HKU CS Tech Report TR-2014-05

Extended version of article in *Proceedings of the 21st IEEE International Conference on Web Services* (*ICWS '14*), IEEE Computer Society, 2014

Is XML-based Test Case Prioritization for Validating WS-BPEL Evolution Effective in both Average and Adverse Scenarios?

Changjiang Jia
City University of Hong Kong
Kowloon Tong, Hong Kong
cjjia.cs@gmail.com

Lijun Mei†
IBM Research – China
Beijing, China
meilijun@cn.ibm.com

W.K. Chan, Y.T. Yu
City University of Hong Kong
Kowloon Tong, Hong Kong
{wkchan,csytyu}@cityu.edu.hk

T.H. Tse
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

***Abstract*—In real life, a tester can only afford to apply *one* test case prioritization technique to *one* test suite against a service-oriented workflow application *once* in the regression testing of the application, even if it results in an *adverse scenario* such that the actual performance in the test session is far below the *average*. It is unclear whether the factors of test case prioritization techniques known to be significant in terms of average performance can be extrapolated to adverse scenarios. In this paper, we examine whether such a factor or technique may consistently affect the rate of fault detection in both the average and adverse scenarios. The factors studied include prioritization strategy, artifacts to provide coverage data, ordering direction of a strategy, and the use of executable and non-executable artifacts. The results show that only a minor portion of the 10 studied techniques, most of which are based on the iterative strategy, are consistently effective in both average and adverse scenarios. To the best of our knowledge, this paper presents the first piece of empirical evidence regarding the consistency in the effectiveness of test case prioritization techniques and factors of service-oriented workflow applications between average and adverse scenarios.**

***Keywords*—XML-based factor; WS-BPEL; adaptation; adverse**

I. Introduction

A service-based workflow program such as a WS-BPEL application [30] is a service that, at runtime, binds each of its workflow steps to another service. A fault in such a service will affect the correctness of the service itself as well as each service that binds to it. Any modification of a workflow service should be thoroughly tested to reduce the potential impact of any fault to its consumers. To stay competitive, the service should be continuously maintained in order to adapt to any changing business requirements, or else a service consumer will bind to a competing service if the consumer's requirements are not made available in time. In short, from the verification viewpoint, workflow services demand highly efficient test sessions.

Regression testing is a widely used industrial practice [21]. Developers may execute the modified service over the test cases in a regression test suite to assess whether this service executes normally and computes outputs as specified [17]. It should be conducted on any modified version of a service before the new version is deployed.

Owing to the need for thorough testing, the number of test case invocations with respect to the service under test may be very large. At the same time, native executions of workflow service (including the time to bind and invoke external services [16]) may be long. As a result, an entire test session may become too time-consuming, which is in conflict with the demand for efficient test sessions. Prioritization of the order of execution of test cases has the potential to alleviate this inefficient test session problem by revealing faults earlier, thereby starting the service repair earlier and shortening the maintenance period. Many research studies (such as [11][18][21][25][33]) have reported that test case prioritization is an important aspect of practical regression testing.

Although existing studies have shown that the fault detection rate of some test case prioritization techniques can be excellent, different strategies may exhibit subtle differences in their tradeoff [4][9]. For instance, comparing the results of the same technique on different languages (such as Java [3] versus C [4]) has revealed that such a technique may exhibit varying extents of effectiveness. Some prioritization techniques are also found to be multimodal, that is, their effectiveness has multiple peak regions [11].

Each test run of a WS-BPEL program may indicate the execution of a specific workflow path with reference to certain XML tags in the WSDL documents [31] of the program. Thus, the same execution of a service has the potential to generate coverage data based on different types of artifacts. Mei et al. [18] proposed to integrate progressively the coverage data from multiple artifacts by using a level-exploration strategy with a greedy-based test case prioritization strategy at each level of exploration [13]. Moreover, XML messages sent and received by a service along the execution trace of each test case can be used as

† All enquiries should be addressed to Lijun Mei.

runtime artifacts to provide a new dimension of data sources for prioritizing test cases [17]. Mei et al. [13][14] found that using more types of coverage data systematically can also make similarity-based test case prioritization techniques more effective.

In practice, a developer only applies *one* test case prioritization technique to *one* test suite *once*. The developer does not have the luxury to apply multiple test suites to look for the *average* (that is, mean or median) performance of the technique on the same service. Thus, even when the average performance of a technique (or a factor across multiple techniques) is excellent, if the technique (or factor) performs very ineffectively in scenarios that are far below average (hereafter simply referred to as *adverse scenarios*), the technique (or factor) may not be reliably used in practice.

To the best of our knowledge, existing test case prioritization techniques (including the above-stated work) exhaustively focus on comparing and analyzing the average effectiveness among multiple techniques (in terms of the fault detection rate such as APFD [26]). From our discussions above, we would like to see that they also perform satisfactorily in adverse scenarios, but this remains a conjecture, as formulated as follows, to be validated empirically.

> **Conjecture 1**: Suppose that a *factor* of a test case prioritization technique has been shown to exhibit significant effectiveness in the average scenarios in terms of fault detection rate. Then the factor will also be a significant factor in the adverse scenarios.

This conjecture has significant implications because handling adverse scenarios can significantly improve the confidence in accepting such techniques in practice, and there is a large body of research results on the average performance of various test case prioritization techniques. If the conjecture can be (largely) established, we may extrapolate existing research results to adverse scenarios. If the conjecture cannot be established, then there is an obvious gap between previous analyses of test case prioritization techniques and the above practical consideration, which urges for more effort to be filled. However, to the best of our knowledge, there is as yet no evidence about the validity of the conjecture.

To examine Conjecture 1, we empirically study a suite of factors in this paper: (1) *prioritization strategy*, (2) *type of artifacts that generate coverage data for prioritization usages*, (3) *ordering direction* of a prioritization strategy (whether *ascending* or *descending*), and (4) the *nature of the artifacts* that produces the coverage data (whether the data are obtained from *executable* or *non-executable* artifacts). For instance, these four factors have been studied in previous experiments [4][13][17][18] in the standard (that is, "average") scenarios.

In the experiment, we attempt to observe the effects of these factors in both the average and adverse scenarios, the latter being quantified as the lowest 25th percentile of the *Average Percentage of Faults Detected* (*APFD*) [26] achieved by a test case prioritization technique on a suite of subjects. To support our experiment, we formulate four new XRG-based techniques (M5 to M8 in Section III) to bridge the gap between existing strategies, namely, the additional/ total greedy strategy [4] and the iterative strategy [13]. The results show that only a minor portion of techniques (4 out of 10 techniques studied) are effective in both average and adverse scenarios. Moreover, only one of the four factors (namely, the iterative strategy in the strategy factor) can be largely consistent in effectiveness in both types of scenarios. Our results provide the first piece of evidence data to show that, among the four factors and their choices studied, only one single choice (the iterative strategy) out of one single factor (the *strategy* factor) supports Conjecture 1.

The main contribution of this paper is twofold. (i) To the best of our knowledge, this paper presents the *first* analysis of adverse scenarios, in which it examines a suite of techniques and four factors. Moreover, it shows that, for only 4 out of 10 studied techniques, the effectiveness in average scenarios can be extrapolated to adverse scenarios. It also identifies the iterative strategy in the *strategy* factor as the only choice that supports Conjecture 1. (ii) We propose four new XRG-based techniques that supplement existing ones.

The rest of the paper is organized as follows: Section II describes the background of the test case prioritization problem and related work. Section III reviews the test case prioritization techniques under study. Section IV presents our empirical study. Finally, Section V concludes the paper.

II. RELATED WORK

This section reviews related work and revisits the terminology used in test case prioritization.

Regression testing is widely used in the industry [21]. It is a testing process performed after the modification of a program [10]. Leung and White [10] pointed out that it is not a simple testing process by just rerunning all the test cases. Regression testing can be more effective by selecting only those test cases relevant to the modified components. Test case prioritization is one of major tasks in regression testing, enabling test cases to be executed in selected order to achieve specific testing purposes, such as a higher fault detection rate [26].

The test case prioritization problem has been formally defined in [26], which we adapt as follows:

Given: T, a test suite; PT, a set of permutations of T; and f, a function from PT to real numbers.

Objective: To find a reordered test suite $T' \in PT$ such that $\forall T'' \in PT, f(T') \geq f(T'')$.

Leung and White [10] provided a principle of retests by dividing the regression testing problem into two sub-problems: *test selection* and *test plan update*. Rothermel and Harrold [25] surveyed earlier families of techniques for regression test selection (such as symbolic execution techniques, path analysis techniques, dataflow techniques [22], and modification-based techniques). More recently, Yoo and Harman [37] reported that there are an increasing number of papers that study regression testing techniques.

Generally, there are two kinds of test case prioritization, namely *general* test case prioritization and *version-specific* test case prioritization [26]. For the former, a test suite *T* for a program *P* is sorted with the intent of being useful over the subsequent modified versions of *P*. For the latter, the test suite is prioritized to be useful on a specific version *P′* of *P*. Such a test suite may be more effective at meeting the goal of the prioritization for *P′*. Our study in this paper focuses on the former kind.

Many coverage-based prioritization techniques (such as [4][25][26][33]) have been proposed, including prioritizing test cases by the total statement or branch coverage achieved by individual test cases, and by additional statement or branch coverage (or additional cost [33]) achieved by not-yet-selected test cases. Zhang et al. [41] generalized the total-and-additional test case prioritization strategies. Some techniques are not purely based on code coverage data of test cases such as prioritization based on test costs [5], fault severities [4], ability to detect specification-based faults [38], data from the test history [5][9], or fault-exposing-potential [26]. The effects of granularity [23] and compositions of test suites have been reported. Srivastava and Thiagarajan [28] built an *Echelon* system to prioritize test cases according to the potential change impacts of individual test cases between versions of a program to cover maximally the affected programs. Most of the existing experiments are conducted on procedural and object-oriented programs [3]. In addition, studies on prioritizing test cases using input domain information [7][39] and service discovery mechanisms [40] have been explored. Methods to reveal internal state transitions have also been developed [2][36].

Xu and Rountev [35] proposed a regression test selection technique for AspectJ programs. They use a control-flow representation for AspectJ software to capture aspect-related interactions and develop a graph comparison algorithm to select test cases. Martin et al. [12] gave a framework that generates and executes web-service requests, and collects the corresponding responses from web services. Using such request-response pairs, they test the robustness aspect of services. They discuss the potential of using request-response pairs for regression testing. Ruth [27] proposed a framework that automates safe regression test selection [24] for web services. Tsai et al. [29] proposed an adaptive group testing technique to address the challenges in testing a service-oriented application with a large number of web services simultaneously.

Using the mathematical definitions of XPath constructs [34] as rewriting rules, Mei et al. [15] developed a data structure known as an XPath Rewriting Graph (*XRG*). They propose an algorithm to construct XRGs and a family of unit testing criteria to test WS-BPEL applications. Their research group has also developed test case prioritization techniques for service testing [13][14][16][17][18]. However, they do not study the factors that may affect the fault detecting effectiveness in adverse scenarios.

III. TEST CASE PRIORITIZATION

This section introduces the set of test case prioritization techniques used in our empirical study. To study the effectiveness and tradeoff of different strategies and types of artifacts, we follow existing work [4][17][26] to compare them with two control techniques, namely random and optimal.

C1: Random ordering [4]. This strategy randomly orders the test cases in a test suite *T*.

C2: Optimal prioritization [4]. Given a program *P* and a set of known faults in *P*, if we also know which of the test cases can expose which faults in *P*, then we can figure out an optimal ordering of the test cases in the test suite *T* to maximize the fault detection rate of *T* for that set of faults. We adopt the definition presented in Rothermel et al. [26] to implement this technique. Specifically, test cases are iteratively selected by the ability of exposing the most faults not yet exposed. The remaining test cases are prioritized by the same method, until test cases that expose all the faults have been selected. As noted by Rothermel et al., such an optimal prioritization is not practical and only serves as an *approximation* to the optimal case, but it can be used as a technique for comparison purposes.

TABLE 1. FACTORS OF PRIORITIZATION TECHNIQUES
Strategy (A: Additional Greedy; T: Total Greedy; I: Iterative), Order Direction (A: Ascending; D: Descending)

Prioritization Techniques		Factors			
Ref.	**Name**	**Type of Artifact**	**Strategy**	**Order Direction**	**Are Coverage Data Obtained from Executable Artifacts?**
M1	Total-BPEL-Activity [4][26]	BPEL	T	D	Yes
M2	Addtl-BPEL-Activity [4][26]		A	D	Yes
M3	Total-BPEL-Workflow [4][26]		T	D	Yes
M4	Addtl-BPEL-Workflow [4][26]		A	D	Yes
M5	Total-XPath-Selection	XRG	T	D	Yes
M6	Addtl-XPath-Selection		A	D	Yes
M7	Ascending-XRG-Node		I	A	No
M8	Descending-XRG-Node		I	D	No
M9	Ascending-WSDL-Element [17]	WSDL	I	A	No
M10	Descending-WSDL-Element [17]		I	D	No

Apart from the two control techniques above, a total of 10 other techniques are examined in our empirical study. We recall that a WS-BPEL application includes three types of artifacts: BPEL, XPath, and WSDL. If we consider a BPEL program as a conventional program, then the next four techniques (M1–M4) resemble the statement and branch coverage-based techniques of conventional programs [4][26]. The remaining six techniques (M5–M10) explore the dimension of XML technologies to address the challenges caused by XPath and WSDL when using Greedy as the base test case prioritization techniques. We do so because Additional Greedy techniques [41] are still the most effective series of techniques (in terms of APFD) ever proposed. These 10 techniques are listed in Table 1.

A. BPEL Code Coverage Prioritization

This section presents four techniques using activity and workflow transition coverage of BPEL artifacts.

M1: Total BPEL activity coverage prioritization (Total-BPEL-Activity). Adapted from the *total-statement* technique presented in Elbaum et al. [4], this technique sorts the test cases in descending order of the total number of BPEL activities executed by each test case. If multiple test cases cover the same number of BPEL activities, M1 orders them randomly.

M2: Additional BPEL activity coverage prioritization (Addtl-BPEL-Activity). This technique iteratively selects a test case that yields the maximum cumulative BPEL activity coverage, and then removes the covered activities from the coverage information of each remaining test case. Additional iterations will be conducted until all the activities have been covered by at least one test case. If multiple test cases cover the same number of activities in the current coverage information of the test cases, M2 selects one of them randomly. Having achieved the complete coverage of all the activities by the prioritized subset of test cases in the given test suite, M2 resets the coverage information of each remaining test case to its initial value and then reapplies the algorithm to the remaining test cases. M2 is adapted from the *addtl-statement* technique used by Elbaum et al.

M3: Total BPEL workflow coverage prioritization (Total-BPEL-Workflow). This technique is the same as M1 (Total-BPEL-Activity) except that it uses test coverage measured in terms of BPEL workflow transitions rather than BPEL activities. It is adapted from the *total-branch* technique presented in Elbaum et al.

M4: Additional BPEL workflow coverage prioritization (Addtl-BPEL-Workflow). This technique is the same as M2 (Addtl-BPEL-Activity) except that it uses test coverage measured in terms of BPEL workflow transitions rather than BPEL activities. It is adapted from the *addtl-branch* technique presented in Elbaum et al.

B. XRG Coverage Prioritization

The next two techniques are inspired by M3, M4, and prioritization of XPath queries proposed by Mei et al. [15].

M5: Total XPath selection coverage prioritization (Total-XPath-Selection). This technique is the same as M3 (Total-BPEL-Workflow) except that it uses test coverage measured in terms of XPath selections rather than BPEL workflow transitions.

M6: Additional XPath selection coverage prioritization (Addtl-XPath-Selection). This technique is the same as M4 (Addtl-BPEL-Workflow) except that it uses test coverage measured in terms of XPath selections rather than workflow transitions. Similar to M4, after complete coverage using M6 has been achieved, this technique will reset the coverage of each remaining test case to its initial value and will then be reapplied to the remaining test cases.

The next two techniques (M7 and M8) are adapted from M9 and M10, respectively, by using XRG instead of WSDL as the artifacts to provide coverage data.

M7: Ascending XRG node coverage prioritization (Ascending-XRG-Node). This technique first partitions test cases into groups such that all the test cases with the same number of XRG nodes are placed in the same group. Suppose that the partitioning process results in $m+1$ groups G_0, G_1, ..., G_m, where G_i is a group of test cases each of which covers exactly i XRG nodes. This technique will select one test case randomly from a group starting from G_0 to G_m in ascending order of the index of the groups. The procedure is repeated until all the test cases in all the groups have been selected.

M8: Descending XRG node coverage prioritization (Descending-XRG-Node). This technique is the same as M7 (Ascending-XRG-Node) except that it selects a test case randomly from a group in descending order instead of ascending order of the group index.

C. WSDL Element Coverage Prioritization

In this section, we introduce the two techniques proposed in Mei et al. [17]. WSDL documents define the XML schemas used by WS-BPEL applications. Each XML schema contains a set of elements and can be used by one or more XPaths. Thus, the coverage data of these elements can reveal the usage of the internal messages among the workflow steps. For ease of presentation, we simply assume that all the XML schemas are included in WSDL documents. In this paper, we call an XML element z defined in any XML schemas as a *WSDL element*. If a test case t includes an XML message or causes the WS-BPEL application to generate an XML message that has an XML element z, then we say that the WSDL element z *is* covered by t.

M9: Ascending WSDL element coverage prioritization (Ascending-WSDL-Element) [17]. This technique is the same as M7 (Ascending-XRG-Node) except that it uses test coverage measured in terms of the elements in WSDL documents rather than XRG nodes.

M10: Descending WSDL element coverage prioritization (Descending-WSDL-Element) [17]. This technique is the same as M8 (Descending-XRG-Node) except that it uses test coverage measured in terms of the elements in WSDL documents rather than XRG nodes.

IV. Empirical Study

The empirical study aims to examine the following research questions.

RQ1: To what extent is a prioritization technique that is effective in average scenarios also effective in adverse scenarios?

RQ2: Do any of the following factors significantly affect the effectiveness of a technique in both the average and adverse scenarios: (i) the prioritization strategy, (ii) the type of artifacts used to provide coverage data, (iii) the ordering direction of the prioritization technique, and (iv) the executable nature of the artifacts?

A. *Experimental Setup*

To evaluate the techniques, we chose a benchmark suite of eight service-based subjects [1][8][32] (all developed in WS-BPEL) as representative service-based applications, as listed in Table 2. This benchmark suite was also used in previous empirical studies reported in existing work [13][14][15][16][17]. To the best of our knowledge, this suite is larger than that used by Ni et al. [19] in their experiment in terms of the number of individual subjects, the variety of subjects, the number of versions, and the size of individual subjects.

Each modified version had one fault seeded with three typical types of mutations [20], namely, value mutation, decision mutation, and statement mutation. Since BPEL can be treated as Control Flow Graphs (CFGs), the mutations were performed in the same way as seeding faults in CFGs. An XPath fault is a wrong usage of XPath expressions, such as extracting the wrong content or failing to extract any content. A WSDL fault is a wrong usage of WSDL specifications, such as binding to a wrong WSDL specification, or inconsistent message definitions. The faults in the modified versions have been reported in Mei et al. [15].

The statistics of the selected modified versions are shown in the rightmost column of Table 2. The size of each application, under the labels “Elements” and “LOC” in the table, refers to the number of XML elements and the lines of code in each WS-BPEL application. Other descriptive statistics of the benchmark suite are shown in the four rightmost columns in the table.

To facilitate the experiment, we implemented a tool to generate random test cases for each application. A thousand (1000) test cases were generated to form a test pool for each subject. We applied each constructed test case to each faulty version of the corresponding subject. To determine whether the test case revealed a failure, our tool compared its execution result against the original subject program with its result against a faulty version. If there is any difference, we deem the output of the faulty version reveals a failure.

Then, from each generated test pool, we randomly selected test cases one by one and put it into a test suite (which was initially empty). The selection was repeated until all the workflow branches, XRG branches, and WSDL elements had been covered at least once. This process was the same as that in the test suite construction in Elbaum et al. [4] and Mei et al. [18], except that we used the adequacy on BPEL, XRG and WSDL instead of that on program statements as the stopping criterion.

In total, we constructed 100 test suites for each subject. Table 3 shows the maximum, mean, and minimum sizes of the test suites. We followed existing work [3][17] to exclude a faulty version from data analysis if more than 20 percent of the test cases detected failures from the version. As such,

TABLE 2. Subjects and Their Descriptive Statistics

Ref.	Subject	Modified Versions	Elements	LOC	XPath Queries	XRG Branches	WSDL Elements	Used Versions
A	atm	8	94	180	3	12	12	5
B	buybook	7	153	532	3	16	14	5
C	dslservice	8	50	123	3	16	20	5
D	gymlocker	7	23	52	2	8	8	5
E	loanapproval	8	41	102	2	8	12	7
F	marketplace	6	31	68	2	10	10	4
G	purchase	7	41	125	2	8	10	4
H	triphandling	9	94	170	6	36	20	8
	Total	60	527	1352	23	114	106	43

TABLE 3. Statistics of Test Suite Sizes

Size \ Ref. / Subject	A	B	C	D	E	F	G	H	Mean
	atm	buybook	dslservice	gymlocker	loanapproval	marketplace	purchase	triphandling	
Maximum	146	93	128	151	197	189	113	108	140.6
Mean	95	43	56	80	155	103	82	80	86.8
Minimum	29	12	16	19	50	30	19	27	25.3

for each generated test suite, we further marked which test case reveals which fault.

B. Effectiveness Measure

Following most of the previous test case prioritization studies, we use the *Average Percentage of Faults Detected* (*APFD*) to measure the weighted average of the percentage of faults detected over the life of a test suite. As Elbaum et al. [4] pointed out, a high APFD value intuitively indicates a better fault detection rate. Let T be a test suite containing n test cases, and F be a set of m faults revealed by T. Let TF_i be the first test case in a reordering T' of T that reveals fault i. A higher APFD value indicates a more effective result [26]. The APFD value for T' is given by the formula

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{n\,m} + \frac{1}{2\,n}$$

C. Procedure

We applied each of the techniques C1, C2, and M1 to M10 to prioritize every generated test suite to produce an ordered test suite. Based on the failure marking of individual test cases in the ordered test suite, we computed the APFD value of each technique against each faulty version, and repeated the process 100 times.

D. Data Analyses

1. Average Scenarios

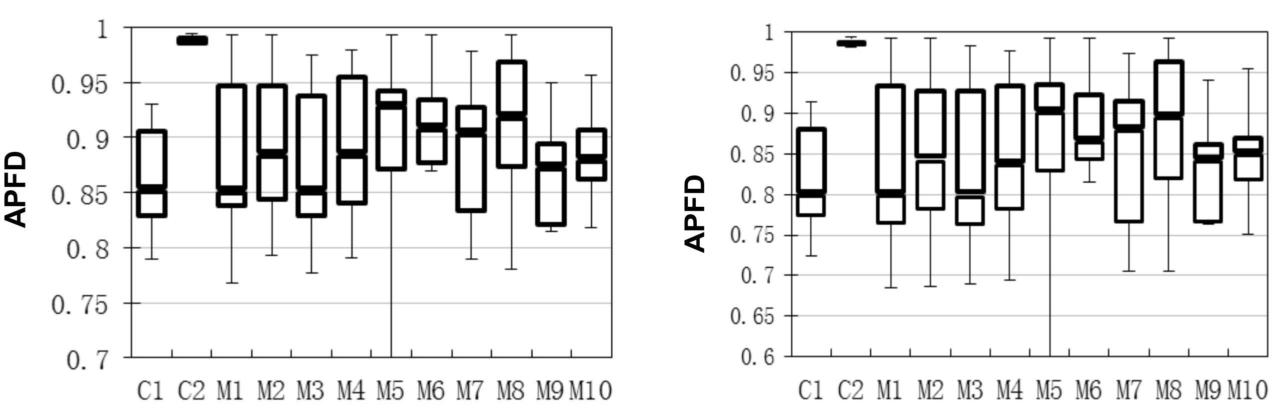
To understand the performance of the same techniques and factors, we first analyze the average scenarios.

The aggregated APFD results of C1, C2, and M1 to M10 on all the test suites against all the subjects are shown in Figure 1(a), where the *x*-axis shows the techniques C1, C2, and M1–M10 from left to right, and the *y*-axis shows the APFD values. Each box-plot in the graph shows the lowest, 25th percentile, median, 75th percentile and the highest APFD values of a prioritization technique. The breakdowns for individual subjects are shown in Figure 2, which can be interpreted in a similar way as Figure 1(a).

We first discuss the median APFD values achieved by the techniques in the average scenarios. Figure 1(a) shows that the median APFD values of M1 to M10 are generally higher than or equal to that of C1, indicating that techniques M1 to M10 can be effective in improving the fault detection rate in the average scenarios. Moreover, M5 to M8 are more effective than other techniques. Nonetheless, M5 appears to exhibit problems because there is a long line below the box for this technique, whereas C1 (*random ordering*) does not show a similar problem. This result indicates that in quite a number of cases, M5 is worse than C1 in the adverse scenarios, which confirms our motivations as stated in Section 1.

To find out the extent of differences among the techniques, we have performed hypothesis testing using Analysis of Variance (ANOVA). The test confirms that for each subject, techniques M1 to M10 differ statistically at a significance level of 5%. We further conducted a multiple-mean comparison using MatLab (with HSD [39], which is the default option for MatLab comparisons). The results are shown in Figure 3, in which the *x*-axis of each graph shows the APFD values while the *y*-axis shows the techniques C1, C2, and M1 to M10. If the bars of two techniques shown in a graph are non-overlapping, the difference in their effectiveness is statistically significant at a level of 5%.

We find that techniques M5 to M8 (which use the XRG artifacts for prioritization) are more effective than techniques M1 to M4 (which use the BPEL artifacts), while techniques M9 to M10 (which use the WSDL artifacts) applied to 4 (or 50%) of the 8 subjects are at least as effective as other techniques applied to two other subjects (or 25% of all the subjects). However, techniques M1 to M4, which use the executable artifacts (BPEL), can be either as effective as techniques that use the non-executable artifacts (if WSDL is involved) or worse than the latter (if XRG is involved). In fact, techniques M1 to M4 only outperform M7 to M10 (which use the non-executable artifacts WSDL or XRG) on one subject. On the other hand, comparing techniques that use the ascending order (M7 and M9) and those using the



(a) Comparison on all data

(b) Comparison in the adverse scenarios

Figure 1. Overall comparisons of the techniques using all the experimental subjects

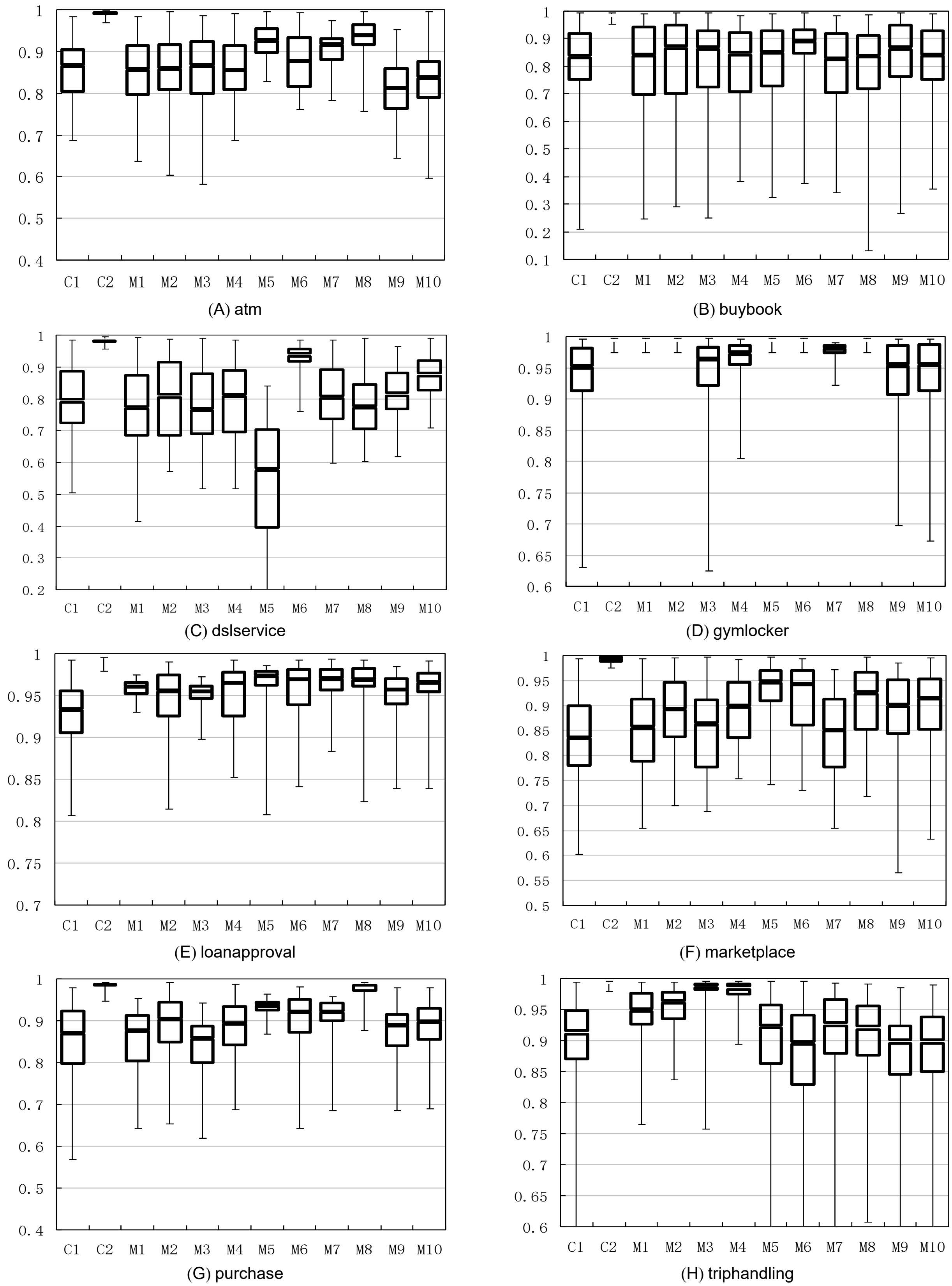


Figure 2. Comparisons of the techniques on all data of each individual subject (y-axis shows the APFD values)

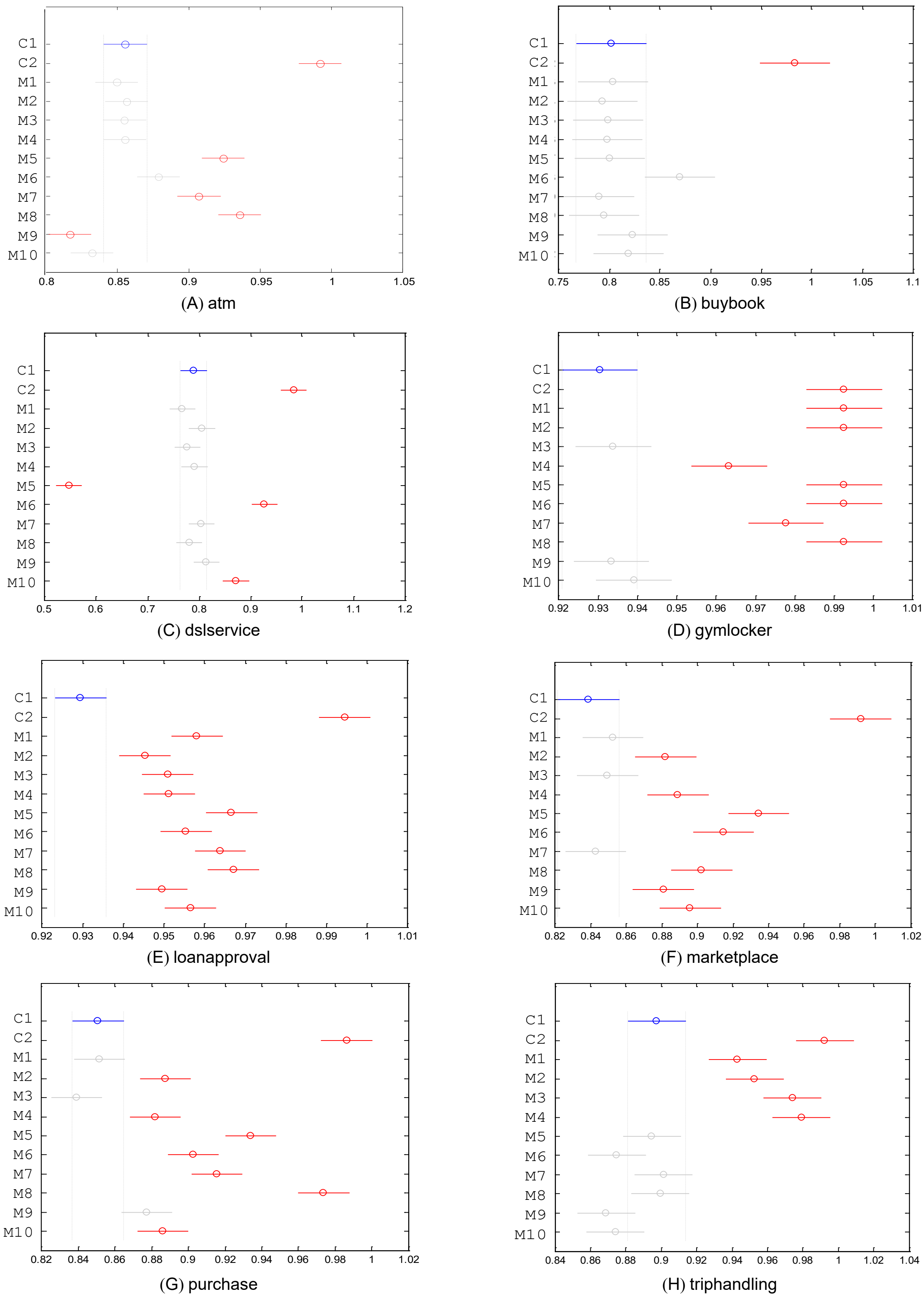


Figure 3. Multiple-mean comparisons of the techniques on all data of each individual subject (*x*-axis shows the APFD values)

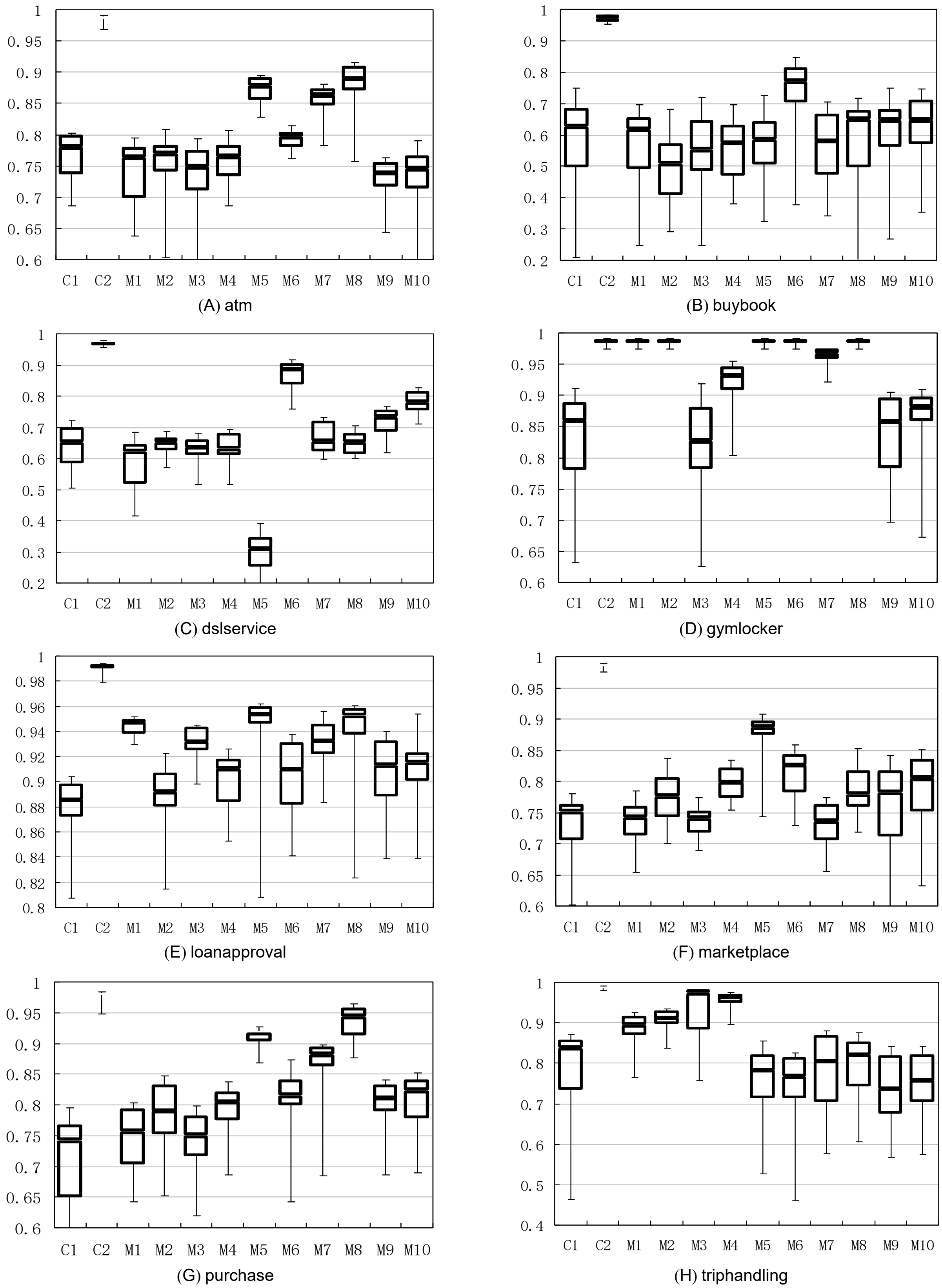


Figure 4. Comparisons of the techniques in the adverse scenarios of each individual subject (*y*-axis shows the APFD values)

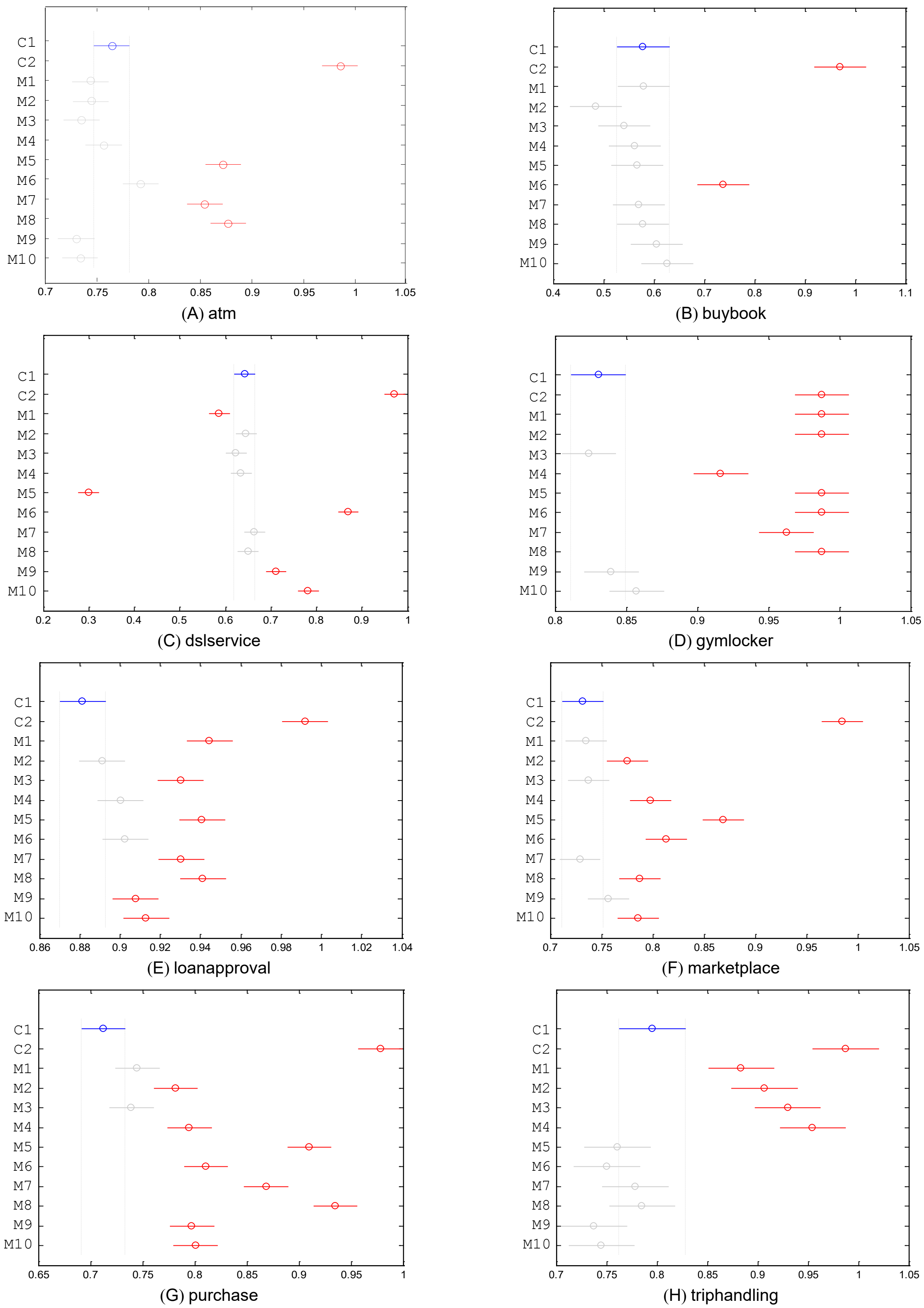


Figure 5. Multiple-mean comparisons of the techniques in the adverse scenarios of each individual subject (*x*-axis shows the APFD values)

descending order (M8 and M10), we notice no consistent trend of one ordering direction being more effective than the other. Last but not least, techniques M7 and M8 that use the iterative strategy can be more effective than techniques M1 to M6 that use the additional or total greedy strategy, even though the additional greedy strategy was found to be the best one for C++ or Java programs so far [41].

2. Adverse Scenarios

We define an *adverse scenario* as a test run in which the APFD value of a subject falls below the 25th percentile. In this section, we analyze the data for these adverse scenarios.

The aggregated APFD results of C1, C2, and M1 to M10 on the ordered test suites in the adverse scenarios against all the subjects are shown in Figure 1(b), which can be interpreted in a similar way as Figure 1(a). The breakdowns for individual subjects are shown in Figure 4, which can be interpreted in a similar way as Figure 2. The corresponding multiple mean comparisons are shown in Figure 5, which can be interpreted in a similar way as Figure 3.

From Figure 1(b), we find that in the adverse scenarios, each technique is about 5% less effective than that shown in Figure 1(a) in terms of the medians of the datasets. Between each pair of the corresponding plots, the effectiveness of each of M1 to M10 (relative to other techniques in the same plot) is consistent in terms of the medians of the datasets.

We have also performed the ANOVA test for the dataset depicted in Figure 1(b) similar to what we have presented in Section IV.D.1. The test shows again that techniques M1 to M10 are statistically different at a significance level of 5%. We further find from Figure 5 that, in general, techniques that use the XRG artifacts (M5–M8) are more effective than those using the BPEL artifacts (M1–M4), whereas applying techniques that use the WSDL artifacts (M9–M10) to 6 subjects (or 75% of all the subjects) are at least as effective as applying the other techniques to one other subject (or 12.5% of all the subjects). We only see techniques that use the executable artifacts (M1–M4) outperforming those using the non-executable artifacts (M7–M10) for one subject.

3. Comparisons of the Same Techniques between the Average Scenarios and the Adverse Scenarios

Based on the findings presented in Sections IV.D.1–2, we have the following observations. First, in both the average and adverse scenarios, prioritization is more effective with the use of the non-executable artifacts. This finding provides new evidence to support the investigation of using non-executable artifacts for test case prioritization. Second, our findings show no consistent trend that the use of a particular ordering direction will result in more effective techniques. However, unlike the average scenarios, we find that the differences between M5 and M6 on some subjects in the adverse scenarios are drastic, which reconfirm that the relatively stable performances in the average scenarios cannot be directly extrapolated to the adverse scenarios. Third, Figure 1(a) shows a long tail at the bottom of the box-plot of M5, meaning that it can sometimes perform very poorly. Comparing M5–M6 with M7–M8 or comparing M1–M6 with M7–M10, the iterative strategy (M7 to M10) tends to be more consistent in effectiveness than the additional or total greedy strategy (M1 to M6).

Table 4 summarizes the hypothesis testing results of the multiple mean comparisons presented in Sections IV.D.1–2 to validate each of M1 to M10 against *random ordering* (C1) in both the average scenarios (denoted by **All**) and the adverse scenarios (denoted by **25%**). Specifically, if a technique M*i* is significantly more effective than C1, we mark “>” in the cell; if M*i* does not differ significantly from C1, we mark “=” in the cell; and if M*i* is significantly worse than C1, we mark “<” in the cell.

We find that only M3, M7, M8, and M10 always do not perform worse than C1 (i.e., no “<” is marked in these columns) and at the same time produce consistent results (i.e., two corresponding cells sharing the same labels “<”, “=”, or “>”) for all the corresponding pairs in both the **All** and **25%** columns. The result also indicates that the use of the iterative strategy (M7, M8, and M10 but not M9; or 75% of the studied techniques using this strategy) can be a technique factor to consider in the two scenarios. On the other hand, only one technique (M3) out of six (M1 to M6)

TABLE 4. HYPOTHESIS TESTING COMPARISONS OF M1–M10 WITH C1 FOR ALL THE ORDERED TEST SUITES AND FOR THE LEAST EFFECTIVE 25% OF SUCH SUITES (AT A SIGNIFICANCE LEVEL OF 5%)

Ref.	Subject	M1		M2		M3		M4		M5		M6		M7		M8		M9		M10	
		All	**25%**	**All**	**25%**	**All**	**25%**	**All**	**25%**	**All**	**25%**	**All**	**25%**	**All**	**25%**	**All**	**25%**	**All**	**25%**	**All**	**25%**
A	**atm**	=	=	=	=	=	=	=	=	>	>	=	=	>	>	>	>	<	=	=	=
B	**buybook**	=	=	=	=	=	=	=	=	=	=	=	>	=	=	=	=	=	=	=	=
C	**dslservice**	=	<	=	=	=	=	=	=	<	<	>	>	=	=	=	=	=	>	>	>
D	**gymlocker**	>	>	>	>	=	=	>	>	>	>	>	>	>	>	>	>	=	=	=	=
E	**loanapproval**	>	>	>	=	>	>	>	=	>	>	>	=	>	>	>	>	>	>	>	>
F	**marketplace**	=	=	>	>	=	=	>	>	>	>	>	>	=	=	>	>	>	=	>	>
G	**purchase**	=	=	>	>	=	=	>	>	>	>	>	>	>	>	>	>	=	>	>	>
H	**triphandling**	>	>	>	>	>	>	>	>	=	=	=	=	=	=	=	=	=	=	=	=

can achieve a comparative result with C1. We do not find that the other three factors (namely, the type of artifact, the ordering direction, and whether the artifacts are executable) correlate with more than 50% of the studied techniques that show similar consistent results across the two types of scenarios and across all the subjects.

Finally, we answer the research questions posed in Section IV. To answer RQ1, we find that all the techniques (except M5) that are effective in the average scenarios can also be effective in the adverse scenarios, but the differences in effectiveness among techniques widen in the adverse scenarios. However, only some techniques (M3, M7, M8, and M10) show consistent effectiveness results over random ordering in both types of scenarios.

To answer RQ2, we find that out of all the choices in the four factors, only the iterative strategy under the *strategy* factor has a potential to be a significant factor in both types of scenarios.

Our result provides evidence to support Conjecture 1 that there exists a factor such that the resulting technique is effective in both the average and adverse scenarios. At the same time, Conjecture 1 is not generally established.

E. Threats to Validity

In this section, we discuss the threats to validity of the experiment. First, APFD is a commonly used measure of the effectiveness of a prioritization technique, but it cannot be computed unless the faults are known [4][26]. Zhai et al. [39] proved that APFD depends on the size of a test suite. Other measures such as APSC [11], FATE [38], or HMFD [40] may also be used to evaluate a prioritization technique. Second, we have tested our automated tool using small WS-BPEL programs. Third, our experiment used a suite of subjects and techniques to study the research questions. The use of other programs, test cases, faults, test oracles, and techniques may yield different results. The use of alternative definitions of the adverse scenarios may also yield other results. Our experiment has not used any mutation operators proposed to mutate XML documents. The use of such operators may produce results that are different from ours.

V. CONCLUSION

We have empirically studied test case prioritization techniques for WS-BPEL applications in both the average and adverse scenarios. We find that only 4 out of the 10 studied techniques are consistently effective in both types of scenarios, and only the iterative strategy shows its promise as a significant factor affecting the effectiveness of a technique in both types of scenarios. Our empirical study shows, however, that Conjecture 1 cannot be established.

In the future, we would like to study mechanisms to improve test case prioritization techniques with respect to the "lower end" spectrum of their effectiveness. Our work can be put into a broader context of software testing in general. We have raised a generic research question of whether it is sufficient to assess testing techniques in terms of their average performance. Our empirical results have provided counterexamples in test case prioritization techniques in regression testing. Do other types of testing techniques suffer from a similar problem that their average performance cannot be extrapolated effectively to the adverse scenarios? More studies should be carried out in these aspects in the future in order to transfer reliable research on testing techniques to the industry.

ACKNOWLEDGMENT

This work is supported in part by the Early Career Scheme and the General Research Fund of the Research Grants Council of Hong Kong (project numbers 111313, 123512, 125113, 716612, and 717811).

REFERENCES

[1] *alphaWorks Technology: BPEL Repository*, IBM, 2006, https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=AW-0KN.

[2] C. Bartolini, A. Bertolino, S.G. Elbaum, and E. Marchetti, "Bringing white-box testing to service oriented architectures through a service oriented approach," *Journal of Systems and Software*, vol. 84, no. 4, 2011, pp. 655–668.

[3] H. Do, G. Rothermel, and A. Kinneer, "Empirical studies of test case prioritization in a JUnit testing environment," *Proceedings of the 15th International Symposium on Software Reliability Engineering* (*ISSRE '04*), IEEE Computer Society, 2004, pp. 113–124.

[4] S.G. Elbaum, A.G. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, 2002, pp. 159–182.

[5] Y.-C. Huang, K.-L. Peng, and C.-Y. Huang, "A history-based cost-cognizant test case prioritization technique in regression testing," *Journal of Systems and Software*, vol. 85, no. 3, 2012, pp. 626–637.

[6] C. Jia, L. Mei, W.K. Chan, Y.T. Yu, and T.H. Tse, "Is XML-based test case prioritization for validating WS-BPEL evolution effective in both average and adverse scenarios?" *Proceedings of the 21st IEEE International Conference on Web Services* (*ICWS '14*), IEEE Computer Society, 2014.

[7] B. Jiang and W.K. Chan, "Bypassing code coverage approximation limitations via effective input-based randomized test case prioritization," *Proceedings of the IEEE 37th Annual Computer Software and Applications Conference* (*COMPSAC '13*), IEEE Computer Society, 2013, pp. 190–199.

[8] M.B. Juric, *A Hands-on Introduction to BPEL, Part 2: Advanced BPEL*, Oracle Technology Networks, http://www.oracle.com/technetwork/articles/matjaz-bpel2-082861.html.

[9] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," *Proceedings of the 24th International Conference on Software Engineering* (*ICSE '02*), ACM, 2002, pp. 119–129.

[10] H.K.N. Leung and L.J. White, "Insights into regression testing," *Proceedings of the IEEE International Conference on*

Software Maintenance (*ICSM '89*), IEEE Computer Society, 1989, pp. 60–69.

[11] Z. Li, M. Harman, and R.M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, 2007, pp. 225–237.

[12] E. Martin, S. Basu, and T. Xie, "Automated testing and response analysis of web services," *Proceedings of the IEEE International Conference on Web Services* (*ICWS '07*), IEEE Computer Society, 2007, pp. 647–654.

[13] L. Mei, Y. Cai, C. Jia, B. Jiang, and W.K. Chan, "Prioritizing structurally complex test pairs for validating WS-BPEL evolutions," *Proceedings of the IEEE International Conference on Web Services* (*ICWS '13*), IEEE *Computer* Society, 2013, pp. 147–154.

[14] L. Mei, Y. Cai, C. Jia, B. Jiang, and W.K. Chan, "Test pair selection for test case prioritization in regression testing for WS-BPEL programs," *International Journal of Web Services Research*, vol. 10, no. 1, 2013, pp. 73–102.

[15] L. Mei, W.K. Chan, and T.H. Tse, "Data flow testing of service-oriented workflow applications," *Proceedings of the 30th International Conference on Software Engineering* (*ICSE '08*), ACM, 2008, pp. 371–380.

[16] L. Mei, W.K. Chan, T.H. Tse, B. Jiang, and K. Zhai, "Preemptive regression testing of workflow-based web services," *IEEE Transactions on Services Computing*, vol. 8, no. 5, 2015, pp. 740–754.

[17] L. Mei, W.K. Chan, T.H. Tse, and R.G. Merkel, "XML-manipulating test case prioritization for XML-manipulating services," *Journal of Systems and Software*, vol. 84, no. 4, 2011, pp. 603–619.

[18] L. Mei, Z. Zhang, W.K. Chan, and T.H. Tse, "Test case prioritization for regression testing of service-oriented business applications," *Proceedings of the 18th International Conference on World Wide Web* (*WWW '09*), ACM, 2009, pp. 901–910.

[19] Y. Ni, S.-S. Hou, L. Zhang, J. Zhu, Z.J. Li, Q. Lan, H. Mei, and J.-S. Sun, "Effective message-sequence generation for testing BPEL programs," *IEEE Transactions on Services Computing*, vol. 6, no. 1, 2013, pp. 7–19.

[20] J. Offutt, A. Lee, G. Rothermel, R.H. Untch, and C. Zapf, "An experimental determination of sufficient mutant operators," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 2, 1996, pp. 99–118.

[21] A.K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM*, vol. 41, no. 5, 1998, pp. 81–86.

[22] S. Rapps and E.J. Weyuker, "Selecting software test data using data flow information," *IEEE Transactions on Software Engineering*, vol. 11, no. 4, 1985, pp. 367–375.

[23] G. Rothermel, S.G. Elbaum, A.G. Malishevsky, P. Kallakuri, and B. Davia, "The impact of test suite granularity on the cost-effectiveness of regression testing," *Proceedings of the 24th International Conference on Software Engineering* (*ICSE '02*), ACM, 2002, pp. 130–140.

[24] G. Rothermel and M.J. Harrold, "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 2, 1997, pp. 173–210.

[25] G. Rothermel and M.J. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, 1996, pp. 529–551.

[26] 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, 2001, pp. 929–948.

[27] M.E. Ruth, "Concurrency in a decentralized automatic regression test selection framework for web services," *Proceedings of the 15th ACM Mardi Gras Conference* (*MG '08*), ACM, 2008, pp. 7:1–7:8.

[28] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA '02*), ACM, 2002, pp. 97–106.

[29] W.-T. Tsai, Y. Chen, R.A. Paul, H. Huang, X. Zhou, and X. Wei, "Adaptive testing, oracle generation, and test case ranking for web services," *Proceedings of the 29th Annual International Computer Software and Applications Conference* (*COMPSAC '05*), vol. 1, IEEE Computer Society, 2005, pp. 101–106.

[30] *Web Services Business Process Execution Language Version 2.0: OASIS Standard*, Organization for the Advancement of Structured Information Standards (OASIS), 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[31] *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C, 2007, http://www.w3.org/TR/wsdl20/.

[32] *Web Services Invocation Framework: DSL Provider Sample Application,* Apache Software Foundation, 2006, http://svn.apache.org/viewvc/webservices/wsif/trunk/java/samples/dslprovider/README. html?view=co.

[33] W.E. Wong, J.R. Horgan, S. London, and H. Agrawal, "A study of effective regression testing in practice," *Proceedings of the 8th International Symposium on Software Reliability Engineering* (*ISSRE '97*), IEEE Computer Society, 1997, pp. 264–274.

[34] *XML Path Language (XPath) 2.0: W3C Recommendation*, W3C, 2007, http://www.w3.org/TR/xpath20/.

[35] G. Xu and A. Rountev, "Regression test selection for AspectJ software," *Proceedings of the 29th International Conference on Software Engineering* (*ICSE '07*), IEEE Computer Society, 2007, pp. 65–74.

[36] C. Ye and H.-A. Jacobsen, "Whitening SOA testing via event exposure," *IEEE Transactions on Software Engineering*, vol. 39, no. 10, 2013, pp. 1444–1465.

[37] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, 2012, pp. 67–120.

[38] Y.T. Yu and M.F. Lau, “Fault-based test suite prioritization for specification-based testing,” *Information and Software Technology*, vol. 54, no. 2, 2012, pp. 179–202.

[39] K. Zhai, B. Jiang, and W.K. Chan, “Prioritizing test cases for regression testing of location-based services: metrics, techniques, and case study,” *IEEE Transactions on Services Computing*, vol. 7, no. 1, 2014, pp. 54–67.

[40] K. Zhai, B. Jiang, W.K. Chan, and T.H. Tse, “Taking advantage of service selection: a study on the testing of location-based web services through test case prioritization,” *Proceedings of the IEEE International Conference on Web Services* (*ICWS ’10*), IEEE Computer Society, 2010, pp. 211–218.

[41] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, “Bridging the gap between the total and additional test-case prioritization strategies,” *Proceedings of the 2013 International Conference on Software Engineering* (*ICSE ’13*), IEEE, 2013, pp. 192–201.