP (FICBP): FICBP [32] iteratively selects the element as the next test case from candidate ATCs such that it covers the largest number of λ-wise value combinations that have not yet been covered by the ATCs already selected. Before prioritization, FICBP needs to assign a value to an integer λ, the prioritization strength. Based on previous investigations [21, 24, 33–35], the assignment of the prioritization strength usually ranges from 1 to 6. To reduce the prioritization cost, a new FICBP technique has been proposed that uses repeated base-choice coverage, FICBPR [36]. Although FICBPR leverages a similar mechanism to FICBP, it only assigns a value of 1 to the prioritization strength λ, and forgets previous prioritization details when the coverage of 1-wise value combinations is fully achieved.
• Incremental-strength ICBP (IICBP): IICBP [37, 38] first uses a small prioritization strength λ (λ 1), and presents it to the FICBP algorithm for prioritizing the candidates. Once all λ-wise value combinations have been covered by selected test cases, IICBP increases the prioritization strength with an increment i — λ = λ + i (i 1) — and then uses this new prioritization strength for the FICBP algorithm to prioritize remaining ATCs. This process is repeated until all candidates have been chosen. In this study, we used the IICBP algorithm from Huang et al. [38], initially setting λ to 1, and i to 1.
• Aggregate-strength ICBP (AICBP): AICBP [20] makes use of hybrid interaction coverage by considering different prioritization strength λ values ranging from 1 to the generation strength □ in combinatorial testing [17]. As we know, □ is chosen in the stage of test suite construction, however, it may be not applicable to adopt previous AICBP algorithms for prioritizing ATC sets (because it is infeasible to choose the value of □). In this paper, therefore, we use a simplified version of AICBP that only takes prioritization strength λ = 1, 2, and 3 into consideration (i.e., □ = 3), and can thus be used for prioritizing any sets of ATCs. The mechanism of the AICBP algorithm is similar to that of FICBP, except that AICBP uses hybrid interaction coverage by aggregating three

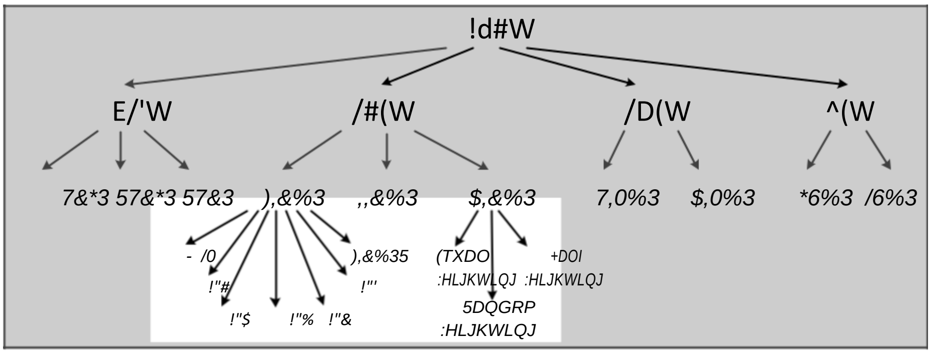


Fig. 1: Overview of ATCP techniques

prioritization strengths 1, 2, and 3) rather than interaction coverage by using a single prioritization strength. As discussed by Huang et al. [20], there are three weighting distributions for different prioritization strengths, i.e., three ways of assigning the weightings !1, !2, · · · , and !□ to the prioritization strength λ1, λ2, · · · , and λ□ , respectively, where !1 + !2 + · · · + !□ = 1.0. More specifically, Equal Weighting assigns the same weighting to each prioritization strength, i.e., !1=!2=···=!□ = $\frac{1}{□}$; Random Weighting randomly assigns the weighting to each prioritization strength; and Half Weighting sets the weighting as following: !1 = !, !j+1 = $\frac{1}{2}$!j , and !□ = 1.0 (!1 +!2 +···+!□ 1).

2.2.3 Input-model Mutation Based Prioritization (IMBP): The IMBP strategy [15] creates the mutants of the flattened model that is derived from the SUT's input model, and then uses the mutant detection capability of each test case to guide the process of ATCP. More specifically, IMBP first mutates the flattened model from [25] to obtain a mutant by changing a constraint, for example, the constraint from the input model is ("2" ! "0"), and a mutant may be ("2" ! "1"). The mutants that are distinguished by the test cases are killed; otherwise they are live. After that, IMBP prioritizes test cases based on their capabilities of killing mutants. Based on different selection strategies, two IMBP techniques are included: 'total' IMBP (TIMBP) and 'additional' IMBP (AIMBP) [15].

- Total IMBP (TIMBP): TIMBP refers to previous 'total' TCP strategies [1, 31], by repeatedly choosing each element as the next test case from the remaining candidates such that it kills the maximum (total) number of model mutants.
- Additional IMBP (AIMBP): Similar to TIMBP, AIMBP refers to previous 'additional' TCP strategies [1, 31], by repeatedly selecting the next test case which can kill the largest number of model mutants that have not yet been detected by previously selected ATCs.

2.2.4 Similarity Based Prioritization (SBP): SBP [23] makes use of the Jaccard similarity between candidates to prioritize test cases, with each ATC being a set of parameter values. In particular, SBP selects each next test case such that it achieves the smallest similarity to previously selected test cases. Based on different implementations, Henard et al. [23] introduced two versions of SBP: global SBP (GSBP) and local SBP (LSBP).

- Global SBP (GSBP): GSBP first determines the first two test cases by choosing two elements from candidates with the minimum similarity, then iteratively selects a candidate as the next test case. In detail, for each candidate c, GSBP first calculates the similarity between c and each already selected ATC, and sums the similarity values as the fitness value of c. Then the candidate with the minimum fitness value is chosen as the next test case.
- Local SBP (LSBP): LSBP iteratively identifies a pair of candidates sharing the minimum similarity as the next two test cases, until all candidate test cases are selected. The order of the two test cases is determined in a random manner.

Figure 1 shows an overview of ATCP techniques, involving four categories with 18 techniques.

2.2.5 Strengths and Weaknesses: In this section, we briefly summarize the strengths and weaknesses of ATCP techniques, listed as follows:

(1) For the NIGP category, its main advantage may be high testing efficiency (for example, low prioritization time); however, its disad-vantage may be low testing effectiveness. The main reason for this is that the NIGP category does not use additional information to guide the prioritization process.
(2) As for the ICBP category, its main benefit is that each ATCP technique makes use of the information of interaction coverage to prioritize ATCs, resulting in high testing effectiveness. Regard-ing the drawbacks, FICBP may face the challenges of choosing an appropriate prioritization strength, as different prioritization strengths may lead to different testing performances; and AICBP may require more prioritization time, because it uses more infor-mation for the prioritization. IICBP can be considered as a balanced technique compared with FICBP: it may have better testing effec-tiveness than FICBP with low prioritization strengths but less testing efficiency than that with high prioritization strengths.
(3) For the IMBP category, its main strength is that it brings the concept of mutation analysis [39] to the input model of the system under test, which may provide some new insights for ATCP. However, it may face some potential challenges, for example, the quality of mutants may influence the performance of IMBP. Gen-erally speaking, AIMBP may have better testing effectiveness but worse testing efficiency than TIMBP, because it requires collecting more information.
(4) Regarding the SBP category, its main strength is that it may achieve high testing efficiency with comparable testing effectiveness to FICBP. However, it may suffer from the drawback of needing to choose the appropriate similarity measure between ATCs. Intuitively speaking, GSBP may have better performance than LSBP, because the former adopts more information for choosing each element from candidates as the next test case.

Based on this analysis, when testing resources are limited, it may be better to use FICBP with a low prioritization strength, SBP, or NIGP. On the contrary, when testing resources are sufficient, it may be better to adopt FICBP with a high prioritization strength, IICBP, or AIMBP. Additionally, the selection of IMBP may depend on the input model of the system under test.

2.3 Previous Empirical Work

In this section, we report on some previous empirical work into the prioritization of abstract test cases.

Petke et al. [24] initially investigated FICBP with the prioritization strength λ values 2, 3, 4, and 5; and later added λ = 6 in an extended study [21]. They mainly focused on the analysis of different prioritization strength values used in FICBP for different covering arrays constructed for combinatorial testing [17]. Compared with their work, however, our study examines most current ATCP techniques, including, but beyond, FICBP.

Henard et al. [23] proposed two similarity based ATCP algorithms, (GSBP and LSBP), and compared them with the random test case prioritization and 2-wise FICBP technique. Similar to Petke et al. [24], their work focused on the prioritization of combinatorial test suites (i.e., covering arrays). Additionally, they focused mainly testing software product lines, which means that the input models used were binary — each parameter containing exactly two possible values.

Henard et al. [15] compared 20 TCP techniques (ten for white-box and ten for black-box) — some of their black-box prioritization techniques have also been considered in our study. Nevertheless, their study focused on the comparison of white-box and black-box test prioritization techniques, whereas our study is a comparison of black-box ATCP techniques.

2.4 Research Questions

Our study was motivated by a number of outstanding issues in the field of ATCP. The following five research questions (RQs) guided the study in this paper.

RQ1: How well do the three ICBP strategies studied perform in terms of the rates of interaction coverage and fault detection?

– For the FICBP methods, which strength is more suitable for prioritizing ATCs?

– For the AICBP methods, which weighting distribution is more effective?

– Which level of interaction coverage is adequate for the ICBP?

Answering RQ1 will help testers identify which interaction-coverage-based technique is the most effective. For some ICBP sub-categories, we also had sub-questions to further investigate their effectiveness, and also analyzed the main influential parameters. All ICBP methods use interaction coverage information to guide the prioritization — but they use different levels of interaction coverage. It is therefore meaningful to study which level of interaction coverage is adequate.

RQ2: How well do the two IMBP techniques studied perform according to the rates of interaction coverage and fault detection?

Answering RQ2 will help testers know which technique is the most suitable for IMBP. Previous studies based on code coverage information [1, 31] have shown that the 'additional' TCP tech-nique performs better than the 'total' TCP technique, but there are no reported observations related to input-model mutation coverage information. It is therefore interesting to investigate this issue.

RQ3: How well do the two SBP techniques studied perform in terms of interaction coverage rate and fault detection rate?

Answering RQ3 will help testers know which technique is the most suitable for SBP. Previous investigations have indicated that the SBP strategy is an effective technique for ATCs [22, 23], how-ever the comparison between GSBP and LSBP has not yet been fully explored.

RQ4: How differently do the NIGP, ICBP, IMBP, and SBPS techniques perform, according to interaction coverage rate and fault detection rate?

Answering RQ4 will help guide testers in their selections. It is useful for testers to know which prioritization technique, among all studied techniques, has the best performance.

RQ5: How do all the ATCP techniques compare in terms of the required prioritization time?

ATCP is important, especially when testing resources are too limited to allow execution of all ATCs. It is therefore useful to consider the prioritization time of each prioritization technique. Answering RQ5 will help testers make a decision on the selection of prioritization techniques.

3 Methodology

3.1 Subject Programs

Five subject programs, written in the C language, were chosen. These programs were obtained from the GNU FTP server*. The flex program is a fast lexical analysis generator; the grep program is a widely-used utility for pattern matching; the sed program is a stream editor that performs text transformations on an input stream; the make program is to control the compile and build processes for programs; and the gzip program is a popular command-line tool used for file compression and decompression. These programs have been widely adopted in previous TCP research [1, 7, 15, 21, 24, 34, 35].

Table 2 describes the detailed information for each subject program such as, the version number, the year of release, the uncommented size of code (measured by cloc†), and the number of

* http://ftp.gnu.org/

† http://cloc.sourceforge.net/

Table 2 Subject Programs

Program	Input Model	Test Pool	Information	V0	V1	V2	V3	V4	V5
flex	Model($2^{6}3^{2}5^{1}$, C1), |C1| = 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($2^{1}3^{3}4^{2}5^{1}6^{1}8^{1}$, C2), |C2| = 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($2^{7}3^{1}4^{1}6^{1}10^{1}$, C3), |C3| = 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}, C4), |C4| = 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($2^{13}3^{1}$, C5), |C5| = 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

mutated faults. The table also gives the information of input models and sizes of candidate ATCs, where all input models and ATC sets were downloaded from a standard library, i.e., the Software Infrastructure Repository (SIR) [40]. These input models were used in previous work by Petke et al. [21, 24])

3.2 Fault Seeding

For each of the subject programs, the original version does not contain any seeded-in faults. There are a number of hand-seeded faults that are available from the SIR [40], but many of these faults can be detected by more than 60% of test cases (on average). Therefore, in this paper we have used mutation analysis [39] to evaluate different ATCP techniques. As discussed in previous studies [41, 42], mutation analysis can provide more realistic faults than hand-seeding, and may be more appropriate for studying test case prioritization.

For the five subject programs, we used the same mutation faults as used by Henard et al. [15]: that is, we employed the mutant operators set used by Andrews et al. [41], including statement deletion, constant replacement, unary insertion, arithmetic operator replacement, logical operator replacement, relational operator replacement, and bitwise logical operator replacement. Following previous practice [1, 31, 41], we removed the duplicate and equivalent mutants, and also removed all those mutants that would not be killed by any ATC. In addition, all subsuming mutants [43] (also called minimum mutants [44] or disjoint mutants [45]) that would be too easily killed were also removed — these mutants may otherwise negatively affect the mutation score measurement [41, 44–46]. A mutation fault is said to be identified by a test case when the output of the original version is different to that of the fault-seeded version. Table 2 shows the number of faults in this study.

3.3 The 16 Investigated ATCP Algorithms

Table 3 presents an overview of the 16 ATCP techniques studied, giving the mnemonic, description, a reference to its original research publication, and category, for each. For NIGP, we only considered random test case prioritization, because test-case-generation prioritization (TCGP), and its reversed version (RTCGP), only depend on the original test set. However, because the test pool used in this paper was provided by the SIR [40], which has no correspondence for the original or reversed set, therefore, TCGP and RTCGP were removed from the experiments. For FICBP, we considered the prioritization strengths λ = 1, 2, 3, 4, 5, and 6. For AICBP, we considered three weighting distributions of prioritization strengths: equal, random, and half weighting [20]. For IMBP, the model mutants needed to be seeded, and in this study we used the model mutant matrix file[*] used by Henard et al. [15].

For SBP, compared to the previous versions of GSBP and LSBP [23], the algorithms in our study have two main differences:

[*] ~~http://henard.net/research/regression/ICSE 2016/~~

Table 3 ATCP techniques considered in the experiments

Mnemonic	Description	Reference	Category
RDP	Random test case prioritization	[32]	NIGP
FP1	FICBP at prioritization strength 1	[33]	ICBP
FP2	FICBP at prioritization strength 2	[32]	
FP3	FICBP at prioritization strength 3	[32]	
FP4	FICBP at prioritization strength 4	[38]	
FP5	FICBP at prioritization strength 5	[37]	
FP6	FICBP at prioritization strength 6	[21]	
FPR	FICBPR	[36]	
IIP	IICBP	[37]	
APE	AICBP with Equal Weighting	[20]	
APR	AICBP with Random Weighting	[20]	
APH	AICBP with Half Weighting	[20]	
TIM	TIMBP	[15]	IMBP
AIM	AIMBP	[15]	
SPG	GSBP	[23]	SBP
SPL	LSBP	[23]	

(1) When meeting a tie-breaking case, i.e., there exist more than one pair of ATCs sharing the same minimum similarity, the original version adopts a first-test-case tie-breaking technique (i.e., choosing the first one) [47]. However our study uses the random tie-breaking technique (i.e., choosing a pair randomly); (2) After choosing the best pair of ATCs from candidates, the original version adds these two ATCs to the prioritized set successively, however our study adds them in a random order. Our GSBP and LSBP algorithms, therefore, are less biased than the originals.

Because the ATCP techniques involve randomization (due to random tie-breaking [47]), we ran each experiment 100 times.

3.4 Metrics

To evaluate different ATCP techniques, in this study we focused on the following three metrics: (a) interaction coverage rate — to measure the speed of achieving the interaction coverage of each prioritized test suite; (b) fault detection rate — to measure the speed of identifying faults of each prioritized test suite; and (c) prioritization cost — to measure how quickly each prioritized test suite was obtained.

3.4.1 Interaction Coverage Rate:
The average percentage of combinatorial coverage (APCC) [24, 48] was adopted to evaluate the speed of achieving the interaction coverage at strength λ for a prioritized set of ATCs. If $S = \langle tc_1, tc_2, \cdots, tc_n \rangle$ is an ordered set of n ATCs, the λ-wise ($1 \le \lambda \le k$) APCC definition for S is:

$$APCC(\lambda, S) = \frac{\sum_{i=1}^{n-1} \left| \bigcup_{j=1}^{i} CombSet(\lambda, tc_j) \right|}{n \times \left| \bigcup_{j=1}^{n} CombSet(\lambda, tc_j) \right|} \quad (2)$$

where $CombSet(\lambda, tc_j)$ is a set of λ-wise value combinations covered by the abstract test case tc_j.

The APCC metric values are numerical values ranging from 0.0 to 1.0, with higher values implying better rates of achieving interaction coverage. Following previous investigations [21], in this paper, six λ values from 1 to 6 were considered for APCC.

3.4.2 Fault Detection Rate:
The average percentage of faults detected (APFD) was previously used to evaluate different prioritization techniques [1],. APFD requires details of the fault-detection capability of each executed test case.

Let T be a test suite with size n, and F be a set of m faults that can be detected by T. Let SF_i be the number of test cases, required to detect fault $F_i \in F$, in a prioritized test suite S of T. The APFD for S is given by the following equation (from [1]):

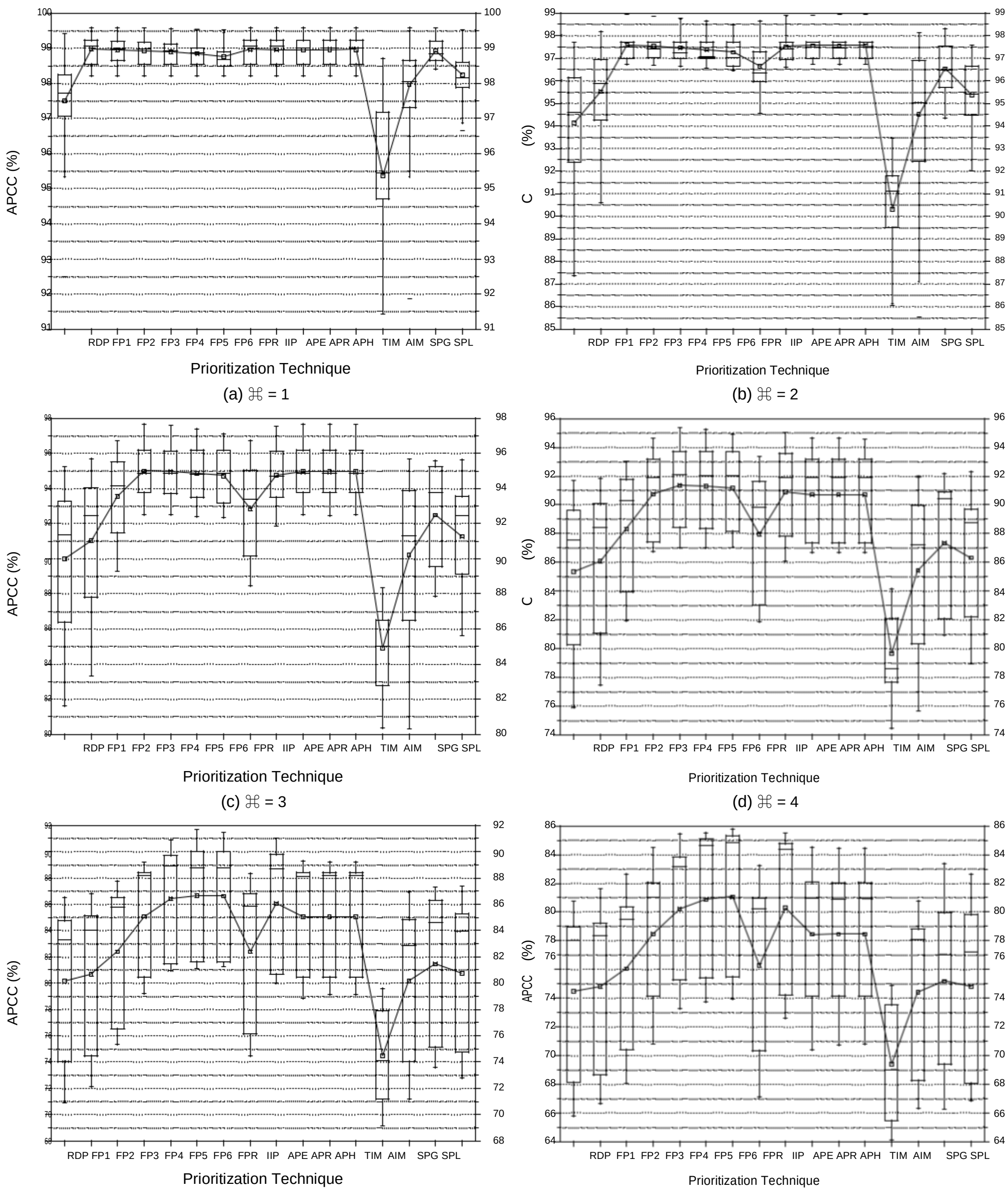
$$APFD(S) = 1 - \frac{SF_1 + SF_2 + \cdots + SF_m}{n \times m} + \frac{1}{2n} \quad (3)$$

3.4.3 Prioritization Cost:
The prioritization cost measures the prioritization time required for each prioritization technique, and represents the efficiency of the technique. Obviously, lower prioritization costs mean better performance.

3.5 Statistical Analysis

When assessing the statistical significance of the differences between the APCC or APFD values (used to evaluate each prioritization technique), because there was no relationship between any of the 100 runs, it is reasonable to use an unpaired test [49]. Furthermore, since no assumptions were made about which prioritization technique is better than others, a two-tailed test is also appropriate [49]. Following previous studies dealing with randomized algorithms [49, 50], we used the unpaired two-tailed Wilcoxon-Mann-Whitney test of statistical significance (set at a 1% level of significance).

Because multiple statistical prioritization techniques were employed, we report the p-values — as the number of executions increases, p becomes sufficiently small [15], which means that there are differences between the two algorithms. We used the non-parametric Vargha and Delaney effect size measure [51], $\hat{A}_{12}$, where the further away from 0.5, the larger the effect size. The effect size can be also represented as the probability that one technique is better than another — with a higher effect size (value) indicating higher probability. For example, $\hat{A}_{12}(x, y) = 1.0$ indicates that, based on the sample, algorithm x always performs better than algorithm y; and $\hat{A}_{12}(x, y) = 0.0$ means that it always has worse performance. Based on the classification [51], the effect size is categorized into one of three degrees — Small (S), Medium (M), or Large (L) — where Small means $|\hat{A}_{12}(x, y)$ 0.5| < 0.1; Medium means 0.1 $|\hat{A}_{12}(x, y)$ 0.5| < 0.17; and Large means 0.17 $|\hat{A}_{12}(x, y)$ 0.5|. The p-value and effect size $\hat{A}_{12}$ value were calculated for each pair-wise comparison of the prioritization techniques.



(a) ⌘ = 1 (b) ⌘ = 2 (c) ⌘ = 3 (d) ⌘ = 4 (e) ⌘ = 5 (f) ⌘ = 6

Fig. 2: APCCs for each ATCP technique for the five subject programs

4 Results

In the plots in Figures 2, 3, and 4, the X-axis shows the ATCP techniques investigated, while the Y-axis lists the APCC or APFD values. Each box plot describes the mean (a square in the box), median (a line in the box), lower/upper quartile, and min/max APCC or APFD values.

4.1 APCC Results

Figure 2 presents the APCC results at different ⌘ (1 ⌘ 6) values. Figure 3 gives the average APCC values over the six ⌘ values, in which each plot describes the distribution of the 500 APCC values (100 orderings ⇸ 5 programs) at ⌘. Table 4 shows the statistical results for comparing any two techniques based on Figure 3.

4.1.1 RQ1: APCC Effectiveness: ICBP: Regarding the FICBP techniques (the first subquestion of RQ1), the FPλ (1 λ 6) generally has the best APCCs at ⌘ (1 ⌘ 6), when ⌘ is equal to the prioritization strength λ. However, not every FPλ always performs best at prioritization strength ⌘ = λ, because local optimization instead of global optimization was applied. In other words, no FICBP method always has the highest APCC values. These observations are consistent with those reported in other studies [21, 24, 38]. Furthermore, at a fixed ⌘ (1 ⌘ 6), when λ increases, FPλ achieves higher APCC while 1 λ ⌘; but lower APCC when ⌘ λ 6. According to the average APCC over the six values of ⌘ (Figure 3), FP4, FP6, and FP5 are the three best FICBP techniques, followed by FP3, and FP2; and FP1 performs worst. Table 4 shows the APCC inferential statistical analysis, which confirms the box plot results. As a consequence, the prioritization strength λ should be assigned a value of at least 4, if we wish to achieve the best performance (according to the interaction coverage rate).

Regarding the AICBP techniques (the second subquestion of RQ1), all three weighting distributions of prioritization strengths have very similar APCC values, irrespective of ⌘ and program. According to the statistical analysis, the p-values for comparisons between any two techniques is greater than 0.01; and the effect size measure Â12 is approximately equal to 50%, which confirms the plot observations. Therefore, the weighting distribution has only a very slight impact on the AICBP techniques.

To answer the last subquestion of RQ1, we compared all eleven ICBP techniques (FPi (i = 1, 2, 3, 4, 5, 6), FPR, APE, APR, APH, and IICBP). Based on this comparison, we observe the following:

- When ⌘ = 1 (Figure 2(a)), FP6 and FP5 have the worst performance, and the other nine ICBP techniques perform similarly (with FP1, FPR, and IICBP having slightly higher APCC results than others).
- When ⌘ = 2 (Figure 2(b)), FP1 is worst, followed by FPR, FP6, and FP5; and all other techniques have similar APCC results.
- When ⌘ = 3 (Figure 2(c)), FP1 and FPR have the worst ICBP performance, followed by FP2; and all other techniques are similar.
- When ⌘ = 4, 5, 6, FP4, FP5, and FP6 generally have the high-est APCC values, followed by IICBP. FP1, FP2, and FPR generally perform worst.
- Based on the average APCC results, FP4, FP5, and FP6 are the three best ICBP techniques, followed by IICBP. The next best tech-niques are FP3, and the AICBP series. FP1 is worst, followed by FPR and FP2. The statistical analysis also confirms these observations.

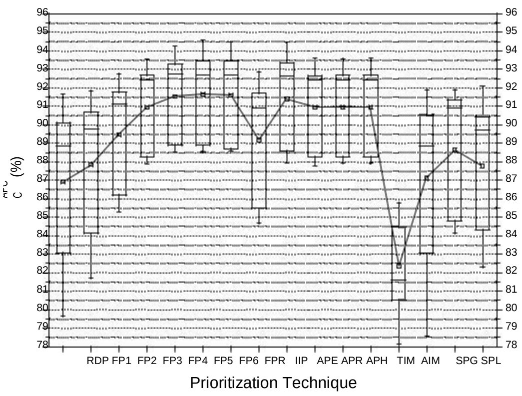


Fig. 3: Average APCC distribution (for ⌘ = 1, 2, 3, 4, 5, 6) for each ATCP technique

4.1.2 Q2: APCC Effectiveness: IMBP: Based on the experi-mental data, it is clear that AIM has much higher APCC values than TIM, regardless of ⌘values. Therefore, AIM also has much higher average APCC values, which is confirmed by the statistical analysis: the p-value is less than 2.04E-72, indicating a significant difference between them; and the effect size measure $\hat{A}_{12}$ is 0.1712, indicating that AIM performs better than TIM about 83% of the time.

4.1.3 RQ3: APCC Effectiveness: SBP: When 1 ⌘ 4, SPG has significantly better ⌘-wise APCC results than SPL; how-ever, when ⌘= 5, 6, SPG is better than SPL, although the differ-ences are small. Based on the statistical analysis, it is clear that SPG performs significantly better than SPL: their p-value is much less than 0.01, which indicates high significance; and their $\hat{A}_{12}$ is 0.6927, which means that SPG has a probability of about 69% of obtaining higher APCC values than SPL.

4.1.4 RQ4: APCC Effectiveness of All Techniques: Consid-ering all sixteen ATCP techniques, we can observe that as ⌘ (1 ⌘6) increases, the APCC values of each prioritization technique decrease, which is expected, due to the characteristics of the APCC metric (Section 3.4.1). More specifically, given a candidate ATC set T , the number of ⌘-wise value combinations covered by T is generally much larger than that of ⌘^{0}-wise value combinations, when 1 ⌘^{0} < ⌘ 6. In other words, the number of ⌘-wise value com-binations covered by T increases as ⌘increases. For each prioritized set S of T , therefore, the speed of covering ⌘^{0}-wise value combi-nations is faster than that of covering ⌘-wise value combinations: APCC(S,⌘^{0}) > APCC(S,⌘).

Among all techniques, TIM generally has the worst performance: this is a surprising result, because it performs worse than RDP, which does not use any information to guide the prioritization process. Additionally, the ICBP series has better APCC results than any other series, such as NIGP, IMBP, and SBP; with SBP as the second best (it should be noted that SPG is better than FP1), followed by IMBP. This observation is also understandable, because the ICBP series uses the interaction coverage information to guide the prioritization, giving higher interaction coverage rates. In addition, the SBP series does not use interaction coverage for prioritizing ATCs, but the sim-ilarity comparison between two test cases effectively achieves this interaction coverage: guaranteeing that at least two test cases could cover the largest number of value combinations at strength 1. How-ever, the IMBP series prioritizes test cases according to the model mutation scores, and hence no interaction coverage is considered for the prioritization.

To conclude, ICBP is the best, with fixed-strength ICBP at higher prioritization strength λ giving the best APCC scores (it is recom-mended that λ be assigned a value of at least 4), and incremental-strength and aggregate-strength ICBP delivering comparable APCC results. SBP is the second best, with the global SBP achieving APCC results comparable to the ICBP series, and better than the local SBP. A surprising result is that NIGP (such as RDP) could some-times achieve better performance than IMBP, according to the APCC values.

Table 4 Statistical APCC analysis of all pairwise comparisons (A, B)

A	B	p-value	Superior	Effect Size	A	B	p-value	Superior	Effect Size
RDP	FP1	8.50E-15	FP1	0.3583 (M)	FP4	APH	6.96E-22	FP4	0.6756 (L)
RDP	FP2	3.56E-44	FP2	0.2453 (L)	FP4	TIM	5.86E-165	FP4	1.0000 (L)
RDP	FP3	1.30E-48	FP3	0.2324 (L)	FP4	AIM	2.22E-59	FP4	0.7968 (L)
RDP	FP4	2.68E-58	FP4	0.2060 (L)	FP4	SPG	5.67E-46	FP4	0.7600 (L)
RDP	FP5	2.24E-58	FP5	0.2058 (L)	FP4	SPL	1.05E-46	FP4	0.7621 (L)
RDP	FP6	3.59E-56	FP6	0.2116 (L)	FP5	FP6	0.0059	FP5	0.5503 (S)
RDP	FPR	6.19E-42	FPR	0.2522 (L)	FP5	FPR	3.70E-42	FP5	0.7485 (L)
RDP	IIP	3.52E-51	IIP	0.2251 (L)	FP5	IIP	3.77E-10	FP5	0.6144 (M)
RDP	APE	1.27E-48	APE	0.2323 (L)	FP5	APE	1.29E-27	FP5	0.6989 (L)
RDP	APR	1.15E-48	APR	0.2322 (L)	FP5	APR	1.01E-28	FP5	0.7031 (L)
RDP	APH	1.22E-48	APH	0.2323 (L)	FP5	APH	5.25E-28	FP5	0.7004 (L)
RDP	TIM	2.97E-71	RDP	0.8260 (L)	FP5	TIM	5.86E-165	FP5	1.0000 (L)
RDP	AIM	0.0273	AIM	0.4597 (S)	FP5	AIM	1.99E-59	FP5	0.7970 (L)
RDP	SPG	1.15E-41	SPG	0.2530 (L)	FP5	SPG	5.67E-46	FP5	0.7600 (L)
RDP	SPL	3.03E-11	SPL	0.3786 (M)	FP5	SPL	1.01E-46	FP5	0.7622 (L)
FP1	FP2	7.62E-40	FP2	0.2587 (L)	FP6	FPR	2.39E-42	FP6	0.7491 (L)
FP1	FP3	5.67E-46	FP3	0.2400 (L)	FP6	IIP	6.19E-08	FP6	0.5989 (S)
FP1	FP4	3.14E-46	FP4	0.2392 (L)	FP6	APE	4.90E-28	FP6	0.7005 (L)
FP1	FP5	3.06E-46	FP5	0.2392 (L)	FP6	APR	4.66E-29	FP6	0.7044 (L)
FP1	FP6	3.07E-46	FP6	0.2392 (L)	FP6	APH	2.08E-28	FP6	0.7019 (L)
FP1	FPR	2.13E-34	FPR	0.2766 (L)	FP6	TIM	5.86E-165	FP6	1.0000 (L)
FP1	IIP	4.61E-46	IIP	0.2397 (L)	FP6	AIM	3.61E-57	FP6	0.7911 (L)
FP1	APE	5.67E-46	APE	0.2400 (L)	FP6	SPG	5.67E-46	FP6	0.7600 (L)
FP1	APR	5.67E-46	APR	0.2400 (L)	FP6	SPL	1.83E-46	FP6	0.7614 (L)
FP1	APH	5.67E-46	APH	0.2400 (L)	FPR	IIP	6.53E-40	IIP	0.2585 (L)
FP1	TIM	1.26E-93	FP1	0.8749 (L)	FPR	APE	1.34E-29	APE	0.2936 (L)
FP1	AIM	5.82E-07	FP1	0.5913 (S)	FPR	APR	1.50E-29	APR	0.2938 (L)
FP1	SPG	6.60E-29	SPG	0.2962 (L)	FPR	APH	1.08E-29	APH	0.2933 (L)
FP1	SPL	0.3229	FP1	0.5181 (S)	FPR	TIM	1.61E-155	FPR	0.9853 (L)
FP2	FP3	4.83E-40	FP3	0.2581 (L)	FPR	AIM	1.45E-35	FPR	0.7274 (L)
FP2	FP4	1.56E-45	FP4	0.2413 (L)	FPR	SPG	1.39E-12	FPR	0.6294 (M)
FP2	FP5	1.63E-45	FP5	0.2413 (L)	FPR	SPL	1.10E-33	FPR	0.7210 (L)
FP2	FP6	8.94E-46	FP6	0.2406 (L)	IIP	APE	4.99E-11	IIP	0.6200 (M)
FP2	FPR	8.67E-06	FP2	0.5813 (S)	IIP	APR	3.57E-11	IIP	0.6209 (M)
FP2	IIP	9.21E-45	IIP	0.2436 (L)	IIP	APH	4.46E-11	IIP	0.6203 (M)
FP2	APE	4.11E-40	APE	0.2578 (L)	IIP	TIM	5.86E-165	IIP	1.0000 (L)
FP2	APR	3.12E-40	APR	0.2575 (L)	IIP	AIM	3.22E-51	IIP	0.7750 (L)
FP2	APH	6.14E-40	APH	0.2584 (L)	IIP	SPG	5.67E-46	IIP	0.7600 (L)
FP2	TIM	8.24E-165	FP2	0.9998 (L)	IIP	SPL	3.67E-46	IIP	0.7606 (L)
FP2	AIM	5.92E-40	FP2	0.7417 (L)	APE	APR	0.9296	APE	0.5016 (S)
FP2	SPG	1.20E-19	FP2	0.6657 (M)	APE	APH	0.9936	APH	0.4999 (S)
FP2	SPL	4.43E-38	FP2	0.7357 (L)	APE	TIM	5.86E-165	APE	1.0000 (L)
FP3	FP4	1.56E-21	FP4	0.3259 (L)	APE	AIM	9.65E-48	APE	0.7652 (L)
FP3	FP5	1.93E-27	FP5	0.3018 (L)	APE	SPG	5.67E-46	APE	0.7600 (L)
FP3	FP6	9.68E-28	FP6	0.3006 (L)	APE	SPL	5.62E-46	APE	0.7600 (L)
FP3	FPR	1.16E-29	FP3	0.7066 (L)	APR	APH	0.9368	APH	0.4986 (S)
FP3	IIP	1.74E-10	IIP	0.3834 (M)	APR	TIM	5.86E-165	APR	1.0000 (L)
FP3	APE	0.9989	FP3	0.5001 (S)	APR	AIM	9.87E-48	APR	0.7651 (L)
FP3	APR	0.9263	FP3	0.5017 (S)	APR	SPG	5.67E-46	APR	0.7600 (L)
FP3	APH	0.9816	FP3	0.5004 (S)	APR	SPL	5.69E-46	APR	0.7600 (L)
FP3	TIM	5.86E-165	FP3	1.0000 (L)	APH	TIM	5.86E-165	APH	1.0000 (L)
FP3	AIM	1.11E-47	FP3	0.7650 (L)	APH	AIM	9.90E-48	APH	0.7651 (L)
FP3	SPG	5.67E-46	FP3	0.7600 (L)	APH	SPG	5.67E-46	APH	0.7600 (L)
FP3	SPL	5.67E-46	FP3	0.7600 (L)	APH	SPL	5.65E-46	APH	0.7600 (L)
FP4	FP5	0.0845	FP5	0.4685 (S)	TIM	AIM	2.04E-72	AIM	0.1712 (L)
FP4	FP6	0.7963	FP6	0.4953 (S)	TIM	SPG	6.34E-121	SPG	0.0729 (L)
FP4	FPR	5.52E-43	FP4	0.7511 (L)	TIM	SPL	1.70E-94	SPL	0.1233 (L)
FP4	IIP	0.0046	FP4	0.5518 (S)	AIM	SPG	7.24E-33	SPG	0.2819 (L)
FP4	APE	1.45E-21	FP4	0.6742 (L)	AIM	SPL	2.38E-05	SPL	0.4228 (S)
FP4	APR	2.38E-22	FP4	0.6776 (L)	SPG	SPL	5.18E-26	SPG	0.6927 (L)

S, M, and L represents Small, Medium, and Large in effect size, respectively.

4.2 APFD Results

Figure 4 presents the APFD results for each subject program (Figures 4(a) to 4(e)), in which each plot lists the distribution of the 500 APFD values (100 orderings × 5 versions). Figure 4(f) gives the APFD results for all programs, in which each plot contains 2500 APFD values (100 orderings × 5 programs × 5 versions).

Table 5 shows the statistical APFD comparisons between two ATCP techniques based on Figure 4(f).

4.2.1 RQ1: APFD Effectiveness: ICBP: To answer the first subquestion of RQ1, regarding FICBP, we have the following observations:

• As the prioritization strength λ (1 λ 6) increases, FPλ can normally achieve higher APFD results, with a few exceptions: for example, in program grep, FP2 performs better than FP3; while FP4 performs worst for program make.

• According to mean and median APFD values, the largest differ-ence between techniques is only 4%, and the differences between high-strength FICBPs are very small. Lower-strength FICBPs are, therefore, surprisingly comparable to higher-strength ones, from the perspective of fault detection.

• As shown in Table 5, the comparisons between higher-strength (FP4, FP5, and FP6) and lower-strength (FP1, FP2, and FP3) FICBP are highly significant: except when comparing FP4 against FP3, the

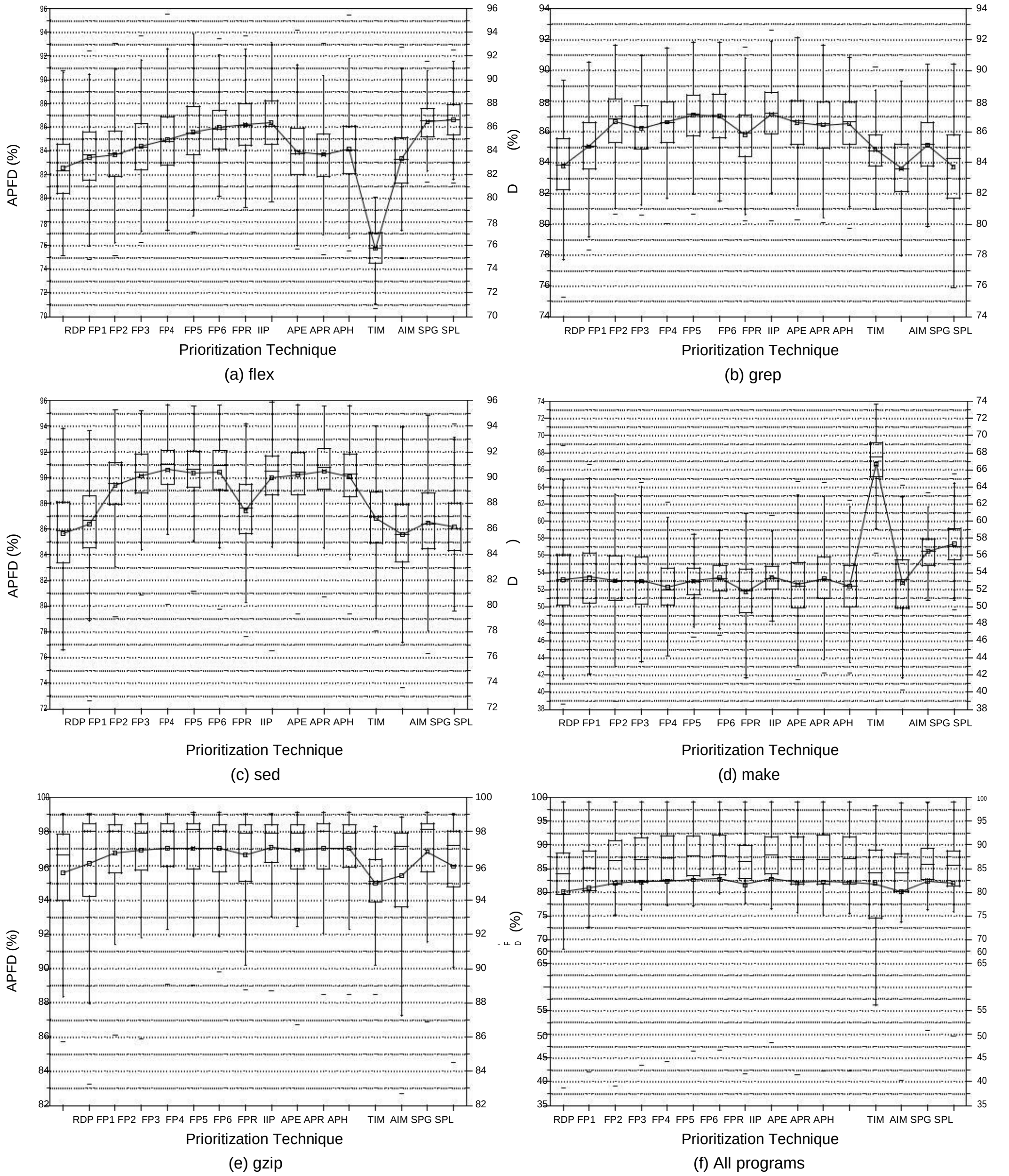


Fig. 4: APFDs for each ATCP technique for V1 to V5

p-values are less than 0.01. Among the higher-strength FICBPs, the APFD results are not significantly different (their p-values are greater than 0.01); but, among lower-strength FICBPs, the difference is highly significant, for example, when comparing FP1 with FP2 or with FP3. In terms of the effect size measure ($\hat{A}_{12}$), higher-strength FICBPs only outperform lower-strength ones between about 52% and 59% of the time. Among the higher-strength FICBPs, the $\hat{A}_{12}$ values range from about 50% to 52%; while they range from about 51% to 59% among the lower-strength FICBPs.

In answering the second subquestion, there is nearly no difference between the AICBPs, irrespective of subject program. This is also confirmed by the statistical comparison: the p-values are greater than 0.01, and the effect size measures are approximately 50%.

Regarding the third subquestion, among all eleven ICBP techniques, we have the following observations: IIP and higher-strength FICBPs generally have the highest APFD values, and IIP has better performance than lower-strength FICBPs. The second best is the AICBP series, followed by FP2, FP3, and FPR. FP1 has the worst

Table 5 Statistical APFD analysis of all pairwise comparisons (A, B)

A	B	p-value	Superior	Effect Size	A	B	p-value	Superior	Effect Size
RDP	FP1	1.27E-06	FP1	0.4604 (S)	FP4	APH	0.1268	FP4	0.5125(S)
RDP	FP2	7.79E-26	FP2	0.4142 (S)	FP4	TIM	4.53E-28	FP4	0.5897(S)
RDP	FP3	2.61E-30	FP3	0.4066 (S)	FP4	AIM	3.51E-38	FP4	0.6055 (M)
RDP	FP4	9.97E-38	FP4	0.3952 (M)	FP4	SPG	5.96E-05	FP4	0.5328(S)
RDP	FP5	1.03E-46	FP5	0.3828 (M)	FP4	SPL	3.31E-10	FP4	0.5513(S)
RDP	FP6	4.56E-51	FP6	0.3773 (M)	FP5	FP6	0.5841	FP6	0.4955(S)
RDP	FPR	4.40E-26	FPR	0.4137 (S)	FP5	FPR	3.20E-09	FP5	0.5483(S)
RDP	IIP	4.86E-54	IIP	0.3736 (M)	FP5	IIP	0.4432	FP5	0.4937 (M)
RDP	APE	1.38E-28	APE	0.4094 (S)	FP5	APE	0.0022	FP5	0.5250(S)
RDP	APR	2.14E-29	APR	0.4081 (S)	FP5	APR	0.0128	FP5	0.5203(S)
RDP	APH	2.29E-29	APH	0.4081 (S)	FP5	APH	0.0037	FP5	0.5237(S)
RDP	TIM	0.7419	RDP	0.5027 (S)	FP5	TIM	5.76E-33	FP5	0.5977(S)
RDP	AIM	0.6734	AIM	0.4966 (S)	FP5	AIM	6.95E-48	FP5	0.6187 (M)
RDP	SPG	8.30E-28	SPG	0.4107 (S)	FP5	SPG	2.14E-08	FP5	0.5457(S)
RDP	SPL	1.05E-16	SPL	0.4322 (S)	FP5	SPL	9.25E-15	FP5	0.5633(S)
FP1	FP2	7.49E-11	FP2	0.4468 (S)	FP6	FPR	1.23E-10	FP6	0.5526(S)
FP1	FP3	9.29E-14	FP3	0.4392 (S)	FP6	IIP	0.8092	FP6	0.4980(S)
FP1	FP4	1.37E-19	FP4	0.4261 (S)	FP6	APE	0.0004	FP6	0.5292(S)
FP1	FP5	5.83E-27	FP5	0.4122 (S)	FP6	APR	0.0025	FP6	0.5247(S)
FP1	FP6	1.17E-29	FP6	0.4076 (S)	FP6	APH	0.0006	FP6	0.5279(S)
FP1	FPR	1.39E-09	FPR	0.4505 (S)	FP6	TIM	6.45E-35	FP6	0.6007 (M)
FP1	IIP	3.74E-32	IIP	0.4036 (S)	FP6	AIM	2.06E-52	FP6	0.6244 (M)
FP1	APE	5.81E-13	APE	0.4412 (S)	FP6	SPG	1.67E-09	FP6	0.5492(S)
FP1	APR	1.21E-13	APR	0.4394 (S)	FP6	SPL	2.03E-16	FP6	0.5671(S)
FP1	APH	1.33E-13	APH	0.4395 (S)	FPR	IIP	4.50E-12	IIP	0.4435(S)
FP1	TIM	0.0002	FP1	0.5300 (S)	FPR	APE	0.0255	APE	0.4818(S)
FP1	AIM	2.16E-06	FP1	0.5387 (S)	FPR	APR	0.0094	APR	0.4788(S)
FP1	SPG	2.69E-09	SPG	0.4514 (S)	FPR	APH	0.0142	APH	0.4800(S)
FP1	SPL	0.0007	FP1	0.4722 (S)	FPR	TIM	1.06E-17	FPR	0.5700(S)
FP2	FP3	0.3028	FP3	0.4916 (S)	FPR	AIM	1.26E-26	FPR	0.5872(S)
FP2	FP4	0.0080	FP4	0.4783 (S)	FPR	SPG	0.6340	FPR	0.5039 (M)
FP2	FP5	5.18E-05	FP5	0.4669 (S)	FPR	SPL	0.0036	FPR	0.5237(S)
FP2	FP6	5.24E-06	FP6	0.4628 (S)	IIP	APE	0.0001	IIP	0.5314 (M)
FP2	FPR	0.1867	FP2	0.5108 (S)	IIP	APR	0.0012	IIP	0.5265 (M)
FP2	IIP	1.32E-06	IIP	0.4605 (S)	IIP	APH	0.0002	IIP	0.5303 (M)
FP2	APE	0.3601	APE	0.4925 (S)	IIP	TIM	4.73E-37	IIP	0.6038 (M)
FP2	APR	0.1785	APR	0.4890 (S)	IIP	AIM	2.34E-55	IIP	0.6280 (M)
FP2	APH	0.2857	APH	0.4913 (S)	IIP	SPG	2.69E-11	IIP	0.5544(S)
FP2	TIM	3.34E-18	FP2	0.5710 (S)	IIP	SPL	9.54E-19	IIP	0.5722(S)
FP2	AIM	7.26E-26	FP2	0.5859 (S)	APE	APR	0.6566	APE	0.4964(S)
FP2	SPG	0.1543	FP2	0.5116 (M)	APE	APH	0.8601	APH	0.4986(S)
FP2	SPL	0.0002	FP2	0.5305 (S)	APE	TIM	1.41E-21	APE	0.5779(S)
FP3	FP4	0.1071	FP4	0.4868 (S)	APE	AIM	1.26E-28	APE	0.5906(S)
FP3	FP5	0.0030	FP5	0.4757 (S)	APE	SPG	0.0283	APE	0.5179(S)
FP3	FP6	0.0005	FP6	0.4714 (S)	APE	SPL	8.04E-06	APE	0.5365(S)
FP3	FPR	0.0196	FP3	0.5191 (S)	APR	APH	0.7745	APH	0.5023(S)
FP3	IIP	0.0002	IIP	0.4693 (S)	APR	TIM	3.20E-21	APR	0.5772(S)
FP3	APE	0.9032	FP3	0.5010 (S)	APR	AIM	1.94E-29	APR	0.5920(S)
FP3	APR	0.7688	FP3	0.4976 (S)	APR	SPG	0.0210	APR	0.5188(S)
FP3	APH	0.9618	FP3	0.4996 (S)	APR	SPL	5.76E-06	APR	0.5370(S)
FP3	TIM	1.41E-21	FP3	0.5779 (S)	APH	TIM	2.62E-22	APH	0.5793(S)
FP3	AIM	1.47E-30	FP3	0.5938 (S)	APH	AIM	1.83E-29	APH	0.5920(S)
FP3	SPG	0.0202	FP3	0.5190 (S)	APH	SPG	0.0200	APH	0.5190(S)
FP3	SPL	4.91E-06	FP3	0.5373 (S)	APH	SPL	4.60E-06	APH	0.5374(S)
FP4	FP5	0.1792	FP5	0.4890 (S)	TIM	AIM	0.6916	AIM	0.4968(S)
FP4	FP6	0.0595	FP6	0.4846 (S)	TIM	SPG	2.04E-13	SPG	0.4400(S)
FP4	FPR	2.71E-05	FP4	0.5343 (S)	TIM	SPL	2.14E-08	SPL	0.4543(S)
FP4	IIP	0.0344	FP4	0.4827 (S)	AIM	SPG	1.03E-28	SPG	0.4092(S)
FP4	APE	0.0880	FP4	0.5139 (S)	AIM	SPL	7.39E-17	SPL	0.4319(S)
FP4	APR	0.2311	FP4	0.5098 (S)	SPG	SPL	0.0266	SPG	0.5181(S)

performance. It is surprising that FPR has APFD results comparable to FP2 and FP3, and has higher APFD scores than FP1, because it only repeats 1-wise interaction coverage. Overall, the statistical analysis (see Table 5) supports the box plot observations, with a degree of variation in the performance of different ICBP techniques for different programs. Nevertheless, based on the programs we have studied, our results suggest that IIP and higher-strength FICBPs offer the best rates of fault detection among the ICBP techniques.

4.2.2 RQ2: APFD Effectiveness: IMBP: For subject programs flex and gzip, AIM performs significantly better than TIM, with respect to both the mean and median APFD values. However, for the other three programs (grep, sed, and make), TIM achieves much better APFD results (again from the perspective of both mean and median APFD values). This is especially so for the program make, where the mean APFD for TIM is close to 67%, but for AIM it is only about 53%; the median APFD for TIM is 67.5%, but the AIM median is also only about 53%. In contrast to previous TCP studies [1, 31], an interesting result is that the 'additional' TCP techniques do not guarantee to provide better fault detection rates than the 'total' TCP techniques.

However, the statistical analysis for all five programs suggests that the differences in performance between TIM and AIM are not significant: their p-values are much greater than 0.01, and the effect sizes are approximately 50%. In other words, TIM and AIM have comparable fault detection rates.

4.2.3 RQ3: APFD Effectiveness: SBP:

For programs flex and make, SPG performs slightly better than SPL, however for the other programs (grep, sed, and gzip) SPG is better than SPL, with respect to both the mean and the median APFD values. Considering all programs (Figure 4(f)), overall, SPG is slightly better than SPL, but the differences between them are less than 1%. Similarly, the statistical comparison gives a p-value of 0.0266, and an effect size of 0.5181, which indicates that the difference is not significant.

4.2.4 RQ4: APFD Effectiveness of All Techniques:

Although different techniques have different APFD performances for different programs, we can nonetheless observe the following:

- For program flex, SPG and SPL are the two best techniques, followed by IIP and FPR, in terms of both the median and the mean APFD values — although the differences are very small (less than 1%). Additionally, and surprisingly, TIM has the worst performance — even worse than RDP, which uses no additional information in the prioritization process.
- For program make, TIM is significantly better than all other ATCP techniques, followed by SPG and SPL. Additionally, RDP, FPλ (1 λ 6; except λ = 4), IIP, APR, and AIM, all have similar APFD performance. FP4 and FPR are the two worst techniques.
- For the three programs grep, sed, and gzip, the FICBPs (except FP1) and other ICBP techniques perform best, with FP1, SPG, SPL, TIM, and AIM able to achieve comparable APFD results. Furthermore, it is again surprising that RDP could sometimes have similar fault detection rates to TIM, AIM, and SPL.
- When all programs are considered together, overall, the ICBP series has the best performance, followed by the SBP series; NIGP and IMBP perform worst, with similar fault detection rates. Regarding individual ATCP techniques, the ICBP series is best, as discussed in the first subquestion of RQ1, with IIP and higher-strength FICBPs performing best among all techniques. SPG and SPL are better than AIM, TIM, and FP1; with SPG achieving comparable APFD performance to FP2, FP3, FPR, and the AICBP series.

Taking into consideration both APCC and APFD results, we can conclude that higher-strength FICBPs (FP4, FP5, and FP6) achieve the best rates of both interaction coverage and fault detection, followed by IIP. Although SPG has lower APCC results than the ICBP techniques (as discussed before, because ICBP uses interaction coverage to guide the prioritization), it can achieve higher APFD scores than FP1, and has performance comparable to FP2, FP3, FPR, and AICBP. Additionally, IMBP techniques generally perform similarly, or worse, compared with random test case prioritization.

4.3 Prioritization Time Results

To address RQ5, Table 6 presents the mean prioritization time for each ATCP technique for each subject program — it should be noted that, because we used the model mutation matrix file from previous studies [15], TIM and AIM do not include the model mutation time. Based on the experimental data, and as expected, it is clear that RDP needs the least prioritization time among all ATCP techniques, followed by TIM, FP1, and AIM. The next best performance, in terms of prioritization time, is by FPR, SPG, and SPL (all of which require slightly more time than the four best techniques). FP5 has the slowest prioritization time, followed by FP6, FP4, and IIP; and the AICBP series has similar times to FP3.

Based on the effectiveness and efficiency experiments, our recommendations and guidelines are as follows: given sufficient resources (including time) for prioritizing ATCs, FICBPλ at higher strength λ values (λ = 4, 5, 6) should be the best choice, followed by IIP. However, if time resources are limited, then FPR and SPG would be the best choices, followed by FP2, FP3, and the AICBP series; FP1 and RDP could also be alternatives, when facing very severe time constraints. As discussed in Section 2.2.5, we believe that our experimental results are basically consistent with the expected strengths and weaknesses of each ATCP technique.

Table 6 Prioritization time (in seconds) for each ATCP technique

ATCP Technique	Subject Program flex	grep	sed	make	gzip	Sum
RDP	0.05	0.01	0.04	0.01	0.01	0.12
FP1	0.28	0.47	0.41	0.06	0.16	1.38
FP2	4.37	8.99	10.35	0.42	2.35	26.48
FP3	10.68	26.66	36.42	2.18	27.70	103.64
FP4	52.74	81.19	144.54	6.70	115.86	401.03
FP5	84.91	198.93	339.66	18.72	326.63	968.85
FP6	59.08	108.67	217.38	16.87	518.15	920.15
FPR	2.96	2.23	1.46	0.50	1.36	8.51
IIP	54.14	40.63	41.34	16.56	168.70	321.37
APE	12.32	29.51	40.05	3.94	43.94	129.76
APR	12.92	30.12	40.88	4.07	42.61	130.60
APH	12.94	29.99	40.08	4.13	43.20	130.34
TIM*	0.38	0.79	0.08	0.05	0.03	1.33
AIM*	1.36	1.89	0.13	0.10	0.03	3.51
SPG	3.78	3.23	1.73	0.19	0.49	9.42
SPL	3.84	2.02	1.50	0.17	0.28	7.81

"*" indicates that model mutation time is not included.

4.4 Threats to Validity

In this section, we list some potential threats to validity, including external validity, internal validity, construct validity, and conclusion validity.

4.4.1 External Validity: With respect to the external validity, the main threat is the generalizability of our results. Although we have used only five subject programs, written in C, all of which are of a relatively medium size, we believe that by including six versions of each (giving 30 subject versions under study), that there is suffi-cient data from which to draw the conclusions. Nevertheless, more larger subject programs, written in other languages should also be examined in future work.

Another potential threat to external validity is the representativeness of ATCs for each subject program. In this paper, we focused on ATCs originated from the SIR [40] (using the test specification language to create the input model and construct ATCs [16]), which is only one type of ATC encoding. However, there exist other ATC encoding types [52], which we will investigate in our future work.

4.4.2 Internal Validity: The threat to internal validity relates mainly 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.

4.4.3 Construct Validity: In this study, we have focused on the testing effectiveness and efficiency, measured by the rate of interac-tion coverage, the rate of fault detection, and the prioritization time. Although the APCC and APFD metrics have often been used in the field of test case prioritization [1, 21, 24, 34], we acknowledge that there may be other metrics which may also be relevant.

4.4.4 Conclusion Validity: As for the conclusion validity, the main threat is the randomized computation of our algorithms. To minimize this threat, all algorithms were repeated 100 times, and inferential statistics were applied to the comparisons of results.

5 Conclusions and Future Work

This paper has reported on a comparison of 16 ATCP techniques, classified into four categories, based on an extensive empirical study. Based on comparisons of testing effectiveness and efficiency, some recommendations and guidelines have also been given, to help testers choose among ATCP techniques under different testing situations and scenarios.

The main findings of this study can be summarized as:

(1) With respect to all ATCP categories, the ICBP category has the best testing effectiveness, irrespective of the rates of interaction cov-erage and fault detection. Somewhat surprisingly, because it does not use any additional information to guide the prioritization, NIGP

could achieve comparable performance to IMBP; while SBP has very good testing effectiveness, and even better than some ICBP techniques sometimes. Additionally, IMBP has the worst rates of interaction coverage, but it sometimes has the best fault detection rates. Nevertheless, NIGP, IMBP, SBP, and some ICBP techniques have better testing efficiency than others.

(2) In the category of ICBP techniques, it is evident that higher-strength FICBP techniques, and IICBP have the best testing effectiveness (according to interaction coverage and fault detection), followed by AICBP and lower-strength FICBP techniques. However, higher-strength FICBP and IICBP techniques are less efficient than other ICBP techniques, according to the prioritization time.

(3) Regarding the IMBP techniques, although both ‘total’ and ‘additional’ IMBP techniques have similar prioritization times, they have different performances according to the other evaluation measures. For example, the ‘additional’ IMBP has better rates of interaction coverage than the ‘total’ IMBP, regardless of subject programs. However, for three programs, the ‘additional’ IMBP has better fault detection than the ‘total’ IMBP, but for another two cases, this is reversed: the ‘total’ IMBP can obtain better fault detection.

(4) For the SBP techniques, the global SBP has better rates of interaction coverage than the local SBP. However, they have similar fault detection rates and prioritization costs: the global SBP is slightly better than the local one for some programs, but the opposite is the case for some other programs.

(5) When testers select only some ATCP techniques for prioritizing abstract test cases, we recommend that, given sufficient resources and prioritization time, FICBPλ at higher strength λ values (i.e., λ = 4, 5, 6) should be the best choice, followed by IICBP. However, if facing limited time resources, then GSBP may be the best choice, followed by FICBP2, FICBP3, and AICBP; FICBP1 and NIGP may be alternatives in situations with very severe time constraints.

As discussed before, IMBP uses the model mutation information to prioritize ATCs, so the quality of IMBP is mainly dependent on the model mutation, which may be a reason for the ineffectiveness of IMBP in this study. It will therefore be very interesting to investigate the correlation between model mutation and program mutation in our future work. In addition, since this study adopted mutation analysis [39] to investigate testing effectiveness of ATCP techniques, more experiments with real faults should be conducted to validate our conclusions. Last but not the least, in this paper we only considered the prioritization time as the resource factor for guiding the selection of ATCP techniques. However, there are many other resource factors such as the execution time of test cases. Therefore, it would be interesting to combine more testing requirements for designing more comprehensive guidelines to select ATCP techniques.

Acknowledgements

We would like to thank Christopher Henard for sharing us the fault data materials of each subject program. This work is supported by the National Natural Science Foundation of China (Grant Nos. 61202110, 61502205, and 61872167), and the Senior Personnel Scientific Research Foundation of Jiangsu University (Grant No. 14JDG039). This work is also supported by the Young Backbone Teacher Cultivation Project of Jiangsu University, and the sponsorship of Jiangsu Overseas Visiting Scholar Program for University Prominent Young & Middle-aged Teachers and Presidents. A preliminary version of this paper was presented at the 39th International Conference on Software Engineering (ICSE’17) [53].

6 References

1 Rothermel, G., Untch, R.H., Chu, C., Harrold, M.J.: ‘Prioritizing test cases for regression testing’, IEEE Transactions on Software Engineering, 2001, 27, (10), pp. 929–948

2 Yoo, S., Harman, M.: ‘Regression testing minimization, selection and prioritization: A survey’, Software Testing, Verification and Reliability, 2012, 22, (2), pp. 67–120

3 Di.Nardo, D., Alshahwan, N., Briand, L., Labiche, Y.: ‘Coverage-based regression test case selection, minimization and prioritization: a case study on an industrial system’, Software Testing, Verification and Reliability, 2015, 25, (4), pp. 371–396

4 Li, Z., Harman, M., Hierons, R.M.: ‘Search algorithms for regression test case prioritization’, IEEE Transactions on Software Engineering, 2007, 33, (4), pp. 225–237

5 Parejo, J.A., Sánchez, A.B., Segura, S., Ruiz.Cortés, A., Lopez.Herrejon, R.E., Egyed, A.: ‘Multi-objective test case prioritization in highly configurable systems: A case study’, Journal of Systems and Software, 2016, 122, pp. 287 – 310

6 Chen, J., Zhu, L., Chen, T.Y., Towey, D., Kuo, F.C., Huang, R., et al.: ‘Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering’, Journal of Systems and Software, 2018, 135, pp. 107 – 125

7 Jiang, B., Zhang, Z., Chan, W.K., Tse, T.H. ‘Adaptive random test case prioritization’. In: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE’09). (, 2009. pp. 233–244

8 Zhang, X., Chen, T.Y., Liu, H. ‘An application of adaptive random sequence in test case prioritization’. In: Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE’14). (, 2014. pp. 126–131

9 Zhang, X., Xie, X., Chen, T.Y. ‘Test case prioritization using adaptive random sequence with category-partition-based distance’. In: Proceedings of the 16th IEEE International Conference on Software Quality, Reliability and Security (QRS’16). (, 2016. pp. 374–385

10 Fang, C., Chen, Z., Wu, K., Zhao, Z.: ‘Similarity-based test case prioritization using ordered sequences of program entities’, Software Quality Journal, 2014, 22, (2), pp. 335–361

11 Noor, T.B., Hemmati, H. ‘A similarity-based approach for test case prioritization using historical failure data’. In: Proceedings of the 26th International Symposium on Software Reliability Engineering (ISSRE’15). (, 2015. pp. 58–68

12 Catal, C., Mishra, D.: ‘Test case prioritization: a systematic mapping study’, Software Quality Journal, 2013, 21, (3), pp. 445–478

13 Khatibsyarbini, M., Isa, M.A., Jawawi, D.N.A., Tumeng, R.: ‘Test case prioritization approaches in regression testing: A systematic literature review’, Information and Software Technology, 2018, 93, pp. 74 – 93

14 Grindal, M., Lindström, B., Offutt, J., Andler, S.F.: ‘An evaluation of combination strategies for test case selection’, Empirical Software Engineering, 2006, 11, (4), pp. 583–611

15 Henard, C., Papadakis, M., Harman, M., Jia, Y., Traon, Y.L. ‘Comparing white-box and black-box test prioritization’. In: Proceedings of the 38th International Conference on Software Engineering (ICSE’16). (, 2016. pp. 523–534

16 Ostrand, T.J., Balcer, M.J.: ‘The category-partition method for specifying and generating fuctional tests’, Communications of the ACM, 1988, 31, (6), pp. 676–686

17 Nie, C., Leung, H.: ‘A survey of combinatorial testing’, ACM Computer Survey, 2011, 43, (2), pp. 11:1–11:29

18 Utting, M., Legeard, B.: ‘Practical model-based testing - a tools approach.’, International Journal On Advances in Software, 2007, 2, (1), pp. 1–419

19 Bryce, R.C., Colbourn, C.J.: ‘Prioritized interaction testing for pairwise coverage with seeding and contraints’, Information and Software Technology, 2006, 48, (10), pp. 960–970

20 Huang, R., Chen, J., Towey, D., Chan, A.T.S., Lu, Y.: ‘Aggregate-strength interaction test suite prioritization’, Journal of Systems and Software, 2015, 99, pp. 36–51

21 Petke, J., Cohen, M.B., Harman, M., Yoo, S.: ‘Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection’, IEEE Transactions on Software Engineering, 2015, 41, (9), pp. 901–924

22 Al.Hajjaji, M., Thüm, T., Meinicke, J., Lochau, M., Saake, G. ‘Similarity-based prioritization in software product-line testing’. In: Proceedings of 18th International Software Product Line Conference (SPLC’14). (, 2014. pp. 197–206

23 Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., Traon, Y.L.: ‘Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines’, IEEE Transactions on Software Engineering, 2014, 40, (7), pp. 650–670

24 Petke, J., Cohen, M.B., Harman, M., Yoo, S. ‘Efficiency and early fault detection with lower and higher strength combinatorial interaction testing’. In: Proceedings of the 12th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE’13). (, 2013. pp. 26–36

25 Papadakis, M., Henard, C., Traon, Y.L. ‘Sampling program inputs with mutation analyais: Going beyond combinatorial interaction testing’. In: Proceedings of the 7th International Conference on Software Testing, Verification and Validation (ICST’14). (, 2014. pp. 1–10

26 Zhang, Z., Zhang, J. ‘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

27 Cohen, M.B., Dwyer, M.B., Shi, J.: ‘Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach’, IEEE Transactions on Software Engineering, 2008, 34, (5), pp. 633–650

28 Yilmaz, C., Dumlu, E., Cohen, M.B., Porter, A.A.: ‘Reducing masking effects in combinatorial interaction testing: A feedback driven adaptive approach’, IEEE Transactions on Software Engineering, 2014, 40, (1), pp. 43–66

29 Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: ‘A classification and survey of analysis strategies for software product lines’, ACM Computing Survery, 2014, 47, (1), pp. 6:1–6:45

30 Barus, A.C., Chen, T.Y., Kuo, F.C., Liu, H., Merkel, R., Rothermel, G.: ‘A cost - effective random testing method for programs with non-numeric inputs’, IEEE Transactions on Computers, 2016, 65, (12), pp. 3509–3523

31 Zhang, L., Hao, D., Zhang, L., Rothermel, G., Mei, H. ‘Bridging the gap between the total and additional test-case prioritization strategies’. In: Proceedings of the 35th International Conference on Software Engineering (ICSE’13). (, 2013. pp. 192–201

32 Bryce, R.C., Memon, A.M. ‘Test suite prioritization by interaction coverage’. In: Proceedings of the Workshop on Domain Specific Approaches to Software Test Automation (DoSTA’07). (, 2007. pp. 1–7

33 Bryce, R.C., Sampath, S., Memon, A.M.: ‘Developing a single model and test prioritization strategies for event-driven software’, IEEE Transactions on Software

Engineering, 2011, 37, (1), pp. 48–64

34 Qu, X., Cohen, M.B., Woolf, K.M. ‘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

35 Qu, X., Cohen, M.B., Woolf, K.M. ‘A study in prioritization for higher strength combinatorial testing’. In: Proceedings of the 2nd International Workshop on Combinatorial Testing, (IWCT'13). (, 2013. pp. 285–294

36 Huang, R., Zong, W., Chen, J., Towey, D., Zhou, Y., Chen, D. ‘Prioritizing interaction test suite using repeated base choice coverage’. In: Proceedings of the IEEE 40th Annual Computer Software and Applications Conference (COMPSAC'16). (, 2016. pp. 174–184

37 Huang, R., Chen, J., Zhang, T., Wang, R., Lu, Y. ‘Prioritizing variable-strength covering array’. In: Proceedings of the IEEE 37th Annual Computer Software and Applications Conference (COMPSAC'13). (, 2013. pp. 502–601

38 Huang, R., Xie, X., Towey, D., Chen, T.Y., Lu, Y., Chen, J.: ‘Prioritization of combinatorial test cases by incremental interaction coverage’, International Journal of Software Engineering and Knowledge Engineering, 2013, 23, (10), pp. 1427–1457

39 Jia, Y., Harman, M.: ‘An analysis and survey of the development of mutation testing’, IEEE Transactions on Software Engineering, 2011, 37, (5), pp. 649–678

40 Do, H., Elbaum, S.G., Rothermel, G.: ‘Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact’, Empirical Software Engineering, 2005, 10, (4), pp. 405–435

41 Andrews, J.H., Briand, L.C., Labiche, Y., Namin, A.S.: ‘Using mutation analysis for assessing and comparing testing coverage criteria’, IEEE Transactions on Software Engineering, 2006, 32, (8), pp. 608–624

42 Do, H., Rothermel, G.: ‘On the use of mutation faults in empirical assessments of test case prioritization techniques’, IEEE Transactions on Software Engineering, 2006, 32, (9), pp. 733–752

43 Jia, Y., Harman, M.: ‘Higher order mutation testing’, Information and Software Technology, 2009, 51, (10), pp. 1379 – 1393

44 Ammann, P., Delamaro, M.E., Offutt, J. ‘Establishing theoretical minimal sets of mutants’. In: Proceedings of the 7th International Conference on Software Testing, Verification and Validation (ICST'14). (, 2014. pp. 21–30

45 Kintis, M., Papadakis, M., Malevris, N. ‘Evaluating mutation testing alternatives: A collateral experiment’. In: Proceedings of the 17th Asia-Pacific Software Engineering Conference (APSEC'10). (, 2010. pp. 300–309

46 Papadakis, M., Henard, C., Harman, M., Jia, Y., Le.Traon, Y. ‘Threats to the validity of mutation-based test assessment’. In: Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA'16). (, 2016. pp. 354—365

47 Huang, R., Chen, J., Chen, D., Wang, R. ‘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

48 Wang, Z., Chen, L., Xu, B., Huang, Y.: ‘Cost-cognizant combinatorial test case prioritization’, International Journal of Software Engineering and Knowledge Engineering, 2011, 21, (6), pp. 829–854

49 Arcuri, A., Briand, L.: ‘A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering’, Software Testing, Verification and Reliability, 2014, 24, (3), pp. 219–250

50 Harman, M., McMinn, P., Souza, J., Yoo, S.: ‘Search based software engineering: Techniques, taxonomy, tutorial’, Empirical Software Engineering and Verification, 2012, pp. 1–59

51 Vargha, A., Delaney, H.D.: ‘A critique and improvme of the cl common language effect size statistics of mcgraw and wong’, Journal of Education and Behavioral Statistics, 2000, 25, (2), pp. 101–132

52 Hemmati, H., Arcuri, A., Briand, L.: ‘Achieving scalable model-based testing through test case diversity’, ACM Transactions on Software Engineering and Methodology, 2013, 22, (1), pp. 139–176

53 Huang, R., Zong, W., Towey, D., Zhou, Y., Chen, J. ‘An empirical examination of abstract test case prioritization techniques’. In: Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C'17). (, 2017. pp. 141–143