To appear in *Proceedings of International Conference on Web Services 2013* (Research Track)

Prioritizing Structurally Complex Test Pairs for Validating WS-BPEL Evolutions[†]

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

Yan Cai
City University of Hong Kong
Tat Chee Avenue, Hong Kong
yancai2-c@my.cityu.edu.hk

Changjiang Jia
City University of Hong Kong
Tat Chee Avenue, Hong Kong
cjj.cs@my.cityu.edu.hk

Bo Jiang[‡]
School of Computer Science and Engineering
Beihang University, Beijing, China
jiangbo@buaa.edu.cn

W.K. Chan
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cityu.edu.hk

***Abstract*—Many web services represent their artifacts in the semi-structural format. Such artifacts may or may not be structurally complex. Many existing test case prioritization techniques however treat test cases of different complexity generically. In this paper, we exploit the insights on the structural similarity of XML-based artifacts between test cases, and propose a family of test case prioritization techniques that iteratively selects test case pairs without replacement. The validation experiment shows that these techniques can be more cost-effective than the studied existing techniques in exposing faults.**

***Keywords*—XML similarity, pariwse selection, adaptation.**

I. INTRODUCTION

A WS-BPEL web service [13] may interact with other web services that collectively implement a function. Any maintenance or runtime adaptation of the web service may result in faults or cause incompatible interactions between this web service and its belonging composite services. To validate whether an evolved version of the web service conforms to its previously established functional behaviors, a testing agent (which can be a web service) may apply a test suite to check whether the evolved version of the web service correctly handles the test suite. However, two XML-based messages sharing the same set of tags may structure these tags in quite different ways, potentially causing the same web service to produce radically different results for the two messages [13].

This paper proposes a suite of similarity-based test case prioritization techniques [4][9][14] for the regression testing of web services based on the pairwise selection strategy. Pairwise comparison is a fundamental strategy to examine elements in a finite set. To the best of our knowledge, no existing test case prioritization techniques that are formulated directly on top of this type of strategy for the regression testing of web services has been proposed.

Our techniques compute the structural similarity of XML-based artifacts between test cases. They progressively consider the XML-based artifacts in three levels: BPEL workflow process, WSDL interface specification, and XML-based messages [14]. Each technique assigns the execution priorities to the test cases in a regression test suite by assessing the similarity values of test case pairs via an iterative strategy. We have conducted an experiment to validate our techniques. The empirical results show that the proposed techniques can achieve higher rates of fault detection, in terms of APFD, than other studied techniques and random ordering. Interestingly, the results also show that a brute-force adaption of existing techniques to select test case pairs using XML documents fail to produce effective test case prioritization. Moreover, they may be even less effective than the random ordering of the test suite.

The main contribution of this paper is twofold: (i) To the best of our knowledge, this paper is the first work that formulates a direct proposal using pairwise selection for test case prioritization techniques in the testing of web services. (ii) We report the first experiment that validates the effectiveness of pairwise test case selection strategy, and demonstrates that a simple extension of existing techniques to pairwise test case selection can be undesirable when testing WS-BPEL web services.

The rest of this paper is organized as follows. Section II revisits the preliminaries. Section III gives an example to motivate the work. We present our techniques and the evaluations in Sections IV and V respectively. Section VI reviews the related work. Section VII concludes the paper.

II. PRELIMINARIES

A. Test Case Prioritization

The problem of test case prioritization [5] is as follows:

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

Problem: To find $T' \in PT$ such that, $\forall T'' \in PT \Rightarrow f(T') \geq f(T'')$.

B. XML Distance and XML Set Similarity

An XML document can be modeled as an *ordered, labeled tree T* [7][8]. Given two XML document trees T_1 and T_2, the *tree edit distance* [8] between them, denoted by $TDIST(T_1, T_2)$, is defined as the minimum cost sequence of tree edit operations (that are, node insertions, deletions, and label substitutions) on single tree nodes that are required to

[†] This work is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (project nos. 111410, 123512, and 717811) and the National Natural Science Foundation of China (project no. 61202077).

[‡] Correspondence Author

transform one tree to another. We use the algorithm from Guha et al. [8] to calculate *TDIST* (T_1, T_2). We extend an existing similarity measure [7] to define our similarity metric between two sets of XML documents in the spirit of the standard *Jaccard similarity*: Given two sets of XML documents S_1 and S_2, the similarity between them is defined by π (S_1, S_2) as follows:

$$\pi(S_1, S_2) = 1 - \frac{\sum_{(U,V)\in S_1\times S_2} TDIST(U,V)}{\sum_{(U,V)\in S_1\times S_2} (|U \cup V|)} \qquad \text{(E-1)}$$

where U and V are XML documents and $|U \cup V|$ is the total number of unique XML node labels in U and V. We choose the Jaccard coefficient in this paper because of its generality. A generalization to the other coefficients is feasible.

III. Motivating Example

A. Scenario

This example is taken from *TripHandling* [3]. We follow [14] to use a UML activity diagram to depict this business process in Figure 1(a). In brief, this process receives users' hotel book request, then invokes web services to find the requested rooms. Finally, it replies the result to users. Figure 1(b) further highlights a scenario of service evolution. In Figure 1, a node represents a workflow step, and a link represents a transition between two workflow steps. The nodes are annotated with information such as the input-output parameters and XPath queries [20] that are used to extract the required contents from the XML messages. We number the nodes as A_i (for i from 1 to 8) to ease the illustration.

Suppose that a self-adaptation occurs to the WS-BPEL service in Figure 1(a), which in finding a good adapted service composition, the service changes to a candidate service shown in Figure 1(b). It is however unknown whether this candidate service can be functionally compatible with the other services in the original service composition (even after a successful service selection procedure). This problem urges for a runtime validation, which is a round of regression testing in this scenario.

This adaptation changes the precondition at node A_4 in Figure 1(a) to that at node A_4 in Figure 1(b), and adds a validation at node A_5 to guarantee that the room number information ("roomno") is non-empty. This adaptation attempts to allow customers to select any room that can provide accommodation for the requested number of people.

However, the evolved version of the business process only changes the precondition in the XPath (namely, changing "and" to "or"). Although such adaptations aim to provide customers more choices, yet this particular evolved version does not support the intention. For example, it may immediately proceed to book rooms, but is unable to provide options for customers to select. It is desirable to detect the failures from the candidate version, which allows the self-adaption procedure to discard this candidate version and try other candidates as soon as possible.

We use six test cases (i.e., t_1 to t_6 [14]) to illustrate how our

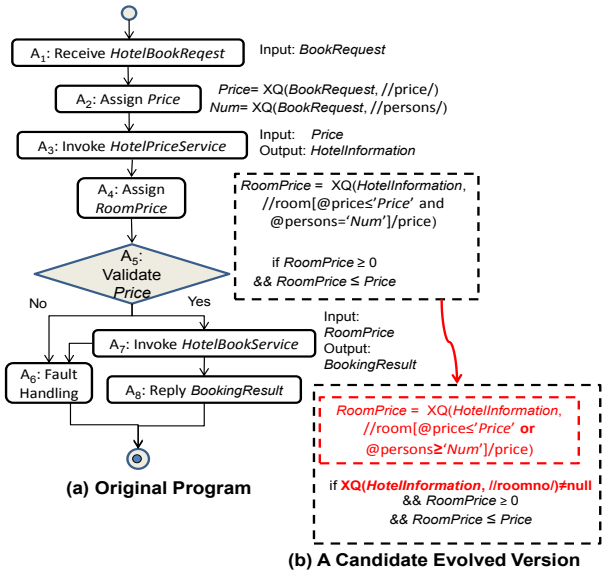

Figure 1. Activity diagram of a WS-BPEL service (adapted from [14]).

techniques reorder the test set and apply them to test the candidate service. Each test case gives an input of the variable *BookRequest* at node A_1. To save space, we use the price value of the variable *Price* and the numerical value of the variable *Num* to stand for the variables, rather than using the original XML formats.

⟨Price, Num⟩	⟨Price, Num⟩
Test case 1 (t_1): ⟨200, 1⟩	**Test case 2 (t_2): ⟨150, 2⟩**
Test case 3 (t_3): ⟨125, 3⟩	**Test case 4 (t_4): ⟨100, 2⟩**
Test case 5 (t_5): ⟨ 50, 1⟩	**Test case 6 (t_6): ⟨ –1, 1⟩**

We note that there are messages sent and received at both node A_3 and node A_7. Figure 3 shows an XML schema in a WSDL document that defines the message type for the messages replied by the service *HotelPriceService* (at A_3).

Moreover, message contents are used in various workflow activities. For example, the messages used at A_4 for t_1 to t_6 are listed in Figure 2.

Let us further consider how these messages are used at A_3. When running the candidate version (Figure 1(b)) over t_1 to t_6, t_1 extracts a right room price; t_4 to t_6 extract no price value; both t_2 and t_3 extract the price 105 of the single room, while they indeed aim to book a double room and a family suite, respectively. We also find that both t_2 and t_3 can detect the fault in the evolved candidate service presented in Figure 1(b).

B. Coverage Analysis and Problems

Suppose that the workflow branch coverage achieved by each test case over the original version (Figure 1(a)) is shown as in Table I. We use a solid dot "•" to refer to an item covered by the respective test case. For instance, t_1 covers six workflow branches (shown as edges in Figure 1(a)): $\langle A_1, A_2\rangle$, $\langle A_2, A_3\rangle$, $\langle A_3, A_4\rangle$, $\langle A_4, A_5\rangle$, $\langle A_5, A_7\rangle$ and $\langle A_7, A_8\rangle$. Table II further presents an example on how t_1 to t_6 cover the WSDL elements of the original version (Figure 1(a)). We record the coverage of WSDL elements in the first

```
1  <xsd:complexType name="hotel">
2     <xsd:element name="name" type="xsd:string"/>
3     <xsd:element name="room" type="xsd:RoomType"/>
4     <xsd:element name="error" type="xsd:string"/>
5  </xsd:complexType>
6  <xsd:complexType name="RoomType">
7     <xsd:element name="roomno" type="xsd:int" />
8     <xsd:element name="price" type="xsd:int"/>
9     <xsd:element name="persons" type="xsd:int"/>
10 </xsd:complexType>
```

Figure 3. WSDL document fragment: XML schema of *hotel*.

TABLE I. WORKFLOW BRANCH COVERAGE FOR T1 TO T6 ON THE ORIGINAL SERVICE

Branch	t_1	t_2	t_3	t_4	t_5	t_6
$\langle A_1, A_2 \rangle$	•	•	•	•	•	•
$\langle A_2, A_3 \rangle$	•	•	•	•	•	•
$\langle A_3, A_4 \rangle$	•	•	•	•	•	•
$\langle A_4, A_5 \rangle$	•	•	•	•	•	•
$\langle A_5, A_6 \rangle$		•	•		•	•
$\langle A_5, A_7 \rangle$	•			•		
$\langle A_7, A_6 \rangle$						
$\langle A_7, A_8 \rangle$	•			•		
Total	6	5	5	6	5	5

TABLE II. STATISTICS OF WSDL ELEMENTS FOR T_1 TO T_6

	XML schema	t_1	t_2	t_3	t_4	t_5	t_6
WSDL artifact with XML message (*)	*hotel*	•	•	•	•	•	•
	name	•	•	•			
	room	•	•	•	•		•
	roomno	•	•	•	•		
	price	•	•	•	•		•
	persons	•	•	•	•		•
	error						•
Dynamic XML Message (**)	*val*(*name*)	•	•	•			
	val(*roomno*)	•	•	•			
	val(*price*)	•	•	•	•		•
	val(*persons*)	•	•	•	•		•
	val(*error*)						•
	Total	10	10	10	7	1	8

part (annotated with an asteria "*") in Table II and the coverage of the tags in XML message in the second part (annotated with "**"). Both parts are reported by collecting the XML messages of the original service over the respective test cases.

We show below a few possible selection orders on the set of test cases by applying the *additional* (or *addtl* for short) and *total* coverage strategies [5] on the workflow branches (Table I), the combination of the workflow branches and WSDL elements (Table II, part *), and the combination of the workflow branches, WSDL elements, and XML messages (Table II, part * & part **), respectively. We choose these techniques because the *additional* and *total* coverage strategies [5] are consistently reported to be very effective test case prioritization strategies in the literature. The *Total-Workflow-Coverage* and *Addtl-Workflow-Coverage* techniques are also known as the *Total-CM1* and *Addtl-CM1* techniques reported by Mei et al. [14].

Test Case 1	Test Case 2	Test Case 3
<hotel> <name>Hilton Hotel</name> <room> <roomno>R106</roomno> <price>105</Price> <persons>1<persons> </room> <room> <roomno>R101</roomno> <price>150</price> <persons>3<persons> </room> </hotel >	<hotel> <name>Hilton Hotel</name> <room> <roomno>R106</roomno> <price>105</Price> <persons>1<persons> </room> <room> <roomno>R101</roomno> <price>150</price> <persons>3<persons> </room> </hotel >	<hotel> <name>Hilton Hotel</name> <room> <roomno>R106</roomno> <price>105</Price> <persons>1<persons> </room> </hotel >
Test Case 4	**Test Case 5**	**Test Case 6**
<hotel> <room> <roomno></roomno> <price>100</Price> <persons>2<persons> </room> </hotel >	<hotel> </hotel >	<hotel> <room> <price>-1</Price> <persons>1<persons> </room> <error>InvalidPrice<error> </hotel >

Figure 2. XML messages for *XQ*(*HotelInformation*, //room[@price ≤ 'Price' and @persons = 'Num']/price/).

Techniques	Test case orderings (in descending order of priority)
Addtl-Workflow-Coverage:	⟨ t_1, t_5, t_4, t_6, **t_2, t_3** ⟩
Total-Workflow-Coverage:	⟨ t_1, t_4, t_6, **t_3, t_5, t_2** ⟩
Addtl-Workflow-WSDL-Coverage:	⟨ t_1, t_6, t_4, **t_3, t_2**, t_5 ⟩
Total-Workflow-WSDL-Coverage:	⟨ t_1, t_4, **t_3, t_2**, t_6, t_5 ⟩
Addtl-Workflow-XML-Coverage:	⟨ t_1, t_6, **t_2**, t_4, **t_3**, t_5 ⟩
Total-Workflow-XML-Coverage:	⟨ t_1, **t_2, t_3**, t_6, t_4, t_5 ⟩

None of these techniques effectively prioritizes t_2 or t_3. That is, they rely on their tie breaking strategies instead of the intrinsic ability of such a technique to assign either test case with high priority. The test suite contains quite a number of test cases that are similar to them. Using a bin counting approach or a traditional test case clustering approach may not help iron out them effectively.

IV. OUR APPROACH: XSP

A. Test Case Similarity

From a test execution on XML-manipulating services, one may collect the coverage information on service code and WSDL documentation, and collect XML messages. Moreover, many researchers consider that services can be black-box, and thus the service structure (i.e., BPEL code) may not be available for testing. Therefore, we first use WSDL documents, then add XML messages, and finally include BPEL code in case code can be available.

To ease the presentation, we define a container (see Definition 1) to hold different kinds of XML documents used in a test case.

Definition 1. *W3-Set* (or *W3S*). A *W3-Set* with respect to a test case t is a set of triples $\{\langle w_1, m_1, b_1\rangle, \langle w_2, m_2, b_2\rangle, \ldots, \langle w_N, m_N, b_N\rangle\}$, where a triple $\langle w_i, m_i, b_i\rangle$ is a workflow module b_i, an XML message m_i, and a WSDL specification w_i for the module b_i and it defines the type for the message m_i. Let $W(t)=\{w_1, w_2, \ldots, w_N\}$, $M(t)=\{m_1, m_2, \ldots, m_N\}$, and $B(t)=\{b_1, b_2, \ldots, b_N\}$ represent the set of WSDL specifications, the set of XML messages, and the set of workflow modules, used or exercised in the execution of t, respectively. □

A workflow module (such as A_i in Figure 2) may also be encoded in the XML format [3][16][18]. Take the test case t_1 in Section III for example: $M(t_1)$ and $W(t_1)$ are given in Figure 2 and Figure 3, respectively. $B(t_1)$ is $\{A_1, A_2, A_3, A_4, A_5, A_7, A_8\}$. We call an XML node label in either $W(t)$, $M(t)$, or $B(t)$ an *element* covered by a test case t. We further define the concept of test case similarity in Definition 2.

Definition 2. *Test Case Similarity* (or *W3-Similarity*).** We define three levels of similarity metrics between two test cases t_i and t_j (namely *W3-Similarity*). (i) Similarity of **W**SDL specification (*W*-I). (ii) Similarity of **W**SDL specification and **W**SDL-governed XML message (*W*-II). (iii) Similarity of **W**SDL specification, **W**SDL-governed XML message, and **W**orkflow module (*W*-III). □

For a test case t, we call the set of elements covered by t using *W*-I, *W*-II, and *W*-III as *WE*-I (t), *WE*-II (t), and *WE*-III (t), respectively. These sets satisfy the equations:

$$WE\text{-I}(t) = W(t)$$
$$WE\text{-II}(t) = W(t) \cup M(t)$$
$$WE\text{-III}(t) = W(t) \cup M(t) \cup B(t)$$

Let the *W3-Set* of test cases t_i and t_j be $\langle W_i, M_i, B_i\rangle$ and $\langle W_j, M_j, B_j\rangle$, respectively. Let the similarity between XML messages, between WSDL specifications, and between workflow modules for t_i and t_j be $\pi(M_i, M_j)$, $\pi(W_i, W_j)$, and $\pi(B_i, B_j)$, respectively. There are many ways to define the similarity metrics. In our approach, we use the XML similarity metric to produce a percentage value of similarity between two sets of XML messages. Meanwhile, we use the *Geometric Mean* (GM) to define the three metrics of *W3-Similarity* for t_i and t_j as follows. It is because one piece of code may associate with many XML Schemas, and one XML Schema may govern contents of many XML messages. Because of this multi-level one-to-many relationship, we use GM rather than other means to combine data from different dimensions.

We consider $W(t)$, $M(t)$, or $B(t)$ as three dimensions that describe a test case t. Hence, each test case t can be regarded as an axis-aligned cube with edge length equal to one in the three dimensional space formed by dimension W, M, and B. If t_i and t_j do not completely differ, the two cubes should overlap and overlap is a cuboid. The similarity values along three dimensions can be represented by the edge lengths of the cuboid, namely $\pi(W_i, W_j)$, $\pi(M_i, M_j)$, and $\pi(B_i, B_j)$. Let the volume of overlap cuboid be $V = \pi(W_i, W_j) \times \pi(M_i, M_j) \times \pi(B_i, B_j)$. However, the unit of the volume is cube percentage. Hence, we compute cube root of V, as *W*-III shown by (E-4) to describe the similarity between t_i and t_j. We can see that *W*-III is actually the GM of the similarity metrics $\pi(W_i, W_j)$, $\pi(M_i, M_j)$, and $\pi(B_i, B_j)$. The formula (E-4) considers three dimensions, the formula (E-2) considers only one dimension W, and the formula (E-3) considers two dimensions W and M, shown as follows.

$$W\text{-}I = \pi(W_i, W_j) \quad \text{(E-2)}$$

$$W\text{-}II = \sqrt[2]{\pi(W_i, W_j) \times \pi(M_i, M_j)} \quad \text{(E-3)}$$

$$W\text{-}III = \sqrt[3]{\pi(W_i, W_j) \times \pi(M_i, M_j) \times \pi(B_i, B_j)} \quad \text{(E-4)}$$

We note that although we illustrate our techniques using three levels, generalizing them to handle more than three levels is simple.

B. Test Case Prioritization Techniques

We use *W-i* (where $i \in \{\text{I, II, III}\}$) to denote the three metrics in *W3-Similarity* used in our techniques. Moreover, to help evaluate them in Section V, we compare them with *random* (C1) and another technique adopted from conventional *total-branch techniques* [5] that use *WE-i* (where $i \in \{\text{I, III, III}\}$) as the source of coverage data, which we denote it by C2. Moreover, we include two adapted techniques C3 and C4. C3 is just an adaptation of C2 by using *W-i* rather than *WE-i* as the metrics, and C4 is just adapts from C3 slightly. We choose then to validate whether a simple extension of existing techniques may be adequate. We further propose two techniques (M1 and M2) that formulate our idea. We present all of them in this section.

We firstly define an auxiliary function: Let T be a test suite. We partition all pairs of distinct test cases into K groups, each containing all those pairs with the same *W-i* similarity value. We denote each group by G_k ($1 \le k \le K$), where k is known as the *group index*. All test case pairs in G_k have the same *W-i* similarity value g_k, such that a smaller group index k indicates a larger *W-i* similarity value g_k. We refer to such handling as the *grouping function* ***GF***.

We categorize the 15 test case pairs among t_1 to t_6 into different groups, and the results are shown in the leftmost columns in Tables IV-VI. The rightmost two columns of each table show one possible ordering of test case pairs for C3, C4, M1, and M2 each, and the corresponding selected pairs. We mark the selection sequence in the "Seq." columns of these three tables. We will explain them in the following sub-sections.

(1) Benchmark Techniques

C1: Random ordering [5]. This technique randomly orders the test cases in a test suite T.

The "imported" techniques (C2) directly use *W3S* (using *WE*-I, *WE*-II, and *WE*-III) to prioritize test cases. We adapted it from [14].

C2: Total *WE-i* coverage prioritization (*Total-WE-Coverage*). C2 sorts the test cases in descending order of the

TABLE III. STATISTICS OF TEST CASE SIMILARITIES (*W*-I).

Similarity	*Group*		Selected Test Cases in Order				
	Index	**Test Case Pairs**	**Seq.**	**C3**	**C4**	**M1**	**M2**
1.000	G_1	(t_1, t_2), (t_1, t_3), (t_2, t_3)	1	$(\boldsymbol{t_1}, \boldsymbol{t_2})$	$(\boldsymbol{t_1}, \boldsymbol{t_5})$	$(\boldsymbol{t_1}, \boldsymbol{t_5})$	(t_1, t_2)
			2	$(t_1, \boldsymbol{t_3})$	$(\boldsymbol{t_2}, t_5)$	$(\boldsymbol{t_4}, t_5)$	(t_2, t_4)
0.833	G_2	(t_1, t_4), (t_2, t_4), (t_3, t_4)	3	(t_2, t_3)	$(\boldsymbol{t_3}, t_5)$	$(t_1, \boldsymbol{t_6})$	(t_4, t_6)
			4	$(t_1, \boldsymbol{t_4})$	$(t_5, \boldsymbol{t_6})$	(t_4, t_6)	(t_3, t_6)
0.667	G_3	(t_4, t_6)	5	(t_2, t_4),	$(\boldsymbol{t_4}, t_5)$	$(\boldsymbol{t_3}, t_4)$	(t_5, t_6)
0.571	G_4	(t_1, t_6), (t_2, t_6), (t_3, t_6)	6	(t_3, t_4)		$(\boldsymbol{t_2}, t_3)$	
			7	$(t_4, \boldsymbol{t_6})$			
0.200	G_5	(t_4, t_5), (t_5, t_6)	8	(t_1, t_6)			
0.167	G_6	(t_1, t_5), (t_2, t_5), (t_3, t_5)	9	(t_2, t_6)			
			10	(t_3, t_6)			
			11	$(t_4, \boldsymbol{t_5})$			

TABLE IV. STATISTICS OF TEST CASE SIMILARITIES (*W*-II).

Similarity	*Group*		Selected Test Cases in Order				
	Index	**Test Case Pairs**	**Seq.**	**C3**	**C4**	**M1**	**M2**
1.00	G_1	(t_1, t_2), (t_1, t_3), (t_2, t_3)	1	(t_1, t_2)	(t_1, t_5)	(t_1, t_5)	(t_1, t_2)
			2	(t_1, t_3)	(t_2, t_5)	(t_5, t_6)	(t_2, t_4)
0.76	G_2	(t_1, t_4), (t_2, t_4), (t_3, t_4)	3	(t_2, t_3)	(t_3, t_5)	(t_4, t_5)	(t_4, t_6)
			4	(t_3, t_4)	(t_5, t_6)	(t_2, t_6)	(t_3, t_6)
0.67	G_3	(t_4, t_6)	5	(t_2, t_4)	(t_4, t_5)	(t_4, t_6)	(t_4, t_5)
0.53	G_4	(t_1, t_6), (t_2, t_6), (t_3, t_6)	6	(t_1, t_4)		(t_3, t_4)	
			7	(t_4, t_6)			
0.17	G_5	(t_4, t_5)	8	(t_1, t_6)			
0.16	G_6	(t_5, t_6)	9	(t_2, t_6)			
0.13	G_7	(t_1, t_5), (t_2, t_5), (t_3, t_5)	10	(t_3, t_6)			
			11	(t_4, t_5)			

TABLE V. STATISTICS OF TEST CASE SIMILARITIES (*W*-III).

Similarity	*Group*		Selected Test Cases in Order				
	Index	**Test Case Pairs**	**Seq.**	**C3**	**C4**	**M1**	**M2**
1.00	G_1	(t_2, t_3)	1	(t_2, t_3)	(t_1, t_5)	(t_1, t_5)	(t_2, t_3)
0.84	G_2	(t_1, t_4)	2	(t_1, t_4)	(t_4, t_5)	(t_4, t_5)	(t_1, t_4)
0.83	G_3	(t_1, t_2), (t_1, t_3)	3	(t_1, t_2)	(t_2, t_5)	(t_2, t_5)	(t_1, t_2)
0.69	G_4	(t_2, t_4), (t_3, t_4)	4	(t_1, t_3)	(t_3, t_5)	(t_5, t_6)	(t_2, t_4)
0.66	G_5	(t_2, t_6), (t_3, t_6)	5	(t_3, t_4)	(t_5, t_6)	(t_1, t_6)	(t_3, t_6)
0.63	G_6	(t_4, t_6)	6	(t_2, t_4)		(t_4, t_6)	(t_4, t_6)
0.55	G_7	(t_1, t_6)	7	(t_3, t_6)		(t_3, t_6)	(t_1, t_6)
0.29	G_8	(t_5, t_6)	8	(t_2, t_6)			(t_5, t_6)
0.26	G_9	(t_2, t_5), (t_3, t_5)	9	(t_4, t_6)			
0.25	G_{10}	(t_4, t_5)	10	(t_1, t_6)			
0.21	G_{11}	(t_1, t_5)	11	(t_5, t_6)			

total number of elements that each test case t has covered (i.e., the number of elements in *WE*-*i*(t)). If multiple test cases cover the same number of elements, C2 will order these test cases randomly.

C3 prioritizes the *most similar* test cases in pairs to be executed first. Turning C3 the other way round, we also use C4 to select the *least similar* test cases in pairs to have higher priorities.

C3: Maximum *W-i* Similarity prioritization (*Total-W-Similarity*). The technique invokes the grouping function GF using *W-i*. The technique selects a pair of test cases with the greatest similarity value (i.e., g_1) using *W-i*, and randomly chooses one test case t in this pair. The technique continues to select all pairs of test cases containing t from the same group. If multiple test case pairs contain t, the technique randomly selects one pair to break the tie. C3 discards any test case in a selected pair if the test case has been included by a previously selected pair. C3 repeats the above selection process first for the group, and once all test cases in the group have been selected, then among the remaining groups in the ascending order of the group index (i.e., from G_2 to G_M) until every unique test case has been selected.

The test cases selected by C3 using W-I are highlighted under the "C3" column in Table III. Other columns in Table IV and Table V can be interpreted similarly.

C4: Minimum *W-i* similarity prioritization *(Total-W-Dissimilarity)*. This technique is the same as C3 except that it first selects a pair of test cases with the minimum similarity value using *W-i* (rather than the maximum *W-i* similarity value according to C3), and C4 repeats the selection process among the remaining groups in ascending order of the group index.

For example, the test cases selected by C4 using *W*-I are highlighted under the "C4" column in Table III.

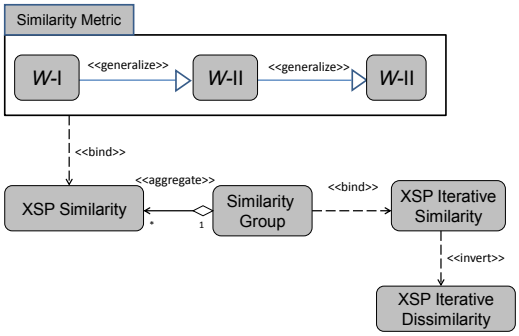

Figure 4. The relations between XSP (dis)similarity and their metrics.

(2) Our Test Case Prioritization Techniques: XSP

As mentioned in Section I, our techniques use **X**ML, **S**imilarity metric, and **P**airs of test cases. We therefore refer our techniques to as ***XSP***.

Figure 4 shows the schematic relationships among XSP (dis)similarity and their metrics. Intuitively, a larger similarity value between two test cases suggests that they have a higher chance in covering the same set of XML document structures.

TABLE VII. PRIORITIZATION TECHNIQUES AND EXAMPLES.

Technique	Index	Order of t_1–t_6					
		t_1	t_2	t_3	t_4	t_5	t_6
XSP-Iterative-Dissimilarity (*W*-I)	M1	2	6	5	3	1	4
XSP-Iterative-Similarity (*W*-I)	M2	1	2	5	3	6	4
XSP-Iterative-Dissimilarity (*W*-II)	M1	1	5	6	4	2	3
XSP-Iterative-Similarity (*W*-II)	M2	2	1	5	3	6	4
XSP-Iterative-Dissimilarity (*W*-III)	M1	1	5	6	3	2	4
XSP-Iterative-Similarity (*W*-III)	M2	3	1	2	4	6	5

We propose M1 and M2, each of which selects, in turn and iteratively, one test case pair from each group in the series of groups, skipping any group having been exhausted. M1 and M2 sample the groups in ascending and descending orders (i.e., from G_1 to G_M, and from G_M to G_1), respectively, of the group index.

M1: Ascending *W-i* similarity prioritization (*XPS-Iterative-Dissimilarity*). The technique invokes the grouping function GF using *W-i*. Then the technique samples all groups G_1, …, G_k, …, G_M in ascending order of the group index k by selecting one pair of test cases, if any, from each group in turn. The technique discards any test case in a selected pair if the test case has been selected. The technique then removes the selected pair from the group. M1 repeats the selection process among the non-empty groups until all the test cases have been selected.

For example, the test cases selected by M1 using *W*-I as the metric are highlighted under the "M1" column in Table III. The columns for M1 in Table IV and Table V can be interpreted similarly.

M2: Descending *W-i* similarity prioritization (*XPS-Iterative-Similarity*). This technique is the same as M1 except that it samples the groups G_M, …, G_k, …, G_1 in descending order of the group index k, rather than in ascending order.

We summarize the result of M1 and M2 on the running example in Table VII. The same result can also be manually computed using the data in Table III, Table IV, and Table V.

V. EXPERIMENT

A. Experimental Design

(1) Subjects, Versions, and Test Suites

We choose eight WS-BPEL applications [3][16][19] to evaluate our techniques, which are shown in Table II, because these applications have also served as benchmarks or illustrative textbook examples, and have been used in previous test case prioritization experiments [13][14]. This set of benchmarks is also larger than the one used by Ni et al. [15]. Like many experiments on test case prioritization for regression testing, we use a set of known faults on the modified versions and the test suites associated with the original version of these subjects to evaluate each test case prioritization technique. The set of faults in the modified versions have been reported by our previous experiment [14], in which the faults are created following the methodology presented by Hutchins et al. [10]. Such a modified version setting was also adopted by the previous test case prioritization research studies (e.g., [5]).

TABLE VI. SUBJECTS AND THEIR DESCRIPTIVE STATISTICS.

Ref.	Applications	Modified Versions	Element	LOC	WSDL Spec.	WSDL Element	Used Versions
A	atm [3]	8	94	180	3	12	5
B	buybook [16]	7	153	532	3	14	5
C	dslservice [19]	8	50	123	3	20	5
D	gymlocker [3]	7	23	52	1	8	5
E	loanapproval	8	41	102	2	12	7
F	marketplace [3]	6	31	68	2	10	4
G	purchase [3]	7	41	125	2	10	4
H	triphandling [3]	9	94	170	4	20	8
	Total	60	527	1352	20	106	43

We use a random test case generation tool [14] to create random test suites for each subject based on WSDL specifications, XPath queries, and workflow logics of the original version of each subject. Each generated test suite ensures that all workflow branches, XRG branches, and WSDL elements of the original versions are covered at least once, as what the experiment from Mei et al. [14] did.

Specifically, we add a test case to a constructing test suite (initially empty) until the above-mentioned criterion has been fulfilled. This procedure is similar to the test suite construction from Elbaum et al. [5] and Hutchins et al. [10]. Moreover, the set of XML message received or generated by the original version of the subject in question over the test case is also recorded.

Using the above scheme, we successfully created 100 test suites for each subject that can detect at least one fault among the modified versions of the subject. Table VIII shows the maximum, average, and minimum sizes of the created test suites.

TABLE VIII. STATISTICS OF THE GENERATED TEST SUITE SIZE.

	Size								
Ref.	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**Avg.**
Max.	146	93	128	151	197	189	113	108	140.6
Avg.	95	43	56	80	155	103	82	80	86.8
Min.	29	12	16	19	50	30	19	27	25.3

(2) Effectiveness Measure

We choose to use *APFD* [5], a widely adopted metric in evaluating test case prioritization techniques (see [9][14] for example). It matches our objective to verify whether a technique supports service evolution.

Let T be a test suite containing n test cases, F be a set of m faults revealed by T, and TF_i be the first test case index in ordering T' of T that reveals fault i. The following equation gives the *APFD* value for a test suite T'.

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{nm} + \frac{1}{2n} \qquad \text{(E-5)}$$

(3) Procedure

Our tool applied C1−C4 and M1−M2 to prioritize each constructed test suite for each subject. C2 used the three levels of information *WE-i* (for i = I, II, and III) in turn. For C3, C4, M1, and M2, they used the three similarity metrics *W-i* (for i = I, II, and III) in turn. In essence, there are 16 (= 1 + 3 × 5) technique instances used in the experiment.

We executed the reordered test suite on each modified version of the subject and collected each TF_i value for *i*-th fault (if the *k*-th test case in the reordered test suite is the first test case that can detect the *i*-th fault, then TF_i is set to *k*). We finally calculated the *APFD* value of this reordered test suite (by E-5).

B. Data Analysis

To ease the view on the differences among techniques (especially techniques using various coverage/similarity metrics), we summarize the 25th, 50th (i.e., median), 75th percentiles and the standard deviations (the column of "SD") on all applications in Table VIII.

We have a number of interesting observations. Table IX shows that, at each of the 25th percentile, the median, and the 75th percentile *APFD* values, M1−M2 using *W*-II are more effective than C1, C2 using *WE*-II, and C3−C4 using *W*-II. The same is true for M1−M2 using *W*-III when comparing with the corresponding metric levels for C1−C4. The corresponding effectiveness of M1−M2 using *W*-I is however close to these of C1−C4. The results show that using runtime data as well as code coverage can improve the effectiveness of M1−M2 more often than these of C2−C4.

Moreover, M1 and M2 generally show an *upward* trend in effectiveness (in terms of the 25th percentile, the median, and the 75th percentile *APFD* values) and achieve smaller standard deviations when the similarity metrics changes from *W*-I to *W*-II, and from *W*-II to *W*-III. Surprisingly, the standard deviation on the APFD values achieved by C2 is even worse than that of random ordering.

TABLE IX. AVERAGE EFFECTIVENESS OF C1−C4 AND M1−M2 IN DIFFERENT PERCENTILES AND STANDARD DEVIATIONS

Technique	25th	50th	75th	SD
C1	0.7878	0.8659	0.9205	0.1227
C2 (*WE*-I)	0.8285	0.8863	0.9283	0.1178
C2 (*WE*-II)	0.7821	0.8325	0.8772	0.1520
C2 (*WE*-III)	0.7812	0.8354	0.8809	0.1540
C3 (*W*-I)	0.6464	0.7586	0.8468	0.1039
C3 (*W*-II)	0.5996	0.7088	0.7805	0.1034
C3 (*W*-III)	0.5859	0.7007	0.7872	0.0860
C4 (*W*-I)	0.7252	0.7841	0.9234	0.1084
C4 (*W*-II)	0.5836	0.7691	0.8751	0.0977
C4 (*W*-III)	0.6197	0.7582	0.8803	0.0765
M1 (*W*-I)	0.8393	0.8944	0.9377	0.1038
M1 (*W*-II)	0.8783	0.9173	0.9511	0.0993
M1 (*W*-III)	0.8669	0.9180	0.9506	0.0853
M2 (*W*-I)	0.8289	0.8807	0.9295	0.1119
M2 (*W*-II)	0.8718	0.9193	0.9478	0.1003
M2 (*W*-III)	0.8756	0.9174	0.9482	0.0862

C2−C4 are *increasingly less* effective as the coverage level *increases* from *W*-I to *W*-II or from *WE*-I to *WE*-II. Moreover, their corresponding standard deviations have no consistent trend. Initially, we are surprised by these two. Later, we realize that C2−C4 are insensitive to diversify the testing efforts to test different scenarios captured in the test suite. The result further indicates that these algorithms (best for C programs [5] for example) have shown up their problems when being adapted to use XML messages.

C3 and C4 are adapted from C2 by using the metrics that are also used by M1 and M2. Either C3 or C4 technique achieves better standard deviation than C2 in the experiment. It appears to suggest that using *W-i* can be more predictable than using *WE-i* in terms of APFD. Nonetheless, C3 and C4 are still less effective than M1 and M2 in terms of the 25th percentile, the median, and the 75th percentile *APFD* values. In the experiment, we find that neither C3 nor C4 is more effective than either C1 or C2. The empirical result further indicates that simply using *W-i* alone cannot achieve effective permutation of test suites in terms of APFD.

C. Threats to Validity

Our benchmarks are not large in scale, but are likely to be larger than the benchmarks used by Ni et al. [15]. Using more and larger real life benchmarks and their evolutions will strengthen the results obtained; unfortunately, we have not found such publicly released benchmarks.

We used APFD as the metric. Using other metrics such as HMFD [23] may produce different results. We have implemented our tool carefully and sampled the results of our techniques to validate them manually. We have used previously evaluated benchmarks and testing tools to conduct the experiment to minimize the chance of having an error. We have also compared the results of our techniques with the results of random ordering and three other peer techniques.

Our experiment has allowed test cases of the same web service to be executed in any order. The results obtained here may not be generalized to scenarios that there are casual constraints between test cases.

VI. RELATED WORK

We firstly review work on the unit testing and integration testing of services. Bartolini et al. [2] proposed to collect code coverage data for test runs from services so that service consumers can know the progress of their testing. Xu et al. [21] perturbed messages to test for the robustness of web services. Their techniques are useful to support our technique using the *W*-I metric.

Zhang [24] proposed an agent-based approach to selecting reliable web services components efficiently. Zhai et al. [22] kept a blacklist of services that failures have been revealed in regression testing to improve the cost-

effectiveness. Either work aims at reducing the number of service invocations for testing. Our techniques have not explored this area. Martin et al. [12] perturbed web-service requests to test for the robustness of web services. Their approach suffers from the functional test oracle problem needed for regression test. Our technique does not modify service messages, and use the regression test oracle produced by the previous round of regression test. Bai et al. [1] proposed to partition scenarios based on the ontology associated with services. They used the logical relationship to group test scenarios; whereas, our techniques use test cases similarity but not semantic relations to group test cases.

Hou et al. [9] proposed to consider the constraint on the number of times that a web service may be invoked in the test case prioritization for the regression testing of service-centric applications. However, how a service applies WSDL specifications to XML documents has not been studied. Chen et al. [4] applied dependence analysis to prioritize test cases according to the amount of modification-affected elements per test case. Mei et al. developed a family of test case prioritization techniques atop a multilevel coverage model [14]. They also proposed techniques using tags embedded in XML messages [13]. The use of runtime artifacts (XML message) have been explored by Mei et al. [14]. Zhai et al. [22][23] proposed prioritize test cases for the testing of location-based services using the locational information in the input and output of the test cases. Li et al. [11] proposed to use the extensible BPEL flow graph to model the changes of composite service in terms of processes, bindings, and interfaces. They further performed control flow analysis based on such graphs to identify the changes within the composite services. All the above reviewed techniques have only considered test cases one by one or the whole test suite without any breakdown. They have not systematically considered test case pairs.

VII. Conclusion

Pairwise selection is a fundamental strategy to compare elements in a finite set. Using the notion of structural similarity is attractive to spot the differences in semi-structural artifacts like XML documents. In this paper, we have proposed test case prioritization techniques based on this strategy for the regression testing of web services. We have empirically demonstrated that our techniques are feasible, and they can be more effective than existing techniques or random ordering in terms of APFD.

In terms of pairwise comparison, our techniques give an exact solution, but are NP-complete. A further optimization such as the use of an approximation approach can be further developed. Another extension is to study the *n*-way selection strategy. Our work only deals with a part of the self-adaptation cycle needed for web service evolution. A more comprehensive study that deals with self-adaptation is necessary.

References

[1] X. Bai, S. Lee, W.-T. Tsai, and Y. Chen. Ontology-Based Test Modeling and Partition Testing of Web Services. In *Proceeding of ICWS 2008*, pages 465–472, 2008.

[2] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti. Whitening SOA testing. In *Proceedings of FSE 2009*, pages 161–170, 2009.

[3] BPEL Repository, IBM, 2006. Available at http://www.alphaworks.ibm.com/tech/bpelrepository.

[4] L. Chen, Z. Wang, L. Xu, H. Lu, and B. Xu. Test case prioritization for web service regression testing. In *Proceedings of SOSE 2010*, pages 173–178, 2010.

[5] S. Elbaum, A. G. Malishevsky, G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE TSE*, 28(2): 159–182. 2002.

[6] X. Fu, T. Bultan, and J. Su. Model checking XML manipulating software. In *Proceedings of ISSTA 2004*, pages 252–262. 2004.

[7] M. Garofalakis and A. Kumar. XML stream processing using tree-edit distance embeddings. *ACM TODS*, 30(1): 279–332. 2005.

[8] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML Joins. In *Proceedings of SIGMOD 2002*, pages 287–298, 2002.

[9] S. Hou, L. Zhang, T. Xie, and J. Sun. Quota-constraint test-case prioritization for regression testing of service-centric systems. In *Proceedings of ICSM 2008*, pages 257–266, 2008.

[10] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of ICSE 1994*, pages 191–200, 1994.

[11] B. Li, D. Qiu, H. Leung, D. Wang, Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph. In *Journal of Systems and Software*, 85(6):1300–1324, 2012.

[12] E. Martin, S. Basu, and T. Xie. Automated testing and response analysis of web services. In *Proceedings of ICWS 2007*, pages 647–654. 2007.

[13] L. Mei, W.K. Chan, T.H. Tse, and R.G. Merkel. XML-manipulating test case prioritization for XML-manipulating services. In *Journal of Systems and Software*, 84(4):603–619, 2011.

[14] L. Mei, Z. Zhang, W.K. Chan, and T.H. Tse. Test case prioritization for regression testing of service-oriented business applications. In *Proceedings of WWW 2009*, pages 901–910. 2009.

[15] Y. Ni, S.S. Hou, L. Zhang, J. Zhu, Z. Li, Q. Lan, H. Mei, and J.S. Sun, Effective Message-Sequence Generation for Testing BPEL Programs. To appear in *IEEE Transactions on Services Computing*.

[16] Oracle BPEL Process Manager. Oracle Technology Network. Available at http://www.oracle.com/technology/products/ias/bpel/.

[17] W3C. Web Services Description Language (WSDL) 1.1. 2001. Available at http://www.w3.org/TR/wsdl.

[18] Web Services Business Process Execution Language Version 2.0. 2007. Available at http://www.oasis-open.org/committees/wsbpel/.

[19] Web Services Invocation Framework: DSL provider sample application. Apache Software Foundation. 2006. Available at http://ws.apache.org/wsif/wsif_samples/index.html.

[20] World Wide Web Consortium. XML Path Language (XPath) Recommendation. 2007. Available at http://www.w3.org/TR/xpath20/.

[21] W.Xu, J.Offutt, and J. Luo. Testing Web Services by XML Perturbation. In *Proceedings of ISSRE 2005*, pages 257–266, 2005.

[22] K. Zhai, B. Jiang, W.K. Chan, and T.H. Tse. Taking advantages of service selection: a study on the testing of location-based web services through test case prioritization. In *Proceeding of ICWS 2010,* pages 211–218, 2010.

[23] K. Zhai, B. Jiang, and W.K. Chan. Prioritizing test cases for regression testing of location-based services: metrics, techniques and case study. To appear in *IEEE Transactions on Services Computing*.

[24] J. Zhang. An approach to facilitate reliability testing of web services components. In *Proceedings of ISSRE 2004*, pages 210–218, 2004.