A Subsumption Hierarchy of Test Case Prioritization for Composite Services [†]

Lijun Mei, Yan Cai, Changjiang Jia, Bo Jiang, W.K. Chan, Zhenyu Zhang, and T.H. Tse

Abstract—Many composite workflow services utilize non-imperative XML technologies such as WSDL, XPath, XML schema, and XML messages. Regression testing should assure the services against regression faults that appear in both the workflows and these artifacts. In this paper, we propose a refinement-oriented level-exploration strategy and a multilevel coverage model that captures progressively the coverage of different types of artifacts by the test cases. We show that by using them, the test case prioritization techniques initialized on top of existing greedy-based test case prioritization strategy form a subsumption hierarchy such that a technique can produce more test suite permutations than a technique that subsumes it. Our experimental study of a model instance shows that a technique generally achieves a higher fault detection rate than a subsumed technique, which validates that the proposed hierarchy and model have the potential to improve the cost-effectiveness of test case prioritization techniques.

Index Terms—Test case prioritization, service orientation, XPath, WSDL, XML messages.

1 INTRODUCTION

In a composite business service (typically specified in *Web Services Business Process Execution Language* (*WS-BPEL* or simply *BPEL*) [41], a business process may invoke external web services to execute the required functionality by matching the contents of XML messages with schemas in *Web Services Description Language* (*WSDL*) specifications [42].

A revised XPath expression [45] to support a particular workflow step may extract wrong sets of contents from XML messages for some other workflow steps [29]. Similarly, a revised XML schema embedded in a WSDL specification that includes an additional field may mistakenly cause the XPath expression to match some extra query paths in an XML message. Modifying the workflow logics may route mismatched XML messages along workflow steps different from the previous workable ways unintentionally.

Regression testing [47] aims at detecting potential faults caused by software changes, and is the de facto approach to assuring revised applications [25], [37]. It reruns the existing test cases to assure that no previously working function has failed as a result of the modification [25]. To reduce costs, it is desirable to detect failures as soon as possible. Test Case Prioritization (TCP) [19], [28], [33], [48] is an important aspect in regression testing [19], [47]. It schedules the test cases in a regression test suite with a view to maximizing certain objectives (such as revealing faults earlier), which helps reduce the cost of maintenance.

A composite service has multiple levels of details. Take WS-BPEL services as an example. Different services may have different workflow steps. Every workflow step in a WS-BPEL service may declare or refer to an XPath query. Each XPath query incorporates a layer of "conceptual" branch decisions into the workflow step (such as deciding whether a hotel room can be selected by following the XML document structure). It associated with a WSDL specification [29], each "conceptual" branch decision *becomes* a set of concrete tags, which is *enumerable* and varies according to the actual WSDL specification available. One single such concrete tag in an XML message suffices to make the corresponding "conceptual" branch decision to be evaluated as true (such as successfully finding a hotel room according to a particular XML tag). Of course, the specific tag matching the enumerable set may vary from one XML message to another. For instance, by viewing inside-out, the level of details changes from *workflows* to *XPath queries*, then to *WSDL specifications*, *specific tag usages*, and finally to *XML messages*. Similarly, viewing from outside-in produces another sequence of artifacts.

We have modeled such a layer of enumerable sets of tags as an XPath Rewriting Graph (XRG) [29]. To specify a specific tag in an enumerate set of tags of XRG to be referred to, we defined a corresponding XRG pattern [30].

In [33], we have proposed the *first* multilevel coverage model that considers the first three levels of details and formulated the *first* subsumption hierarchy of TCP techniques. We have also shown that some *level-exploration* strategies (such as the *summation* strategy) fail to result in techniques that form a hierarchy based on our subsumption relation.

In this paper, we extend the multilevel coverage model in [33] to cover the last two levels of details presented above. To explore different levels of details while preserving the subsumption relation, we propose *Refinement-Oriented Level-Exploration* (ROLE). ROLE refers to the next unused level of

- *Lijun Mei is with IBM Research – China, Beijing, China.*
- *Yan Cai, Changjiang Jia and W.K. Chan (contact author) are with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong. E-mails: {yancai2, cjjia2}@gapps.cityu.edu.hk, and wkchan@cityu.edu.hk*
- *Bo Jiang is with the School of Computer Science and Engineering, Beihang University, Beijing, China. E-mail: jiangbo@buaa.edu.cn.*
- *Zhenyu Zhang is with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China. E-mail: zhangzy@ios.ac.cn.*
- *T.H. Tse is with the Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong. E-mail: thtse@cs.hku.hk.*

This research is supported in part by the General Research Fund of the Research Grants Council of Hong Kong (project nos 111410, 716612, and 717811), the National Natural Science Foundation of China (project nos 61202077 and *61379045), the National Science and Technology Major Project of China (grant no. 2012ZX01039-004), National Key Basic Research Program of China (project no 2014CB340702), and IBM UR Project (project no 201309009).*

[†] *A preliminary version [33] of this paper was published in the Proceedings of the World Wide Web Conference (WWW '09). The preliminary version was also one of the six best paper nominees of WWW '09.*

detail only if using the current level of detail cannot help a *prioritization strategy* (e.g., the additional strategy [19]) select a test case. To make our work more focused, this paper does not consider other level-exploration strategies such as the *summation strategy*, which has no subsumption relation with TCP techniques.

Based on the above model, ROLE, and prioritization strategies, we formulate a subsumption relation and a provable hierarchy of TCP techniques, including eight ROLE-enriched techniques. We also verify our work with experiments on fault detection rates using a benchmark suite of eight WS-BPEL subjects as well as a case study on a real-world service-based application. The result shows that techniques located higher in the hierarchy are more effective.

The main contribution of this paper, together with its preliminary version [33], is threefold. (i) We propose the first multilevel coverage model and refinement-oriented level-exploration strategy for TCP techniques. (ii) We show that the resultant TCP techniques form a subsumption hierarchy. To the best of our knowledge, this is the first logical hierarchy to relate TCP techniques in the public literature. The hierarchy concretely demonstrates that *some but not all* TCP techniques can be compared logically. (iii) We report an experimental study that validates the fault detection rates of the techniques in the hierarchy.

The rest of the paper is organized as follows: Section 2 gives the preliminaries. Section 3 outlines a motivating example. Section 4 presents our hierarchy of prioritization techniques, followed by its validation in Section 5. Section 6 discusses related work. Section 7 concludes the paper.

2 Preliminaries

2.1 TCP Metrics and Control Techniques

TCP techniques can be designed to achieve certain goals (such as maximizing the coverage rate) in regression testing of the next revised versions. The TCP problem, adapted from [19], is specified 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 suit $T' \in PT$ such that $\forall T'' \in PT, f(T') \geq f(T'')$.

The metrics *Average Percentage of Faults Detected* (*APFD*) [19], *average Relative Position* (*RP*) [39], and *Harmonic Mean of rate of Fault Detection* (*HMFD*) [48] each evaluates TCP techniques from the perspective of the rate of revealing faults. A higher APFD value indicates a higher (or better) fault detection rate, whereas a lower RP or HMFD value indicates a higher fault detection rate. In this paper, the function f maps every permutation T' in PT to the APFD, RP, or HMFD value of T'. Each metric value ranges between 0 and 1. More specifically, 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 index of the first test case in the reordered test suite T' that reveals fault i. The APFD and HMFD values of T' are computed as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n}$$

$$HMFD = \frac{m}{\frac{1}{TF_1} + \frac{1}{TF_2} + \cdots + \frac{1}{TF_m}}$$

Let $P(TF_i, i)$ be the probability that the first failed test case caused by fault i is in position TF_i. The RP value of fault i is computed as follows:

$$RP(i) = \frac{\sum_{TF_i=1}^{n} TF_i . P(TF_i, i)}{n}$$

We will compare TCP techniques in Section 5 with two control techniques, which we briefly summarize here:

C1: Random ordering [19] randomly orders the test cases in a test suite T.

C2: Optimal prioritization [19]. Given a program P and a set of known faults in P, if we know the specific test cases in a test suite T that expose specific faults in P, then an optimal ordering of the test cases is the one that maximizes the fault detection rate of T. C2 approximates the optimal case [19].

2.2 XPath and XPath Query Model

2.2.1 XPath

We adopt the definition and notation of XPath expression from [34]. Let Σ be the element labels and attribute labels that may appear in XML documents. Figure 1 summarizes the semantics of XPath expressions taken from [34], where an XPath expression is defined by the following grammar:

$$q \rightarrow n \mid * \mid . \mid q / q \mid q // q \mid q[q]$$

where n is any label in Σ, * denotes a wildcard label, and "." (the dot operator) denotes the current node. The constructors / and // mean child and descendant navigations, while the square brackets [ ] enclose a predicate. The set of all trees are denoted by T_Σ, and each tree represents an XML document satisfying an XML schema (denoted by Ω). For a tree $t \in T_\Sigma$, an *XPath query* $q(t)$ is a query on t using an XPath expression q, and returns a set of nodes of t. Following [34], we denote the sets of nodes and edges by use *NODES*(t) and *EDGES*(t), respectively, and denote the label of node x by *LABEL*(x). We also use *EDGES**(t) to denote the Kleene closure of *EDGES*(t).

left hand side		*right hand side*	***Rule***
n(x)	=	{y | (x, y)∈*EDGES*(t), *LABEL*(y) = n}	… 1
*(x)	=	{y | (x, y) ∈*EDGES*(t)}	… 2
.(x)	=	{x}	… 3
(q_1/q_2)(x)	=	{z | y∈q_1(x), z∈q_2(y)}	… 4
(q_1//q_2)(x)	=	{z|y∈q_1(x), (y, u)∈*EDGES**(t), z∈q_2(u)}	… 5
(q_1[q_2])(x)	=	{y | y∈q_1(x), q_2(y)≠Ø}	… 6

Figure 1. Semantics of XPath expressions (from [34]).

For Rule 5, apart from the sub-terms q_1(x) and q_2(u), there is also a sub-term {u | (y, u)∈*EDGES**(t)}, which means all the nodes u in t are reachable from y. For ease of specifying the XPath query model in this paper, we define Rule 7 as

//(x) = {y | (x, y)∈*EDGES**(t)} … 7

We show an XML message on the right of Figure 2. It contains two parts: a specific hotel room under the path /hotel, and a list of hotel rooms under the path

/hotel/hotelList/hotel. Queries 1, 2, and 3 on the left of the same figure are XPath queries each enclosed within a dashed-and-dotted rectangle. They search for hotel name(s) under the first path (/hotel/name/), both paths (//name/), and the second path (/hotel/hotelList/hotel/name/), respectively.

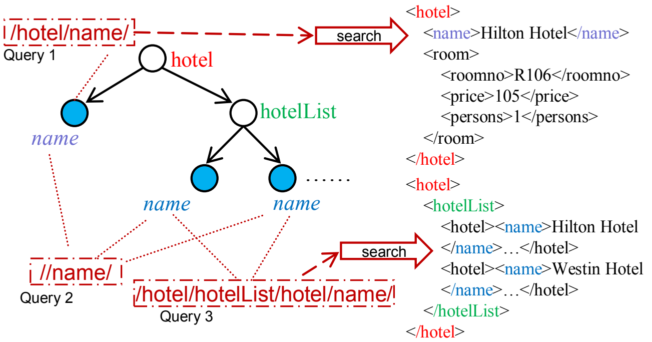
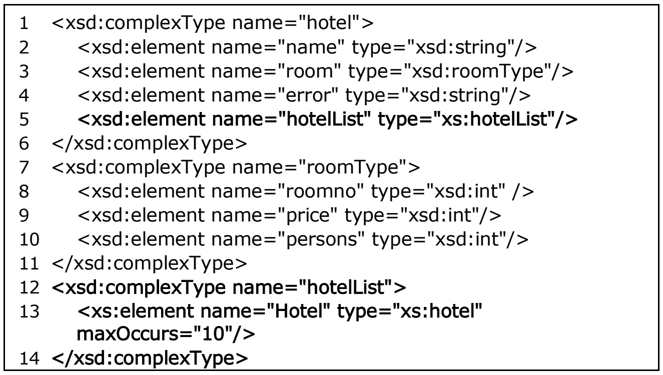
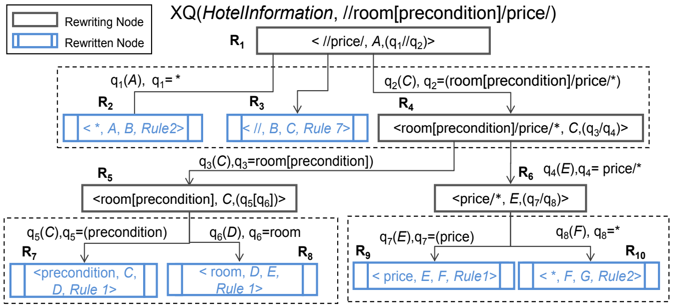


Figure 2. Effects of the structure of an XML message on different XPath queries to locate hotels using hotel name.



```
<xsd:complexType name="hotel">
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="room" type="xsd:roomType"/>
    <xsd:element name="error" type="xsd:string"/>
    <xsd:element name="hotelList" type="xs:hotelList"/>
</xsd:complexType>
<xsd:complexType name="roomType">
    <xsd:element name="roomno" type="xsd:int" />
    <xsd:element name="price" type="xsd:int"/>
    <xsd:element name="persons" type="xsd:int"/>
</xsd:complexType>
<xsd:complexType name="hotelList">
    <xs:element name="Hotel" type="xs:hotel"
    maxOccurs="10"/>
</xsd:complexType>
```

Figure 3. Excerpt from WSDL document: XML schema of *hotel*.


Rewriting Node
Rewritten Node
XQ(*HotelInformation*, //room[precondition]/price/)
R1
< //price/, *A*,(q1//q2)>
q1(*A*), q1= *
q2(*C*), q2=(room[precondition]/price/*)
R2
< *, *A*, *B*, *Rule2*>
R3
< //, *B*, *C*, *Rule 7*>
R4
<room[precondition]/price/*, *C*,(q3/q4)>
q3(*C*),q3=room[precondition])
R5
<room[precondition], *C*,(q5[q6])>
R6
q4(*E*),q4= price/*
<price/*, *E*,(q7/q8)>
q5(*C*),q5=(precondition)
q6(*D*), q6=room
R7
<precondition, *C*, *D*, *Rule 1*>
R8
< room, *D*, *E*, *Rule 1*>
q7(*E*),q7=(price)
q8(*F*), q8=*
R9
< price, *E*, *F*, *Rule1*>
R10
< *, *F*, *G*, *Rule2*>

precondition: (@price≤‘Price’ and @persons=’Num’)

A = {*ROOT*} *B* = {*hotel*} *C* = {*name*, *room*, *roomno*, *price*, *persons*, *error*} *D* = {*room*} *E* = {*room*}

F = {*price*} *G* = {*g* | *g* is the *price* value}

Rule 1: {y|(x, y)∈*EDGES*(t), *LABEL*(*y*) = n} *Rule 2*: {y|(x, y)∈*EDGES* (t)}

Rule 4: {z| y∈q1(x), z∈q2(y)} *Rule 5*: {z|y∈q1(x), (y, u)∈$EDGES^*$(t), z∈q2(u) }

Rule 6: {y| y∈q1(x), q2(y) ≠ Ø } *Rule 7*: {y|(x, y)∈*EDGES**(t)}

Figure 4. Example of XPath Rewriting Graph (XRG).

2.2.2 *XPath query model*

An *XPath Rewriting Graph* (*XRG*) [29] represents potential scenarios of content selections by XML messages. XRG is built on the semantics rules for an XPath expression q presented in Figure 1. It treats every such rule as a “left-to-right” rewriting rule in the spirit of term rewriting [13] to transform the query q. In essence, given a query q and a document model Ω, the algorithm in [29] creates a node for q, locates a rule whose left hand side matches the query q, and then creates a set of XRG nodes pointed to by the former node. Specifically, if the matching rule has *d* sub-terms on the right hand side of the rule (such as *d* = 3 for Rule 5), it creates *d* XRG nodes, one for each sub-term. The types of these *d* nodes depend on whether the sub-term includes any query (such as q_2 in Rule 5). XRG nodes with and without query are referred to as *rewriting nodes* and *rewritten nodes*, respectively. Each rewritten node will not be subject to further rewriting.

The query (such as q_2 in Rule 5) in each of such rewriting nodes is then used to generate a new set of rewriting nodes and rewritten nodes. If any newly generated XRG node is the same as an existing XRG node that is previously generated, the algorithm just reuses the existing XRG node (to create a fixed point) and discards this newly generated one.

A rewriting node is represented by a triple ⟨*q*, L^c, *rule*⟩ and a rewritten node is represented by a quadruple ⟨*q*, L^c, L^n, *S*⟩. In such an XRG node, *q* is an XPath expression. L^c and L^n are sets of nodes in the document model Ω (that is, L^c, $L^n \subseteq$ *NODES*(Ω)). They represent the sets of nodes reachable by *q* based on the semantics defined in Figure 1: L^c is the set of nodes in Ω located by the rewriting step pointing to it, and L^n is the set of nodes in Ω that can be located by *q* starting from at least one node in L^c. *S* is the rewritten form of *q* based on the matching rule. Since Figure 1 shows the semantic rules of XPath expression in set notion, *S* is also expressed in set notation. Also, *rule* denotes the left hand side of the matching rewriting rule (in Figure 1). The set L^c in the rewriting node for the inputted query *q* is singleton and contains the unique root node ROOT of the schema Ω.

In the spirit of data flow analysis, we further consider any variable generated in XRG as a *variable definition*, and the use of a variable provided by a preceding node as a *variable usage*. Such variables (e.g., L^c and L^n in rewritten nodes) are conceptual in nature and are not program variables because they *never* appear in an implemented program. We thus call them *conceptual variables*. For example, the XPath query in Figure 4 returns the conceptual variable *g* at node R_{10}.

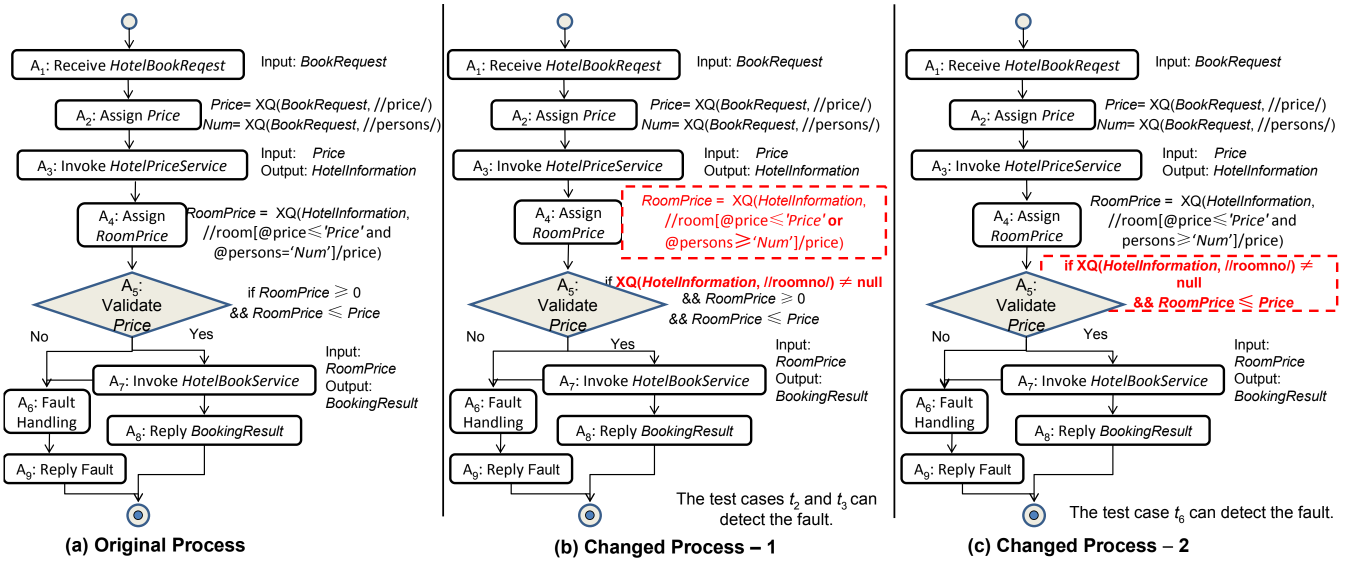
Definition 1. ***XPath Rewriting Graph (XRG).*** An XRG for an XPath query is a 5-tuple ⟨*q*, Ω, N_x, E_x, V_x⟩ such that

(1) *q* is an XPath expression for the XPath query; Ω is an XML schema that describes the XML document to be queried on.

(2) N_x is a set of rewriting nodes and rewritten nodes; V_x is a set of conceptual variables defined on the nodes in N_x.

(3) E_x is a set of edges, each representing the transition between two nodes. Each edge is denoted by a tuple (s_c, s_n), where s_c, s_n ∈ N_x and s_n rewrites s_c.

Let us show an example of an XRG. Suppose, during the reservation of a hotel room (see the full example in Section 3), the booking information (in XML format) is kept in a BPEL variable *HotelInformation*. Figure 3 shows a simplified XML schema *hotel* for *HotelInformation*. A room has three attributes


A1: Receive HotelBookReqest
Input: BookRequest
A2: Assign Price
Price= XQ(BookRequest, //price/)
Num= XQ(BookRequest, //persons/)
A3: Invoke HotelPriceService
Input: Price
Output: HotelInformation
A4: Assign RoomPrice
RoomPrice = XQ(HotelInformation, //room[@price≤'Price' and @persons='Num']/price)
A5: Validate Price
if RoomPrice ≥ 0 && RoomPrice ≤ Price
No
Yes
A6: Fault Handling
A7: Invoke HotelBookService
Input: RoomPrice
Output: BookingResult
A8: Reply BookingResult
A9: Reply Fault
RoomPrice = XQ(HotelInformation, //room[@price≤'Price' or @persons≥'Num']/price)
if XQ(HotelInformation, //roomno/) ≠ null && RoomPrice ≥ 0 && RoomPrice ≤ Price
The test cases t2 and t3 can detect the fault.
RoomPrice = XQ(HotelInformation, //room[@price≤'Price' and persons≥'Num']/price)
if XQ(HotelInformation, //roomno/) ≠ null && RoomPrice ≤ Price
The test case t6 can detect the fault.

(a) Original Process **(b) Changed Process – 1** **(c) Changed Process – 2**

Figure 5. Activity diagrams of a WS-BPEL application.

(lines 8–10): *roomno*, *price*, and *persons* (indicating the maximum number of persons allowed). Consider an XPath query on *HotelInformation*, denoted by XQ(*HotelInformation*, q), where q is //room[@price ≤ 'Price' and @persons = 'Num']/price/. Informally, q finds a room within the requested price that can accommodate the requested number of persons. The corresponding XRG is shown in Figure 4.

We show the first rewriting step to illustrate how an XRG is computed. XQ(*HotelInformation*, q) is identified by Rule 5 as $q_1//q_2$ where q_1 = *, and q_2 = room[precondition$_1$]/price/*), in which precondition$_1$ is "@price ≤ 'Price' and @persons = 'Num' ". Since there are queries on the right hand side of Rule 5, a rewriting node R_1 is generated. The middle sub-term on the right hand side of Rule 5 matches Rule 7, and hence R_3 is generated. Next, the algorithm recursively processes the two queries q_1 and q_2. The query q_1 matches Rule 2, and the right hand side of Rule 2 does not contain any query and only contains one sub-term, and thus one rewritten node R_2 is generated. The query q_2 matches Rule 4, which contains other queries on its right hand side. The rewriting node R_4 is thus generated. The remaining rewriting steps are similar.

After constructing the XRG, we obtain a conceptual path *p* that models a logical computation of an XPath query via an *inorder traversal* of the XRG with all the rewritten nodes R_2, R_3, R_7, R_8, R_9, and R_{10} in sequence. Such a conceptual path *p* contains *implicit predicates*, each of which decides on a legitimate branch (called an *XRG branch*) to be taken. For instance, if an XML document does not contain any element that match the set *B* in R_2, *B* will be empty. This will result in no more applicable rewriting. A succeeding rewritten node will appear in a conceptual path only if its preceding rewritten node provides a non-empty set of L^n. A branch decision can be modeled by whether L^n in a node is empty.

We revisit the notion of *XRG pattern*: In general, a conceptual variable *z* may contain multiple tags of an XML schema. As long as the corresponding XML message matches at least one tag in the tag set of *z*, this variable does not distinguish which subset of tags having been selected. Therefore, we define an *instantiation* of *z* as assigning a concrete value to *z*.

To differentiate the usages of these tags in the same query path, we introduced XRG patterns in [30]. We will give examples of XRG patterns in Section 3 (see Table 4 also).

Definition 2. ***XRG Pattern*** [30]**.** For any given XRG $r = \langle q, \Omega, N_x, E_x, V_x \rangle$, an *XRG pattern* $\xi(r)$ is an instantiation of *r* such that (i) a tag t_i is assigned to the *i*-th variable ($\in V_x$) in a conceptual path *p* based on the definition order of the variables, and (ii) t_i must be used (in terms of data flow associations) by a subsequent rewritten node $n \in N_x$ to locate t_{i+1} in the path *p*.

3 Motivating Example

3.1 Modification Example

We adapt the *HotelBooking* process in *TripHandling* [1] to motivate our work. *HotelBooking* offers hotel booking services. Since the actual BPEL code in XML format is quite lengthy, we use an activity diagram to depict the business process as shown in Figure 5(a).

We represent a workflow step (numbered as A_i for *i* from 1 to 9) and a transition between two steps by a node and a link, respectively. We annotate nodes with data extracted from the process, such as the input/output parameters of the activities and XPath queries. The process in Figure 5(a) is:

(a) A_1 receives a user's hotel booking request, and stores it in the variable *BookRequest*.

(b) A_2 extracts the inputted room price and number of persons via the XPath queries //price/ and //persons/ from *BookRequest*, and stores them in the variables *Price* and *Num*, respectively.

(c) A_3 invokes an external service *HotelPriceService* to find available hotel rooms with prices within budget (not exceeding *Price*), and keeps the result in *HotelInformation* (with its schema as defined in Figure 3).

TABLE 1
WORKFLOW BRANCH COVERAGE FOR T_1 TO T_8

Branch	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
$\langle A_1, A_2\rangle$	•	•	•	•	•	•	•	•
$\langle A_2, A_3\rangle$	•	•	•	•	•	•	•	•
$\langle A_3, A_4\rangle$	•	•	•	•	•	•	•	•
$\langle A_4, A_5\rangle$	•	•	•	•	•	•	•	•
$\langle A_5, A_6\rangle$		•	•		•	•	•	•
$\langle A_5, A_7\rangle$	•			•				
$\langle A_7, A_6\rangle$								
$\langle A_7, A_8\rangle$	•			•				
$\langle A_6, A_9\rangle$		•	•		•	•	•	•
Total	6	6	6	6	6	6	6	6

TABLE 2
XRG BRANCH COVERAGE FOR T_1 TO T_8

XRG branch	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
$\langle R_2, R_3\rangle$	•	•	•	•	•	•	•	•
$\langle R_2, A_4\rangle$								
$\langle R_3, R_7\rangle$	•	•	•	•		•	•	•
$\langle R_3, A_4\rangle$					•			
$\langle R_7, R_8\rangle$	•	•	•	•		•	•	•
$\langle R_7, A_4\rangle$								
$\langle R_8, R_9\rangle$	•			•		•		
$\langle R_8, A_4\rangle$		•	•				•	•
$\langle R_9, R_{10}\rangle$	•			•		•		
$\langle R_9, A_4\rangle$								
Total	5	4	4	5	2	5	4	4

TABLE 3
STATISTICS OF WSDL ELEMENTS AND XML MESSAGES FOR T_1 TO T_8

XML schema	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Hotel	•	•	•	•	•	•	•	•
HotelList	•	•					•	
Name	•	•	•				•	•
Room	•	•	•	•		•	•	•
Roomno	•	•	•	•			•	•
Price	•	•	•	•		•	•	•
Persons	•	•	•	•		•	•	•
Error						•		
Subtotal (WSDL)	7	7	6	5	1	5	7	6
val(name)	•	•	•				•	•
val(roomno)	•	•	•					
val(price)	•	•	•	•		•		
val(persons)	•	•	•	•		•		
val(error)						•		
Total (XML)	11	11	10	7	1	8	8	7

TABLE 4
XRG PATTERN COVERAGE FOR T_1 TO T_8

Index	**XRG pattern**	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
x1	/hotel/hotelList/hotel/room[predication]/price	•							
x2	/hotel/room[prediction]/price				•		•		
	Total	1	0	0	1	0	1	0	0

(d) A_4 assigns *RoomPrice* using the price obtained from the query //room[@price≤’Price’ and @persons= ’Num’]/price/.

(e) A_5 further verifies locally that the price in *HotelInformation* should not exceed the inputted price (the variable *Price*).

(f) If the verification passes, A_7 will invoke *HotelBookService* to book a room, and A_8 returns the result to the customer.

(g) If *RoomPrice* is erroneous or *HotelBookService* (A_7) produces a failure, A_6 will invoke a fault handler, and A_9 will then return the fault.

We present two changes in Figure 5(b) and (c) that may result in integration failures. Suppose John decides that node A_4 in Figure 5(a) should be changed to node A_4 in Figure 5(b). That is, he attempts to allow customers to select any available room for the requested number of persons. However, he wrongly changes the precondition in the XPath query (namely, changing "and" to "or"), which introduces a regression fault. Further, suppose that another engineer Lucy wants to correct this fault. She fixes node A_4 in Figure 5(b) by changing the precondition in the XPath query (namely, changing "or" to "and"). But she considers the precondition in node A_5 to be redundant (that is, no need to require $RoomPrice \geq 0$). She therefore changes node A_5 in Figure 5(b) to node A_5 in Figure 5(c), and forgets to handle another potential scenario ($RoomPrice < 0$). Her change thus introduces a regression fault into the original program.

3.2 Sample Test Cases

The inputs to the WS-BPEL service are XML documents. We use ⟨*Price*, *Num*⟩ to denote the *BookRequest* document at node A_1, where *Price* is the value of the price variable and *Num* is value of the variable denoting the number of persons. (For brevity, we do not introduce the XML schema that defines *BookRequest*.) We use eight test cases (t_1 to t_8) for illustration:

	⟨*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⟩
Test case 7 (t_7):	⟨180, 5⟩	Test case 8 (t_8):	⟨160, 4⟩

```
<hotel>
  <hotelList>
  <hotel><name>Hilton</name>
  <room>
     <roomno>R106</roomno>
     <price>105</price>
     <persons>1</persons>
  </room>
  </hotel>
  <hotel>
  <name>Westin</name>
  <room>
     <roomno>R101</roomno>
     <price>150</price>
     <persons>3</persons>
  </room></hotel>
  </hotelList>
</hotel>
```

for Test Case t_1

```
<hotel>
  <hotelList>
  <hotel><name>Hilton</name>
  <room>
     <roomno>R106</roomno>
     <price>105</price>
     <persons>1<persons>
  </room>
  </hotel>
  <hotel>
  <name>Westin</name>
  <room>
     <roomno>R101</roomno>
     <price>150</price>
     <persons>3</persons>
  </room></hotel>
  </hotelList>
</hotel>
```

for Test Cases t_2

```
<hotel>
  <name>Hilton</name>
  <room>
     <roomno>R106</roomno>
     <price>105</price>
     <persons>1</persons>
  </room>
</hotel >
```

for Test Cases t_3

```
<hotel>
  <room>
     <roomno></roomno>
     <price>100</price>
     <persons>2</persons>
  </room>
</hotel >
```

for Test Case t_4

```
<hotel>
</hotel>
```

for Test Case t_5

```
<hotel>
 <room>
    <price>-1</price>
    <persons>1</persons>
  </room>
  <error>InvalidPrice</error>
</hotel >
```

for Test Case t_6

```
<hotel>
  <hotelList>
  <hotel><name>Hilton</name>
        <room><roomno></roomno><price></price>
        <persons></persons></room></hotel>
  <hotel><name>Westin</name>
        <room><roomno></roomno><price></price>
        <persons></persons></room></hotel>
  </hotelList>
</hotel>
```

for Test Case t_7

```
<hotel>
  <name>Hilton</name>
  <room><roomno></roomno>
  <price></price>
  <persons></persons></room>
</hotel>
```

for Test Case t_8

Figure 6. XML messages for XQ(HotelInformation, //room[@price ≤ 'Price' and @persons = 'Num']/price/) for different test cases.

Figure 6 shows the XML messages used by t_1 to t_8 at node A_4. Each of the first two includes one single room and one triple room. The 3rd contains one single room. The 4th includes the price of one room without the room number. The 6th is an error message. The rest contain no room data.

The test oracle for the example is the booking result, namely, successful booking, failed booking, or error message. When executing the process in Figure 5(b), t_1 extracts a correct price; both t_2 and t_3 extract the price of 105 for a single room, but they actually need to book a double room and a family room, respectively; t_4 extracts a price that it should not extract, and it cannot book any room; each of t_5 to t_8 does not extract any price value. Both t_2 and t_3 detect the fault shown in Figure 5(b). Similarly, executing the process shown in Figure 5(c), t_1 extracts the correct prices; t_2, t_3, t_5, t_7, and t_8 do not extract any price; t_4 extracts a price that it should not extract, and thus cannot book any room as expected; and t_6 extracts a price of –1 while it should not extract any price, and leads to an error message. Only t_6 can detect the fault shown in Figure 5(c).

3.3 Baseline

Table 1 shows the workflow branch coverage of t_1 to t_8 against the *original* process of *HotelBooking* in Figure 5(a). We use a "•" to denote an item covered by a test case in Table 1 (as well as in Tables 2, 3, and 4 and Figure 7). As shown in Table 1, the test cases t_1 to t_8 cover the same number of workflow branches. The *total-branch* prioritization technique (that is, the *total* prioritization strategy utilizing *branch* coverage data) thus behaves like random ordering [19]. If we apply this technique to the coverage of workflow branches only, $t_1 \rightarrow t_5 \rightarrow t_4 \rightarrow t_7 \rightarrow t_8 \rightarrow t_2 \rightarrow t_3 \rightarrow t_6$ is one of the *least effective* orderings. Its APFD value is $1 - (6 + 8) \div (8 \times 2) + 1 \div (2 \times 8) = 0.1875$. We observe that using the coverage data of workflow branches is still nondeterministic in the selection of t_1 to t_8, and finding ways to eliminate such ineffective orderings will help increase the effectiveness of fault detection. This further motivates our work.

4 A Subsumption Hierarchy

In this section, we present the key aspects of TCP techniques in our model followed by a subsumption hierarchy.

4.1 Multilevel Coverage Model

We propose a *multilevel coverage model* to facilitate the application of level-exploration strategies. We use the sequence of artifacts stated in the example in Section 1 to illustrate our model. Our model is general, however. We emphasize that the order of coverage data used in an individual level in the model is independent of both the notion of ROLE and the subsumption relation. Our model can be initialized with other sequences of the same or different coverage datasets.

A coverage model for a service-oriented workflow application P is a 6-tuple $\langle T, \Pi_\alpha, \Pi_\beta, \Pi_\gamma, \Pi_\delta, \Pi_\theta \rangle$, where (a) T is a regression test suite for P, and (b) Π_α, Π_β, Π_γ, Π_δ, and Π_θ represent, respectively, sets of workflow branches, sets of XRG branches, sets of WSDL elements, sets of XRG patterns, and sets of tag values and unique tags in XML messages collected from the executions of all the test cases in T against P.

For any test case $t \in T$, $\Pi_\alpha(t)$, $\Pi_\beta(t)$, $\Pi_\gamma(t)$, $\Pi_\delta(t)$, and $\Pi_\theta(t)$ represent, respectively, the set of workflow branches, the set of XRG branches, the set of WSDL elements, the set of XRG patterns, and the set of tag values and unique tags in XML messages covered by the execution of t against P.

For ease of presentation, we refer to the five levels as CM-*i* levels, where CM stands for Coverage Model and $i = 1$ to 5.

4.2 Refinement-Oriented Level-Exploration Strategy and ROLE-Enriched Prioritization Techniques

In this section, we illustrate a new aspect that systematically explores coverage data in a *stepwise refinement* manner. Independent of the prioritization strategy used, our work makes more deterministic choices along the sequence of coverage data. We refer to it as a ***level-exploration strategy***.

Given a coverage model, one may formulate a strategy to explore different levels of details. In this section, we present a *Refinement-Oriented Level-Exploration* (*ROLE*) strategy.

We adopt two prioritization strategies to illustrate our strategy: the *additional* strategy and the *total* strategy [19] as baselines. They are equipped with the CM-1 level of detail to become the techniques M1 and M2 presented in Section 4.2.1. We illustrate these two prioritization strategies with branch coverage because they are *still* the most effective series of TCP techniques since the inception of TCP research [50].

ROLE enriches each prioritization strategy S with increasing levels of details. Whenever S cannot resolve ambiguity in test case priority due to equivalent coverage statistics, ROLE exposes the level of detail CM-(i+1) next to the current level of detail CM-*i*. At each new level of detail, it uses the strategy S to prioritize test cases still in tie using the coverage data of that level. Each refinement step turns nondeterministic choices into more deterministic choices.

ROLE is orthogonal to the prioritization strategy S used. For instance, it can be incorporated into a coverage-based prioritization strategy S that adopts random resolution of tie cases. In theory, some prioritization strategies have their own deterministic approaches to resolving tie cases. For example, a prioritization strategy may consistently prefer to give a higher priority to test cases having smaller test case identities (or appearing earlier in a test suite). Intuitively, such a deterministic strategy can be replaced by the ROLE strategy.

4.2.1 CM-1 Level: Baselines

M1 (Total-CM1) is the *total-branch* technique [19]. It sorts the test cases in T in descending order of the total number of Π_α items executed by each test case. If a set of test cases cover the same number of Π_α items, M1 orders them randomly.

M2 (Addtl-CM1) [19] iteratively selects a test case t in T that yields the greatest cumulative Π_α item coverage, and then removes the covered Π_α items from all remaining test cases to indicate that the removed items have been covered. Additional iterations will be conducted until all the Π_α items have been covered by at least one selected test case. If multiple test cases cover the same number of Π_α items in the current round of selection, M2 selects one of them randomly. If no remaining test cases can further improve the cumulative Π_α item coverage, M2 resets the Π_α item covered by each remaining test case to its original value. It repeats the above procedure until all test cases in T have been selected.

4.2.2 CM-2 to CM-5 Levels

This section presents techniques M3 to M10 recursively. We first describe the ROLE-Enriched techniques based on the *total* strategy, followed by the *additional* strategy.

Total-CM*i*-Refine Techniques: M3 (Total-CM2-Refine), M5 (Total-CM3-Refine), M7 (Total-CM4-Refine), and M9 (Total-CM5-Refine). Total-CM*i*-Refine (*i* = 2 to 5) is the same as Total-CM(*i*–1)-Refine, except when multiple test cases cover the same number of CM-(*i*–1) items, it will order them in descending order of the number of CM-*i* items covered by each test case involved in the tie. If there is still a tie, Total-CM*i*-Refine randomly orders the test cases involved.

Addtl-CM*i*-Refine techniques: M4 (Addtl-CM2-Refine), M6 (Addtl-CM3-Refine), M8 (Addtl-CM4-Refine), and M10 (Addtl-CM5-Refine). Addtl-CM*i*-Refine (*i* = 2 to 5) is the same as Addtl-CM(*i*–1)-Refine except three things. (1) In each iteration, Addtl-CM*i*-Refine removes the covered CM-1 to CM-*i* items of the selected test cases from the remaining test cases to indicate that the removed items have been covered by the selected test cases. (Note that Addtl-CM*i*-Refine still selects test cases based on the CM-1 item coverage as in M2.) (2) If multiple test cases cover the same number of CM-(*i*–1) items in the current round of selection, Addtl-CM*i*-Refine selects the test case that has the maximum number of uncovered CM-*i* items. If there is still a tie, it randomly selects one of the test cases involved. (3) When resetting is needed, Addtl-CM*i*-Refine resets each remaining test case to the corresponding original coverage of CM-1 to CM-*i* items.

4.3 Illustration Using Motivating Example

Different XRG branches may lead to different content selections, and return different values to the workflow step [29]. For example, the XRG branch of t_1 extracts the value 150 from the price tag and assigns the value to the variable *Price*. However, for t_2, t_3, t_5, t_7, and t_8, it will return no value (referred to as the "null value" for ease of discussion) to *Price*.

As shown in Table 2, test cases t_1, t_4, and t_6 cover the same set of XRG branches each; and test cases t_1, t_4, and t_6 covers another set of XRG branches each. The XRG branches covered by t_5 are different from the other seven test cases.

After considering the XRG branches in solving the tie cases, $t_1 \rightarrow t_4 \rightarrow t_6 \rightarrow t_7 \rightarrow t_8 \rightarrow t_2 \rightarrow t_3 \rightarrow t_5$ (with an APFD value of 0.50) is one of the *least effective* orderings.

Similarly, the above level of detail, denoted by CM-2, does not help resolve tie cases among t_1, t_4, and t_6, or among t_2, t_3, t_7, and t_8. ROLE then extends the coverage model to the next level of detail, denoted by CM-3. Table 3 shows that t_1, t_2, and t_7 cover the same set of WSDL elements but are different from those covered by the other test cases. By using the *total* prioritization strategy and the WSDL coverage data to resolve tie cases, $t_1 \rightarrow t_4 \rightarrow t_6 \rightarrow t_7 \rightarrow t_2 \rightarrow t_8 \rightarrow t_3 \rightarrow t_5$ (with an APFD value of 0.56) is one of *least effective* orderings.

The above CM-3 level of detail does not resolve tie cases among t_4 and t_6, among t_2 and t_7, and among t_3 and t_8. ROLE thus further includes the next level of detail, denoted by CM-4, as shown in Table 4. For the purpose of illustration, we only consider the XRG pattern at node A_4.

From Figure 5(a) and Figure 6, t_1 only searches for hotels from the hotel list, whereas t_4 only selects from a specific hotel. Both test cases cannot select any hotel, even though the XRG patterns they cover are different. We denote the query paths of t_1 and t_4 on the XML messages in Figure 6 by XRG patterns $x1$ and $x2$, respectively. As an illustration that adding more coverage data may not resolve tie cases, these two XRG patterns cannot resolve tie cases among t_4 and t_6, among t_2 and t_7, and among t_3 and t_8. Hence, $t_1 \rightarrow t_4 \rightarrow t_6 \rightarrow t_7 \rightarrow t_2 \rightarrow t_8 \rightarrow t_3 \rightarrow t_5$ is still one of the *least effective* orderings.

This process is continued to include the next level of detail (the XML message level), denoted by CM-5, whose coverage data is also shown in Table 4. For example, t_4 and t_6 cover the same number of XRG patterns, but t_6 achieves higher coverage than t_4 in terms of the number of elements in XML messages, where $t_1 \rightarrow t_6 \rightarrow t_4 \rightarrow t_2 \rightarrow t_7 \rightarrow t_3 \rightarrow t_8 \rightarrow t_5$ (with an APFD value of 0.69) is one of the *least effective* orderings.

Table 5 summarizes the acronyms and reference codes of the 10 techniques presented above. We also use sample least effective prioritization results of t_1–t_8 against the *HotelBooking* process to illustrate each technique. We also show the APFD values of each sample ordering under the "APFD" column.

TABLE 5
DIFFERENT LEVELS OF PRIORITIZATION TECHNIQUES WITH EXAMPLES

CM Level		Technique	Ref. Code	Example of least effective ordering								APFD
				t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	
	CM-1	Total-CM1	M1	1	6	7	3	2	8	4	5	0.19
		Addtl-CM1	M2	1	4	5	3	2	6	7	8	0.44
ROLE-Enriched	CM-2	Total-CM2-Refine	M3	1	6	7	2	8	3	4	5	0.50
		Addtl-CM2-Refine	M4	1	4	6	3	2	5	7	8	0.50
	CM-3	Total-CM3-Refine	M5	1	5	7	2	8	3	4	6	0.56
		Addtl-CM3-Refine	M6	1	5	7	4	2	3	6	8	0.56
	CM-4	Total-CM4-Refine	M7	1	5	7	2	8	3	4	6	0.56
		Addtl-CM4-Refine	M8	1	4	2	3	8	5	6	7	0.63
	CM-5	Total-CM5-Refine	M9	1	4	6	3	8	2	5	7	0.69
		Addtl-CM5-Refine	M10	1	5	2	4	8	3	6	7	0.75

4.4 Subsumption Hierarchy of ROLE-Enriched TCP Techniques

Subsumption relations are a classical concept in various areas of computer science research, such as artificial intelligence [10], databases [23], programming languages [20], and software testing [44]. We propose a notion of subsumption relations for TCP. The basic idea is that if a TCP criterion subsumes another TCP criterion, the former defines more specific coverage requirements while the latter makes a less deterministic choice. Although there is no theoretically proven relationship between the fault detection abilities of the two criteria, empirically speaking, the latter tends to exhibit weaker fault detection ability.

Definition 3. ***Subsumption.*** Given two TCP techniques X and Y, we say that X subsumes Y (denoted by $X \rightarrow Y$) if and only if any permutation of any test suite produced by X can also be produced by Y.

The subsumption relation is reflexive, transitive, and antisymmetric, and is therefore a partial order. We have analyzed the subsumption relations among M1 to M10 and the result is summarized in Figure 7. For instance, we have shown that (M3) Total-CM2-Refine subsumes (M1) Total-CM1, and we use an arrow from M3 to M1 to represent this relation in the figure. Other arrows can be interpreted similarly.

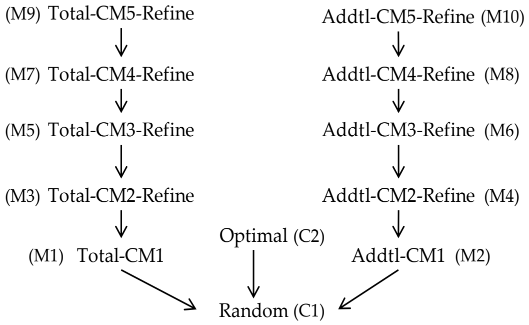


Figure 7. Subsumption hierarchy of test case prioritization techniques.

A sketch of the proof of the subsumption relations among the techniques is as follows: The basic idea is that, if random selection in resolving ties in one technique is replaced by a more deterministic procedure in another technique, then the latter technique subsumes the former. For instance, unlike M1 (which randomly resolve tie cases), M3 refers to XRG branch coverage of test cases to resolve tie cases before using random selection as the last resort. Because any test case that M3 can pick to resolve a tie may also be selected by chance by M1, any test case permutation produced by M3 must be a permutation that can be produced by M1. Other subsumption relations shown in Figure 7 can also be reasoned similarly.

4.5 Discussions of Subsumption Hierarchies

A level-exploration strategy may or may not be good enough to lead to a subsumption hierarchy. In this section, we present a negative example followed by a positive example.

Negative example. We use the summation strategy [33] as a negative example. Under this strategy, the technique at CM-*i* level treats all the coverage data from CM-1 to CM-*i* levels homogeneously and applies a given TCP strategy to prioritize all the test cases (not just the tie cases as in ROLE). Take the total prioritization strategy S_1 for the sake of discussion. Suppose that at CM-1 level, the three test cases t_a, t_b, and t_c achieve coverage counts of 1, 2 and 3, respectively; and at CM-2 level, they achieve coverage counts of 6, 4, and 2, respectively. At CM-1 level, S produces the test suite permutation $\langle t_c, t_b, t_a\rangle$. At CM-2 level, the summation strategy provides the coverage counts of 7, 6, and 5 for the three test cases for S_1 to rank test cases, which produces the test suite permutation $\langle t_a, t_b, t_c\rangle$, but this permutation is infeasible at CM-1 level.

Positive example. On the other hand, if a technique X located higher in the coverage model instance refines (but not supersede) the decision made by a technique Y located lower in the same model instance, then X will produce a test suite permutation that is also producible by Y (or the other way round). Consider, for instance, a hypothetical search-based level-exploration strategy S_2. At CM-*i* level, this strategy checks the coverage dataset at every CM-*j* level (where j = 1 to i) and finds a most similar (or diverse) not-yet-prioritized test case from each such dataset with respect to the already-prioritized test cases. The strategy S_2 then randomly picks one among these most similar (or diverse) test cases. In this case, a technique at CM-*i* level can produce more possible permutations than a technique at CM-*k* level (where $k < i$). According to Definition 3, it will lead to a subsumption hierarchy but in the reversed direction as shown in Figure 7.

In the above discussion, we use the same baseline TCP strategy across all levels. Using different strategies at different levels is a further generalization. The selection among coverage levels to be explored may also be further integrated with some coverage-based selection strategies.

5 EXPERIMENT

The relative strength in fault detection rate of TCP techniques may not necessarily be proven. We will supplement our analytical result with an experimental study in Sections 5.1 and 5.2, followed by a case study in Section 5.3.

5.1 Experimental Design

We chose eight benchmarks [29] to evaluate our work. The subjects were downloaded from the BPWS4J repository [18], Oracle BPEL Process Manager [24], IBM BPEL repository [41], and Web Services Innovation Framework [43]. These subjects are representative service-based applications developed in WS-BPEL [1], [18], [43]. Previous empirical studies (such as [28], [33]) have reported results on this benchmark suite. (Note that this suite is larger in size than other sets of subjects reported by the testing research papers of the same journal, such as [36]). Table 6 shows the descriptive statistics of the suite. The number of XML elements ("Elements") and the number of lines of BPEL code ("LOC") of each benchmark is shown in the table.

We used the set of faults and associated test suites in the benchmark suite to measure the effectiveness of different prioritization techniques. We followed the spirit of mutation testing [2] to seed faults in the major artifacts (BPEL, XPath, and WSDL) of the benchmarks. Andrews et al. [2] suggested that mutation faults can be representative of real faults. Many researchers thus used mutation testing for empirical evaluation of TCP techniques [17]. We used three typical types of mutations in fault seeding: value mutations, decision mutations, and statement mutations. Since BPEL can be treated as Control Flow Graphs (CFG), the above mutations can be seeded in the way as seeding faults in CFG. Figure 5(c) gives one example of a BPEL fault. An XPath fault is the wrong usage of XPath expressions, such as extracting the wrong content, or failing to extract any content. Figure 5(b) gives one example of an XPath fault. A WSDL fault is the 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 by [29]. The statistics of the selected modified versions are shown in the rightmost column of Table 6.

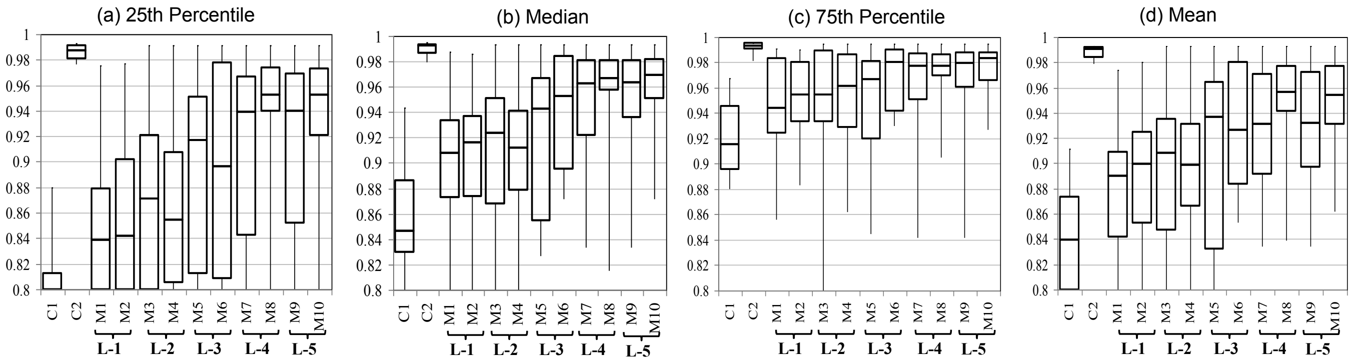


Figure 8. Overall comparisons in terms of APFD measure.

TABLE 6
BENCHMARKS AND THEIR DESCRIPTIVE STATISTICS

Ref.	Applications	Modified Versions	Elements	LOC	XPath	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 7
STATISTICS OF TEST SUITE SIZES

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

When constructing the benchmark suite [29], we implemented a tool that automatically generated a pool of 1000 test cases for each subject. The tool generated test cases to ensure that they covered all workflow branches, XRG branches, WSDL elements, XRG patterns, and types of XML messages of the original subject at least once. We then followed the common practice [19], [28], [32] in evaluating TCP techniques to discard any version if more than 20 percent of the test cases can detect failures due to its fault. The tool adopted the test suite construction process presented in [19], which ensured that the fault detection effectiveness of test suites was not influenced by the order of test case generation [19]: It randomly selected test cases one by one from a test pool and placed them in a test suite *T* (which was initially empty) without applying any test case to the modified versions of the corresponding subject. Such selection was iteratively done until all the workflow branches, XRG branches, WSDL elements, XRG patterns, and all types of XML messages had been covered at least once. If the output of the same test case against a subject and a modified version were different, the test case detected a fault in the modified version. This suite *T* would be retained if it detected a fault in a modified version. The tool successfully retained a total of 100 test suites for each benchmark. Table 7 shows their statistics.

For each subject and for each constructed test suite *T* for the subject, the tool applied every technique to prioritize *T*. The tool executed each prioritized *T* against every modified version of the subject. It used the outputs of the original version as expected outputs. It calculated the corresponding APFD, RP, and HMFD values. In total, 833,280 APFD values, 516 RP values, and 833,280 HMFD values were collected.

5.2 Data Analyses

5.2.1 Overall effectiveness

Figure 8 shows the 25th percentile, median, 75th percentile and mean APFD results of each of the techniques C1, C2, and M1–M10, in which the result of every individual technique is represented using box-plots. Each box-plot shows the 25th percentile, median, and 75th percentile of a particular technique. For instance, the 25th percentile, median, and 75th percentile of the mean APFD of M10 are shown in the last plot of Figure 8(d).

Let us first examine the overall mean APFD result of each technique in Figure 8(d). As expected, the box-plot of C2 shows the best mean APFD in the figure. M8 and M10 are only three percent less effective than C2. All the techniques M1–M10 are more effective than C1.

The effectiveness of the Total-CM series of techniques increases from CM-*i* to CM-(*i*+1) level (*i* = 1 to 4) at the 25th percentile, medium, 75th percentile, and mean APFD, except for the change in mean APFD from CM-1 to CM-2 level and the change in the 25th percentile from CM-3 to CM-4 level. Similarly, the effectiveness of the Addtl-CM series increases from CM-*i* to CM-(*i*+1) level (*i* = 1 to 4) at the 25th percentile, medium, 75th percentile and mean APFD, except for the change in the 25th percentile from CM-4 to CM-5 level.

We further investigate the impact of level changes. If a technique at CM-*i* level is worse than that at CM-(*i*–1) level in terms of median APFD value, we call the scenario an *exception*, and assign to it a value of 1 (and darken the cell); otherwise 0 is assigned. Table 8 shows the result together with the median APFD values of random ordering and the techniques at CM-1 level (M1–M2) and CM-5 level (M9–M10). When CM-1 level is more effective than CM-5 level, such as the case of the *purchase* application in the Total-CM category, an exception occurs. When random ordering is

TABLE 8
IMPACTS OF CM LEVEL CHANGES

Type	Benchmark Application	Is CM-*i* worse than CM-(*i*–1)? (1: yes, 0: no)				Exception Rate	Median APFD Values		
		i=2	*i*=3	*i*=4	*i*=5		Random	CM-1	CM-5
Total-CM	atm	0	0	0	0	0.00	0.837	0.855	0.981
	buybook	0	0	0	0	0.00	0.848	0.879	0.979
	dslservice	1	0	0	0	**0.25**	**0.806**	**0.770**	0.833
	gymlocker	0	0	0	0	0.00	0.943	0.950	0.994
	loanapproval	0	0	0	0	0.00	0.845	0.928	0.947
	marketplace	1	1	0	0	**0.50**	**0.878**	**0.893**	0.948
	purchase	0	0	1	0	**0.25**	0.760	**0.923**	**0.904**
	triphandling	0	0	0	0	0.00	0.912	0.987	0.983
Addtl-CM	atm	0	0	0	0	0.00	0.837	0.874	0.965
	buybook	0	0	0	0	0.00	0.848	0.871	0.974
	dslservice	1	0	0	0	**0.25**	**0.806**	**0.775**	0.983
	gymlocker	0	0	0	0	0.00	0.943	0.962	0.994
	loanapproval	0	0	0	0	0.00	0.845	0.925	0.961
	marketplace	0	1	1	0	**0.50**	**0.878**	**0.907**	**0.871**
	purchase	0	0	1	1	**0.50**	0.760	**0.929**	**0.919**
	triphandling	0	0	0	0	0.00	0.912	0.986	0.981

TABLE 9
STATISTICS OF TIME COSTS OF TEST SUITE PRIORITIZATION STRATEGIES (IN MILLISECONDS)

Ref.	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
A	0.15	1.09	1.24	2.65	1.71	3.12	1.71	3.72	2.34	5.36
B	0.16	0.80	0.63	1.25	0.78	1.4	1.69	2.67	1.39	3.27
C	0.31	0.32	0.93	1.87	0.77	2.18	1.55	2.53	1.41	6.08
D	0.79	0.78	1.09	1.10	1.08	3.43	1.73	4.98	2.04	5.46
E	1.08	2.65	1.25	5.00	2.5	9.03	2.66	10.9	3.72	15.89
F	0.94	1.57	1.58	5.13	1.09	6.71	1.57	8.50	2.02	13.77
G	0.79	0.79	1.23	2.33	0.64	4.69	1.69	4.06	0.93	8.41
H	0.77	1.07	0.47	2.84	1.71	2.93	0.77	6.24	2.06	7.62
Mean	0.62	1.13	1.05	2.77	1.29	4.19	1.67	5.45	1.99	8.23

TABLE 10
RESULT OF HYPOTHESIS TESTING

	C1	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
C1	–	<	<	<	<	<	<	<	<	<	<
M1	>	–	=	=	<	<	<	<	<	<	<
M2	>	=	–	>	=	=	<	<	<	<	<
M3	>	=	<	–	<	<	<	<	<	<	<
M4	>	>	=	>	–	=	<	<	<	<	<
M5	>	>	=	>	=	–	<	<	<	<	<
M6	>	>	>	>	>	>	–	=	<	=	<
M7	>	>	>	>	>	>	=	–	<	=	<
M8	>	>	>	>	>	>	>	>	–	>	=
M9	>	>	>	>	>	>	=	=	<	–	<
M10	>	>	>	>	>	>	>	>	=	>	–

more effective than CM-1 level, such as the case of the *dslservice* application in the Total-CM category, an exception is also said to occur. The observations on the exceptions in the Addtl-CM category are similar to the Total-CM category.

There are 64 comparisons in total, but only 9 exceptions. Hence, 85.9% of the cases show improved effectiveness when the level in the subsumption hierarchy increases.

By comparing the pairs of techniques (M1, M2), (M3, M4), (M5, M6), (M7, M8), and (M9, M10), we observe that an Addtl-CM technique has a shorter length of the box (smaller variances) than the corresponding Total-CM technique at the same CM level, as shown in Figure 8. We observe that the effectiveness of the Total-CM series grows faster than that of the Addtl-CM techniques, and the Addtl-CM techniques do not change significantly until CM-4 and CM-5 levels.

We also observe a trend where a technique is more likely to achieve a higher fault detection rate (in terms of APFD) than a technique that is subsumed by the former. For example, M3 is more effective than M1 and subsumes M1. In Figure 8(d), we find that M6–M10 are generally better than all the other techniques except C2. When we focus on the techniques M1–M4, we find M2 and M4 to be the best two among all the techniques at the same level. In Figure 8(d), the lengths of the boxes in the bars at CM-4 level (M7 and M8) and CM-5 level (M9 and M10) are shorter than the boxes at lower levels ({M1, M3, M5} and {M2, M4, M6}, respectively, and also shorter than the box for random ordering.

We have also collected the times to prioritize test suites for M1 to M10. Table 9 shows that, as the CM level increases, the Total-CM techniques and Addtl-CM techniques use more time. However, even M9 and M10, which use the most time in the corresponding Total-CM series and Addtl-CM series, only use 1.99 and 8.23 milliseconds, respectively.

5.2.2 Hypothesis testing

We have also performed a one-way analysis of variance (*ANOVA*) using MatLab to find out whether the mean APFD for different techniques differ significantly. The null hypothesis is that the mean APFD values for C1 and M1–M10 (11 techniques in total) are equal. To decide whether to accept or reject the null hypothesis, we set the significance level to 5%. For each benchmark, we find that ANOVA returns a *p*-value much less than 0.05, which successfully rejects the null hypothesis at a significance level of 5%.

Following [32], we further apply the multiple comparison procedure to study which TCPs have mean values that differ significantly from others at a significance level of 5%. The *Least Significant Difference* (*LSD*) method was employed in multiple-comparison. Table 10 summarizes the hypothesis testing result. In the table, the symbol ">", "=", "<" refer to the technique in the row is more effective, equally effective (indicating no statistical difference rather than the same distribution), and less effective than the technique in the column, respectively. In general, a technique with more instances of ">" indicates that it is an effective technique.

Table 10 shows that, at CM-2 level and above, a technique achieves a higher APFD value than a technique subsumed by the former, except between M8 and M10 and between M7 and M9. Between CM-1 and CM-2 levels, however, there is no consistent and statistically significant difference.

5.2.3 Further Evaluation of the Benchmarks

We further evaluate the techniques using manual test suites. We have invited *five* non-author and experienced testers (who are vendor developers having 3 to 5 years of testing experience) to manually develop a test suite for each bench-

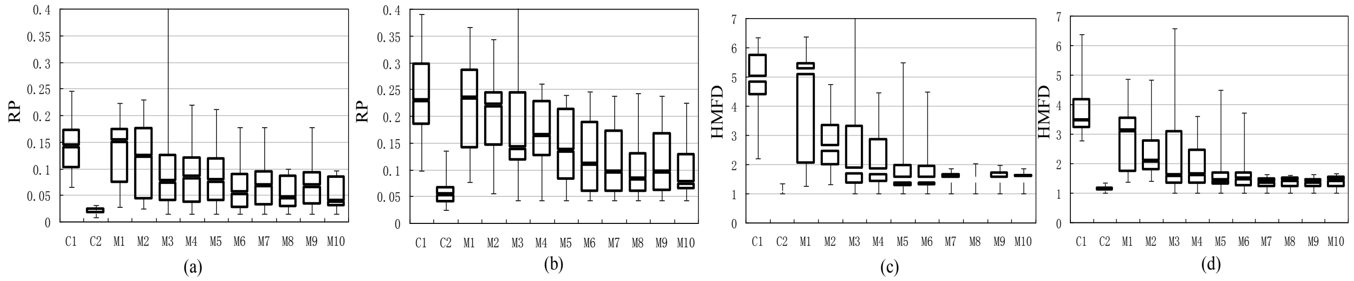


Figure 9. Results of techniques in terms of (a) RP using tool-generated test suites, (b) RP using manual test suites, (c) HMFD using tool-generated test suites, and (d) HMFD using manual test suites.

mark to cover all the workflow branches, XRG branches, WSDL elements, XRG patterns, and types of XML messages of the original subject at least once. Table 11 shows the statistics of the manual test suites. The sizes of the manual test suites are smaller than those of the tool-generated test suites, since the latter test suites contain more randomly selected test cases.

TABLE 11
STATISTICS OF MANUAL TEST SUITE SIZES

Ref. Size	A	B	C	D	E	F	G	H	Mean
Maximum	40	31	25	32	33	20	15	71	33
Mean	32	24	21	25	28	14	12	54	26
Minimum	26	18	18	19	22	8	8	44	20

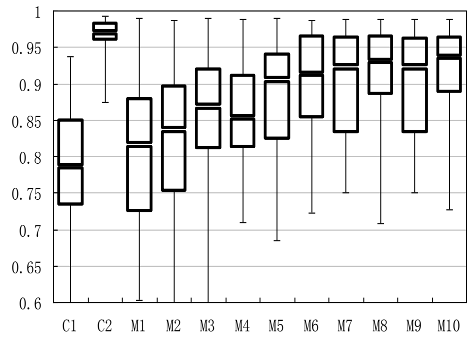


Figure 10. Overall comparisons in terms of mean APFD using manual test suites.

We applied each technique to this test suite, and repeated the procedure 100 times. Figure 10 presents the comparison results in terms of the mean APFD measure.

Comparing Figure 8(d) and Figure 10, we find the trends of the Total-CM series and the Addtl-CM series in both figures are similar. For each series, a technique at CM-*i* level (for *i* = 2 to 5) shows more effective result than that at CM-(*i*–1) level. The result of the manual test suites consolidates our finding on using tool-generated test suites to evaluate the proposed subsumption hierarchy.

We also observe that when a technique uses a manual test suite, it has a smaller APFD value than when it uses a tool-generated test suite. This is because tool-generated test suites contain more randomly selected test cases, which may increase the number of times to reveal a failure, and thus increase the chance of finding a fault earlier.

We also analyze the raw dataset that provides the APFD results using RP and HMFD as alternatives. The results are shown in Figure 10. We find that across the board, the general trends among techniques are similar to what we observe from the APFD values, namely, that as the CM level increases, the prioritization strategy can become more effective. In addition, using manual test suites can be less effective and more effective than tool-generated test suites in terms of RP and HMFD, respectively.

5.3 Case Study

We further evaluated our proposal using the service application presented in the case study of [30]. It was a real-life choreography service for *Data Exchange Platform* (*DEP*). Due to the page limit, we only briefly revisit the key features of this application from [30].

The application had four major subject services. Table 12 recaps their statistics. We followed the fault seeding strategies stated in Section 5.1 (that is, in the spirit of mutation testing [2]) to seed faults in the major artifacts (WSDL, XPath, and WS-CDL specifications [30]) of DEP. The functions of these subject services are as follows:

(1) *AgentService* monitors the database updates, collects the change logs, and collaborates with *MonitorService* to update the data stored in other information systems.
(2) *DataService* enables an agent to upload data to the server and to download data from the server.
(3) *MonitorService* handles the requests from *AgentService*, verifies the authority of the agent, and allocates a data transfer thread to handle the authenticated request.
(4) *AuthenticationService* authenticates whether an agent has the rights to perform the data transfer.

TABLE 12
DESCRIPTIVE STATISTICS OF SUBJECTS IN CASE STUDY

Services	No. of Ports	No. of WSDL	No. of XPath	SLOC (Java)	No. of Faults
AgentService	6	2	6	4,000–5,000	3
MonitorService	8	2	8	6,000–7,000	3
DataService	4	1	4	3,000–4,000	2
AuthenticationService	2	1	2	1,000–2,000	2
Total	20	6	20	> 14,000	10

There was, however, no workflow information in this service application. We used labeled queries [30] (similar to

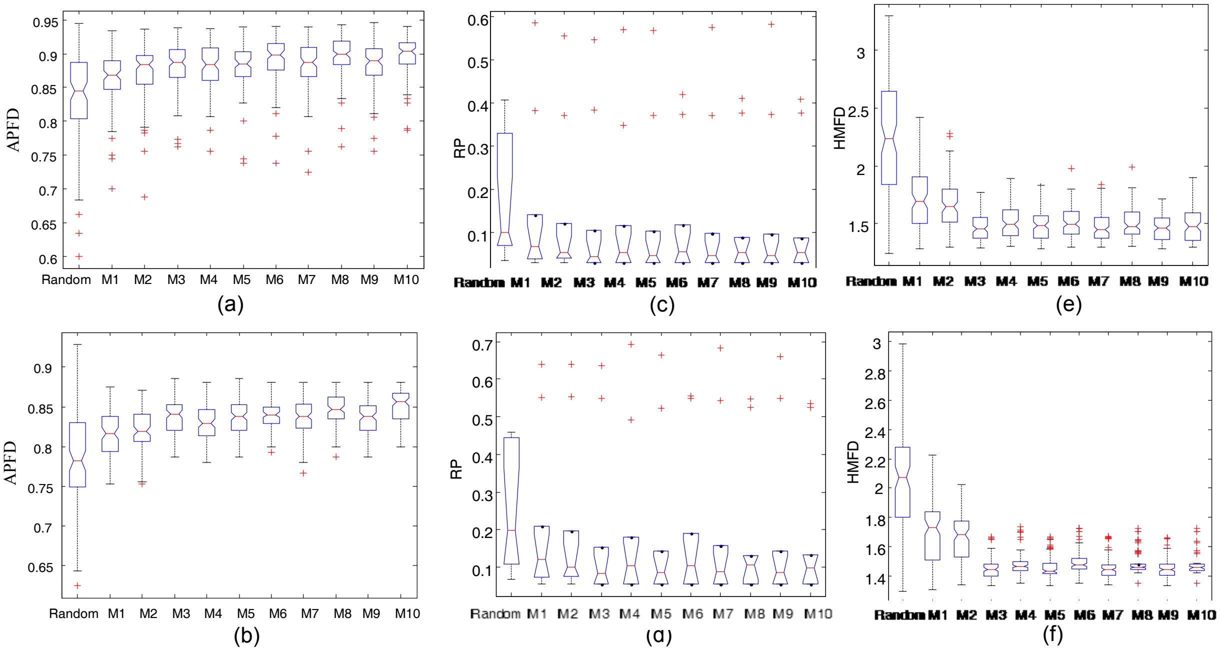


Figure 11. Comparisons in case study, in terms of (a) APFD using tool-generated test suites, (b) APFD using manual test suites, (c) RP using tool-generated test suites, (d) RP using manual test suites, (e) HMFD using tool-generated test suites, and (f) HMFD using manual test suites.

workflow transitions) to replace the workflow data as the Π_α coverage instead. The coverage items for Π_β, Π_γ, Π_δ, and Π_θ were not affected. All the remaining experimental procedure was the same as that presented in Section 5.1. The minimum, average, and maximum test suite sizes were 8, 42, and 178, respectively.

Figure 11(a), (c), and (e) show the APFD, RP, and HMFD results, respectively, of C1 and M1–M10. We observe that the effectiveness of the Total-CM series and the Addtl-CM series of techniques increase from CM-*i* to CM-(*i*+1) level (for *i* = 1 to 4) at the 25th percentile, the medium, the 75th percentile, and the mean for each of the APFD, RP, and HMFD measures.

We find that M1–M10 all outperform C1. M3–M10 all outperform M1 and M2. We also observe that the APFD and HMFD values for outliers (denoted by red + signs) become higher as the coverage levels increase.

We also invited the five testers (introduced in Section 5.2.3) to manually develop five test suites. The maximum, minimum, and average suite sizes were 24, 15, and 19, respectively. We applied each technique to this test suite, and repeated the procedure 20 times. Figure 11(b), (d), and (f) present the APFD, RP, and HMFD results, respectively. Comparing between subfigures (a) and (b), subfigures (c) and (d), and subfigures (e) and (f) of Figure 11, we find the trends of the Total-CM series and the Addtl-CM series in each pair of plots to be similar. We observe that as the CM level increases, the difference between the two types of test suites gradually becomes smaller We find that for each technique (except C1), the standard derivation for the manual test suite is smaller than that for the tool-generated test suite in terms of APFD and HMFD. We find that the techniques using the manually-crafted test suite are slightly less effective than the same techniques using the tool-generated test suites in terms of APFD and RP, but are slightly more effective in terms of HMFD. The results of the manual test suite consolidate our finding on using tool-generated test suites to evaluate the proposed subsumption hierarchy. We also observe that Figure 11(a) and (b) report smaller relative differences between techniques than Figure 8 and Figure 10.

5.4 Threats to Validity

We used APFD, RP, and HMFD in our experiment. They provide useful feedback to specific prioritization techniques after testing has been complete. Different metrics measure different aspects of testing techniques.

Detecting mutation faults can simulate the detection of real faults in the same program [2], and many studies have used these faults to evaluate TCP techniques. We also used mutants. We did not measure the costs of test execution and profiling because remote service executions were obviously the major bottlenecks and independent of the testing techniques used. Although the techniques shared the same such cost if they used the same coverage data, interpreting results across different levels should be carefully conducted. We used both tool-generated test suites and manual test suites, and observed similar trends between them. We implemented our tools for program instrumentation and test suite prioritization in Java, and used MatLab to compute the experimental results. To minimize errors, we carefully tested our tools to assure their correctness. The responses of some services depend on the service contexts (such as database status). In the experiment, our tool did reset the contexts to the same values every time before executing a test case. This approach is also advocated in agile software development.

Our subjects included one real-world choreography service-based application and eight orchestration applications. Our studies also covered mutation faults, as well as both manual and tool-generated test suites. These factors should be considered if the results are used beyond the experimental context.

6 Related Work

Regression testing has been extensively studied [14], [37]. Our work is a kind of general test case prioritization [19], which reorders a test suite for a service P to be useful in subsequent revised versions of P. To adopt it for version-specific test case prioritization, one has to find out the differences between a preceding version and the modified version of the same service. Ruth and Tu [37], Chen et al. [14], Li et al. [26], Liu et al. [27], and Tarhini et al. [40] contributed to this topic in services computing. They conducted impact analyses of web services to identify revised fragments of code in a service by comparing the flow graphs between versions of the same service.

Li et al. [26] selected workflow paths to ease regression testing based on the insights collected from messages. Liu et al. [27] considered concurrency control activities and their control flow in BPEL processes to make regression testing more effective. Tarhini et al. [40] developed a model-based approach to impact analysis so that regression testing can address changes in various development phases.

There have been other studies on TCP in services computing. Chen et al. [14] proposed weighted test case prioritization. Hou et al. [22] proposed to fit the requests (within a maximal number) imposed by external services into TCP. Mei et al. [31] considered an external service may change within a round of regression testing of a WS-BPEL web service. They proposed to detect whether the code coverage of BPEL code had been changed, and initialized nested rounds of regression testing to address this issue. They [28], [32] further followed the preliminary version [33] of the present paper to consider multilevel coverage, but they did not consider the relationships with XPath queries (including XRG and XRG patterns). The mean APFD results published in [28], [32] seldom exceeded 0.90. Our experimental result presented in Section 5 of this paper shows that quite a number of our techniques exceed this mean APFD.

Nguyen et al. [35] integrated TCP with audit testing to control resource consumption. Zhai et al. [49] observed that service selection had the ability to include/exclude a service in consideration, and used this feature to reduce the service invocation cost. They [48] studied location-centric diversity strategies to reorder test cases for location-based services.

There have been many projects on other topics in the testing of web services. Bozkurt et al. [9] provided a comprehensive summary on the testing and verification of service-oriented architecture. Bartolini et al. [5] proposed to use a dataflow-based approach to validate the composition of web services. Mei et al. [29], [30] formulated dataflow-based test adequacy criteria to test WS-BPEL web services. As we have presented in Section 4, our work builds on top of these two studies [29], [30] and puts forward a subsumption hierarchy of prioritization techniques.

Casado et al. [11] used a classification-tree methodology [15], [21] to determine test coverage in the testing of web services transactions. Our coverage model has not considered constraints among coverage elements.

In terms of multilevel coverage, the closest related work is Zou et al. [52]. They studied the integration of coverage data on program statements and HTML elements for the testing of dynamic web applications. Their work did not consider XPath queries and regression testing.

Maintaining a regression test suite has also been an active area of research. Becker et al. [7] checked whether a document of a service is backwardly compatible. Belli et al. [8] and Zheng et al. [51] studied model-based approaches to constructing both abstract and concrete test cases semi-automatically. Li et al. [26] studied the generation of control-flow test cases for the unit testing of BPEL programs. Bartolini et al. [6] generated test cases that conform to WSDL schemas so that these test cases could be meaningfully run by a service under test. Li et al. [26] studied test case selection. All these projects can significantly enhance the practicability of our work.

Our present work also requires profiling the service executions. Bartolini et al. [4] extracted state machine data based on messages from opaque web services. It appears that their model can be integrated with our strategy. Bai et al. [3] studied web services with ontology. Ni et al. [36] modeled a WS-BPEL web service as a message-sequence graph and suggested coordinating messages to control service execution. de Almeida and Vergilio [16] and Xu et al. [46] perturbed inputs to produce test cases for robustness testing.

Regression testing should also address the test oracle problem. Both Chan et al. [12] and Sun et al. [38] studied the use of metamorphic relations to address this problem.

7 Conclusion

In this paper, we have proposed a multi-coverage model, the first refinement-oriented level-exploration strategy, and the first subsumption hierarchy of test case prioritization techniques in the context of regression testing of composite services. To the best of our knowledge, all existing studies on TCP exhaustively observe the effects of prioritization techniques from empirical studies. We have shown that some test case prioritization techniques can be compared analytically. It significantly complements the inadequacy of existing work in theoretical studies of TCP. We have verified our proposal through an experiment.

It will be interesting to extend the notion of level-exploration to other branches of regression testing and handle other types of service scenarios.

References

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

[2] J.H. Andrews, L.C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" *Proceedings of the 27th International Conference on Software Engineering* (*ICSE '05*), pp. 402–411, 2005.

[3] X. Bai, S. Lee, W.-T. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of web services," *Proceedings of the IEEE International Conference on Web Services* (*ICWS '08*), pp. 465–472, 2008.

[4] 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* (*JSS*), 84(4): 655–668, 2011.

[5] C. Bartolini, A. Bertolino, E. Marchetti, and I. Parissis, "Data flow-based validation of web services compositions: perspectives and examples," *Architecting Dependable Systems V*, LNCS, vol. 5135, pp. 298–325, 2008.

[6] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "Towards automated WSDL-based testing of web services," *Service-Oriented Computing* (*ICSOC '08*), LNCS, vol. 5364, pp. 524–529, 2008.

[7] K. Becker, J. Pruyne, S. Singhal, A. Lopes, and D. Milojicic, "Automatic determination of compatibility in evolving services," *International Journal of Web Services Research* (*JWSR*), 8(1): 21–40, 2011.

[8] F. Belli, A.T. Endo, M. Linschulte, and A. Simão, "A holistic approach to model-based testing of web service compositions," *Software: Practice and Experience*, 44(2): 201–234, 2014.

[9] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing and verification in service-oriented architecture: a survey," *Software Testing, Verification and Reliability* (*STVR*), 23(4): 261–313, 2013.

[10] R.J. Brachman and J.G. Schmolze, "An overview of the KL-ONE Knowledge Representation System," *Cognitive Science* 9(2), 1985.

[11] R. Casado, M. Younas, and J. Tuya, "Multi-dimensional criteria for testing web services transactions," *Journal of Computer and System Sciences*, 2013, doi: 10.1016/j.jcss.2013.01.020.

[12] W.K. Chan, S.C. Cheung, and K.R.P.H. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," *JWSR*, 4(2): 60–80, 2007.

[13] H.Y. Chen, T.H. Tse, and T.Y. Chen, "TACCLE: a methodology for object-oriented software testing at the class and cluster levels," *ACM Transactions on Software Engineering and Methodology*, 10(1): 56–109, 2001.

[14] L. Chen, Z. Wang, L. Xu, H. Lu, and B. Xu, "Test case prioritization for web service regression testing," *Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering* (*SOSE '10*), pp. 173–178, 2010.

[15] T.Y. Chen and P.-L. Poon, "On the effectiveness of classification trees for test case construction," *Information and Software Technology*, 40(13):765–775, 1998.

[16] L.F. de Almeida, Junior and S.R. Vergilio, "Exploring perturbation based testing for web services," *Proceedings of ICWS '06*, pp. 717–726, 2006.

[17] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Transactions on Software Engineering* (*TSE*), 32(9): 733–752, 2006.

[18] *Eclipse Environment Implementation of the Business Process Execution Language Engine* (*BPWS4J Engine 2.1*), http://en.pudn.com/downloads53/sourcecode/middleware/detail184250_en.html.

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

[20] R.E. Griswold, D.R. Hanson, and J.T. Korb, "Generators in icon," *ACM Transactions on Programming Languages and Systems*, 3(2): 144–161, 1981.

[21] M. Grochtmann and K. Grimm, "Classification trees for partition testing," *STVR*, 3(2): 63–82, 1993.

[22] S.-S. Hou, L. Zhang, T. Xie, and J.-S. Sun, "Quota-constrained test-case prioritization for regression testing of service-centric systems," *Proceedings of the IEEE International Conference on Software Maintenance* (*ICSM '08*), pp. 257–266, 2008.

[23] H.V. Jagadish, "Incorporating hierarchy in a relational model of data," *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data* (*SIGMOD '89*), pp. 78–87, 1989.

[24] 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.

[25] H.K.N. Leung and L.J. White, "Insights into regression testing," *Proceedings of ICSM '89*, pp. 60–69, 1989.

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

[27] H. Liu, Z. Li, J. Zhu, and H. Tan, "Business process regression testing," *Proceedings of ICSOC '07*, LNCS 4749, pp. 157–168, 2007.

[28] L. Mei, Y. Cai, C. Jia, B. Jiang, and W.K. Chan, "Prioritizing structurally complex test pairs for validating WS-BPEL evolutions," *Proceedings of ICWS '13*, pp. 147–154, 2013.

[29] L. Mei, W.K. Chan, and T.H. Tse, "Data flow testing of service-oriented workflow applications," *Proceedings of ICSE '08*, pp. 371–380, 2008.

[30] L. Mei, W.K. Chan, and T.H. Tse, "Data flow testing of service choreography," *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*ESEC '09 / FSE-17*), pp. 151–160, 2009.

[31] 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* (*TSC*), in press.

[32] L. Mei, W.K. Chan, T.H. Tse, and R.G. Merkel, "XML-manipulating test case prioritization for XML-manipulating services," *JSS*, 84(4): 603–619, 2011.

[33] 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*), pp. 901–910, 2009.

[34] G. Miklau and D. Suciu, "Containment and equivalence for a fragment of XPath," *Journal of the ACM*, 51(1): 2–45, 2004.

[35] C.D. Nguyen, A. Marchetto, and P. Tonella, "Test case prioritization for audit testing of evolving web services using information retrieval techniques," *Proceedings of ICWS '11*, pp. 636–643, 2011.

[36] 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," *TSC*, 6(1): 7–19, 2013.

[37] M.E. Ruth and S. Tu, "Towards automating regression test selection for web services," *Proceedings of WWW '07*, pp. 1265–1266, 2007.

[38] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T.Y. Chen, "A metamorphic relation-based approach to testing web services without oracles," *JWSR*, 9(1): 51–73, 2012.

[39] L. Tahat, B. Korel, M. Harman, and H. Ural, "Regression test suite prioritization using system models," *STVR*, 22(7): 481–506, 2012.

[40] A. Tarhini, H. Fouchal, and N. Mansour, "Regression testing web services-based applications," *Proceedings of the IEEE International Conference on Computer Systems and Applications* (*AICCSA '06*), pp. 163–170, 2006.

[41] *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.

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

[43] *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.

[44] E.J. Weyuker, S.N. Weiss, and D. Hamlet, "Comparison of program testing strategies," *Proceedings of the ACM SIGSOFT 4th Symposium on Software Testing, Analysis, and Verification* (*TAV 4*), pp. 1–10, 1991.

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

[46] W. Xu, J. Offutt, and J. Luo, "Testing web services by XML perturbation," *Proceedings of the 16th International Symposium on Software Reliability Engineering* (*ISSRE '05*), pp. 257–266, 2005.

[47] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *STVR*, 22(2): 67–120, 2012.

[48] K. Zhai, B. Jiang, and W.K. Chan, "Prioritizing test cases for regression testing of location-based services: metrics, techniques and case study," *TSC* 7(1): 54–67, 2014.

[49] 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 ICWS '10*, pp. 211–218, 2010.

[50] 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 ICSE '13*, pp. 192–201, 2013.

[51] Y. Zheng, J. Zhou, and P. Krause, "An automatic test case generation framework for web services," *Journal of Software*, 2(3): 64–77, 2007.

[52] Y. Zou, C. Feng, Z. Chen, X. Zhang, and Z. Zhao, "A hybrid coverage criterion for dynamic web testing," *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering* (*SEKE '13*), 2013.

Lijun Mei is a staff researcher at IBM Research – China. He obtained the PhD degree from The University of Hong Kong. His research interest is to address the issues of program testing and testing management in the business environment. He has conducted extensive research in testing service-based applications.

Yan Cai Yan Cai received the BEng degree in computer science and technology from Shandong University, China, in 2009. He is currently working toward the PhD degree at the Department of Computer Science, City University of Hong Kong. His current research interests include concurrency bug detection and reproduction in large-scale multithreaded and concurrent systems. His research results have been reported in venues such as TSE, TPDS, SPE, JWSR, ICSE, ISSRE, and ICWS. He is a student member of the IEEE.

Changjiang Jia is a PhD student at Department of Computer Science, City University of Hong Kong. He received the BEng and MEng degrees from National University of Defense Technology, China. His research interests are concurrency bug detection and failure diagnosis in large-scale multithreaded programs. His research results have been reported in QSIC, ICWS, JWSR and TPDS. He is a student member of the IEEE.

Bo Jiang is an assistant professor at Beihang University. He obtained the PhD degree from The University of Hong Kong. His research interests are the reliability of mobile applications, program debugging, adaptive testing, and regression testing. He has received the best paper awards from COMPSAC '08, COMPSAC '09, and QSIC '11.

W.K. Chan is an assistant professor at City University of Hong Kong. His current main research interest is program analysis and testing for concurrent software and systems. He is on the editorial board of the *Journal of Systems and Software*. He has published extensively in venues such as *TOSEM*, *TSE*, *TPDS*, *TSC*, *CACM*, *Computer*, ICSE, FSE, ISSTA, ASE, WWW, ICWS, and ICDCS.

Zhenyu Zhang is an associate professor at the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences. He obtained the PhD degree from The University of Hong Kong. His current research interests are program debugging and testing for software and systems, and the reliability issues of web-based services and cloud-based systems. He has published research results in venues such as *Computer*, ICSE, FSE, ASE, and WWW.

T.H. Tse received the PhD degree from the London School of Economics and was a visiting fellow at the University of Oxford. He is a professor in computer science at The University of Hong Kong. His current research interest is in program testing, debugging, and analysis. He is the steering committee chair of QSIC and an editorial board member of the *Journal of Systems and Software*, *Software Testing, Verification and Reliability*, *Software: Practice and Experience*, and the *Journal of Universal Computer Science*. He also served on the search committee for the editor-in-chief of the *IEEE Transactions on Software Engineering* in 2013. He is a fellow of the British Computer Society, a fellow of the Institute for the Management of Information Systems, a fellow of the Institute of Mathematics and Its Applications, and a fellow of the Hong Kong Institution of Engineers. He was awarded an MBE by The Queen of the United Kingdom.