Dynamic Random Testing of Web Services: A Methodology and Evaluation

Chang-ai Sun, *Senior Member, IEEE,* Hepeng Dai, Guan Wang, Dave Towey, *Member, IEEE,* Kai-Yuan Cai, *Member, IEEE,* and Tsong Yueh Chen, *Member, IEEE,*

Abstract—In recent years, Service Oriented Architecture (SOA) has been increasingly adopted to develop distributed applications in the context of the Internet. To develop reliable SOA-based applications, an important issue is how to ensure the quality of web services. */** Dave [3]: We are not consistent in our use of "web" or "Web": which would you prefer? Can we update the paper to ensure consistency? **/* In this paper, we propose a dynamic random testing (DRT) technique for web services that is an improvement over the widely-practiced random testing (RT) and partition testing (PT). We examine key issues when adapting DRT to the context of SOA, including a framework, guidelines for parameter settings, and a prototype for such an adaptation. Empirical studies are reported where DRT is used to test three real-life web services, and mutation analysis is employed to measure the effectiveness. Our experimental results show that, compared with the two baseline techniques, RT and Random Partition Testing (RPT), DRT demonstrates higher fault-detection effectiveness with a lower test case selection overhead. Furthermore, the theoretical guidelines of parameter setting for DRT are confirmed to be effective. The proposed DRT and the prototype provide an effective and efficient approach for testing web services.

Index Terms—Software Testing, Random Testing, Dynamic Random Testing, Web Service, Service Oriented Architecture.

1 Introduction

SERVICE oriented architecture (SOA) [2] defines a loosely coupled, standards-based, service-oriented application development paradigm in the context of the Internet. Within SOA, three key roles are defined: service providers (who develop and own services); service requestors (who consume or invoke services); and a service registry (that registers services from providers and returns services to requestors). Applications are built upon services that present functionalities through publishing their interfaces in appropriate repositories, abstracting away from the underlying implementation. Published interfaces may be searched by other services or users, and then invoked. Web services are the realization of SOA based on open standards and infrastructures [3]. Ensuring the reliability of SOA-based applications can become critical when such applications implement important business processes.

Software testing is a practical method for ensuring the quality and reliability of software. However, some SOA features can pose challenges for the testing of web services [4], [5]. For instance, service requestors often do not have access to the source code of web services which are published and owned by another organization, and, consequently, it is not possible to use white-box testing techniques. Testers may, therefore, naturally turn to black-box testing techniques.

Random Testing (RT) [6] is one of the most widely-practiced black-box testing techniques. Because test cases in RT are randomly selected from the input domain (which refers to the set of all possible inputs of the software under test), it can be easy to implement. Nevertheless, because RT does not make use of any information about the software under test (SUT), or the test history, it may be inefficient in some situations. In recent years, many efforts have been made to improve to RT in different ways [7]–[9]. Adaptive random testing (ART) [8], [10]–[12], for example, has been proposed to improve RT by attempting to have a more diverse distribution of test cases in the input domain.

In contrast to RT, partition testing (PT) attempts to generate test cases in a more "systematic" way, aiming to use fewer test cases to reveal more faults. When conducting PT, the input domain of the SUT is divided into disjoint partitions, with test cases then selected from each and every one. Each partition is expected to have a certain degree of homogeneity—test cases in the same partition should have similar software execution behavior. Ideally, a partition should also be homogeneous in fault detection: If one input can reveal a fault, then all other inputs in the same partition should also be able to reveal a fault. */** Dave [4]: faults or failures? **/ /== Chang-ai [1]: A fault may result in a failure in case there is no protection. Here, we would not distinguish faults and failures. Instead, we focus on the occurrence of a fault, namely the output of an input does not match the expected one. Thus, we believe that "fault" is okay. ==/ /** Dave [5]: OK. I think it may be good to be more explicit about the difference (that a failure occurs when the output of an input does not match the expected one) ... but I'll follow you. I suspect that T.Y. will also flag this*

A preliminary version of this paper was presented at the 36th Annual IEEE Computer Software and Applications Conference (COMPSAC 2012) [1].
C.-A. Sun, H. Dai, and G. Wang are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China. E-mail: casun@ustb.edu.cn.
D. Towey is with the School of Computer Science, University of Nottingham Ningbo China, Ningbo 315100, China. E-mail: dave.towey@nottingham.edu.cn
K.-Y. Cai is with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China. E-mail: kycai@buaa.edu.cn.
T.Y. Chen is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Hawthorn VIC 3122, Australia. Email: tychen@swin.edu.au.

*point. **/*

RT and PT are based on different intuitions, and each have their own advantages and disadvantages. Because it is likely that they can be complementary to each other, detecting different faults, it is intuitively appealing to investigate the their integration. Accordingly, Cai et al. [7] have proposed the random partition testing (RPT) strategy. In RPT, the input domain is first divided into m partitions, $s_1, s_2, \ldots, s_m$, where each s_i is allocated a probability p_i of selection. A partition s_i is randomly selected according to the testing profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$, where $p_1+p_2+\cdots+p_m = 1$. A concrete test case is then randomly selected from the chosen s_i.

In traditional RPT testing, the partitions and corresponding test profiles remain constant throughout testing, which may not be the best strategy. Independent researchers from various areas have observed that fault-revealing inputs tend to cluster into "continuous regions" [13], [14]—there is similarity in the execution behavior of neighboring software inputs. Based on software cybernetics, Cai et al. proposed dynamic random testing (DRT) [7], which aims to improve on both RT and RPT. Unlike the original RPT, where the values of p_i are fixed, DRT attempts to dynamically change the values: If a test case from a partition s_i reveals a fault, the corresponding p_i will be increased by a constant ε; otherwise, it is decreased by ε.

In practice, web services have usually been tested by the service providers, and simple or easy-to-test faults have been removed, meaning that the remaining faults are normally hard to detect. For ensuring a higher reliability of the web services, a simple RT strategy may not be an appropriate technique [15], especially when the scale is large, or there are some stubborn faults. Studies have shown that DRT can improve on RT in term of fault detection effectiveness [16]–[18].

In this paper, we present a dynamic random testing (DRT) approach for web services, as an enhanced version of RT that is an adaptation of DRT to the context of SOA. We examine key issues of such an adaption, and conduct empirical studies to evaluate the feasibility and effectiveness of the proposed DRT, with the experimental results showing DRT outperforms RT in terms of fault detection efficiency. The contributions of this work include:

- We develop an effective and efficient testing technique for web services. This includes a DRT framework that addresses key issues for testing web services, and a prototype that partly automates the framework.
- We evaluate the performance of DRT through a series of empirical studies on three real web services. These studies show that DRT has significantly higher fault-detection efficiency than RT and RPT. That is, to detect a given number of faults, DRT uses less time and fewer test cases than RT and RPT.
- We provide guidelines for the DRT parameter settings, supported by theoretical analysis, and validated by the empirical studies.

The rest of this paper is organized as follows. Section 2 introduces the underlying concepts for DRT, web services and mutation analysis. Section 3 presents the DRT framework for web services, guidelines for its parameter settings, and a prototype that partially automates DRT. Section 4 describes an empirical study where the proposed DRT is used to test three real-life web services, the results of which are summarized in Section 5. Section 6 discusses related work and Section 7 concludes the paper.

2 Background

In this section, we present some of the underlying concepts for DRT, web services, and mutation analysis.

2.1 Dynamic Random Testing (DRT)

DRT combines RT and PT [31], with the goal of benefitting from the advantages of both. Given a test suite *TS* classified into m partitions (denoted $s_1, s_2, \ldots, s_m$), suppose that a test case from s_i ($i = 1, 2, \ldots, m$) is selected and executed. If this test case reveals a fault, $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we then set

$$p'_j = \begin{cases} p_j - \dfrac{\varepsilon}{m-1} & \text{if } p_j \geq \dfrac{\varepsilon}{m-1} \\ 0 & \text{if } p_j < \dfrac{\varepsilon}{m-1} \end{cases}, \tag{1}$$

where ε is a probability adjusting factor, and then

$$p'_i = 1 - \sum_{\substack{j=1 \\ j\neq i}}^{m} p'_j. \tag{2}$$

Alternatively, if the test case does not reveal a fault, we set

$$p'_i = \begin{cases} p_i - \varepsilon & \text{if } p_i \geq \varepsilon \\ 0 & \text{if } p_i < \varepsilon \end{cases}, \tag{3}$$

and then for $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we set

$$p'_j = \begin{cases} p_j + \dfrac{\varepsilon}{m-1} & \text{if } p_i \geq \varepsilon \\ p_j + \dfrac{p'_i}{m-1} & \text{if } p_i < \varepsilon \end{cases}. \tag{4}$$

The detailed DRT algorithm is given in Algorithm 1. In DRT, the first test case is taken from a partition that has been randomly selected according to the initial probability profile $\{p_1, p_2, \ldots, p_m\}$ (Lines 2 and 3 in Algorithm 1). After each test case execution, the testing profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$ is updated by changing the values of p_i: If a fault is revealed, Formulas 1 and 2 are used; otherwise, Formulas 3 and 4 are used. The updated testing profile is then used to guide the random selection of the next test case (Line 8). This process is repeated until a termination condition is satisfied (Line 1). Examples of possible termination conditions include: "testing resources have been exhausted"; "a certain number of test cases have been executed"; and "a certain number of faults have been detected".

As can be seen from Formulas 1 to 4, updating the testing profile involves m simple calculations, thus requiring a constant time. Furthermore, the selection of partition s_i, and subsequent selection and execution of the test case, all also

Algorithm 1 DRT

Input: $\varepsilon, p_1, p_2, \ldots, p_m$

```
while termination condition is not satisfied
    Select a partition s_i according to the testing profile
    {⟨s_1, p_1⟩, ⟨s_2, p_2⟩, ..., ⟨s_m, p_m⟩}.
    Select a test case t from s_i.
    Test the software using t.
    if a fault is revealed by t
        Update p_j (j = 1, 2, ..., m and j ≠ i) and p_i
        according to Formulas 1 and 2.
    else
        Update p_j (j = 1, 2, ..., m and j ≠ i) and p_i
        according to Formulas 3 and 4.
    end_if
end_while
```

involve a constant time overhead. The execution time for one iteration of DRT is thus a constant, and therefore the overall time complexity for DRT to select n test cases is $O(m \cdot n)$.

2.2 Web Services

A web service is a platform-independent, loosely coupled, self-contained, programmable, web-enabled application that can be described, published, discovered, coordinated and configured using XML artifacts for the purpose of developing distributed interoperable applications [2]. A web service consists of a description (usually specified in WSDL) and implementation (that can be written in any programming language). Web services present their functionalities through published interfaces, and are usually deployed in a service container. Invocation of a web service requires analysis of the input message in its WSDL, test data generation based on its input parameters, and wrapping of test data in a SOAP message.

A web service is a basic component of SOA software, and, accordingly, the reliability of such SOA software depends heavily on the quality of the component web services. While testing is an obvious potential activity to help assuring the quality of web services, due to the unique features of SOA, web service testing can be more challenging than traditional software testing [5]. Some of these features include:

- *Lack of access to service implementation:* Normally, web service owners will not make source code of their web services accessible. Typically, service users only have access to the service interface defined in a WSDL file, which means that white-box testing approaches are not possible.
- *Incomplete documentation or specification:* A service provider may only offer an incomplete or inaccurate description of a service's functional and non-functional behavior. This makes it difficult to decide whether or not a test passes, especially when details about behavior or restrictions on implementations are missing [19].
- *Lack of control:* Unlike traditional software testing where testers can control the execution of software under test, there is usually no opportunity to intervene in the execution of the web service under test, which is often deployed in a remote service container.
- *Side effects caused by testing:* A large number of tests may introduce an additional communication load, and hence impact on the performance of the web service under test. This suggests that the number of tests should be kept as low as possible [20].

Although RT is a widely-used software testing method, some of its characteristics may make it inefficient for testing web services. This study explores the application of DRT to web services with an aim of improving on the fault detection efficiency of RT.

2.3 Mutation Analysis

Mutation analysis [12], [21]–[23] has been widely used to assess the adequacy of test suites and the effectiveness of testing techniques. Mutation operators are used to seed various faults into the program under test, and thus generate a set of variants, called mutants. If a test case causes a mutant to behave differently to the program under test (for example, by giving different output for the same input), then we say that this test case "kills" the mutant, and thus detects the injected fault. The mutation score (MS) is used to measure how thoroughly a test suite "kills" the mutants. The MS is defined as:

$$MS(p, ts) = \frac{N_k}{N_m - N_e} \tag{5}$$

where p is the program being mutated; ts is the test suite under evaluation; N_k is the number of mutants killed; N_m is the total number of mutants; and N_e is the number of equivalent mutants (mutants whose behavior is always the same as that of p). It has been highlighted that, compared with manually seeded faults, automatically generated mutants can be more similar to real-life faults, and thus the mutant score is a good indicator of the effectiveness of a testing technique [24]. In this paper, we use mutation analysis to evaluate the effectiveness of our proposed DRT for web services.

3 DRT for Web Services

In this section, we describe a framework for applying DRT to web services, discuss guidelines for DRT's parameter settings, and present a prototype that partially automates DRT for web services.

3.1 Framework

Considering the principle of DRT and the features of web services, we propose a DRT for web services framework, as illustrated in Figure 1. In the figure, the DRT components are inside the box, and the web services under test are located outside. Interactions between DRT components and the web services are depicted in the framework. We next discuss the individual framework components.

1 *WSDL Parsing*. Web services are composed of services and the relevant WSDL documents. By parsing the WSDL document, we can get the input information for each operation in the services. This includes the number of

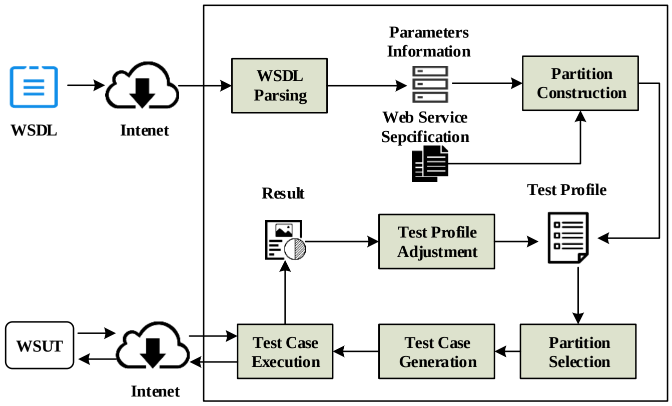


Fig. 1. DRT for web services framework

parameters, their names and types, and any additional requirements that they may have.

2 *Partition Construction*. Partition testing (PT) refers to a class of testing techniques that break the input domain into a number of partitions [25]. Because DRT is a black-box testing technique, combining RT and PT, the PT approaches used are at the specification level. Various approaches and principles for achieving convenient and effective partitions have been discussed in the literature [25]–[28]. The input domain of the web service under test (WSUT) can be partitioned based on the WSUT specifications and the parsed parameters. Once partitioned, testers can assign probability distributions to the partitions as an initial testing profile. This initial testing profile can be assigned in different ways, including using a uniform probability distribution, or one that sets probabilities according to the importance of the partition: For example, a partition within which faults were previously detected should be given higher priority.
3 *Partition Selection*. DRT randomly selects a partition according to the testing profile.
4 *Test Case Generation*. Given the selected partition s_i, a test case is then randomly and independently generated within s_i. Because the WSDL document has been parsed, generation of this test case is non difficult, and can be automated.
5 *Test Case Execution*. The relevant DRT component receives the generated test case, converts it into an input message, invokes the web service(s) through the SOAP protocol, and intercepts the test results (from the output message).
6 *Test Profile Adjustment*. Upon completion of each test, its pass or fail status is determined by comparing the actual and expected results (with the test passing if both are the same). The pass or fail status is then used to adjust the (partition) probability distribution accordingly. Situations where determination of the test outcome status is not possible (i.e. in the presence of the oracle problem [29]–[31]) may potentially be addressed using metamorphic testing [32].

Generally speaking, DRT test case generation is both in accordance with the probability distribution (for selection of the relevant partition), and with the principles of RT, taking advantage of the ease of RT and the effectiveness of PT. */** Dave [6]: are we sure that PT is effective? **/ /== Chang-ai [2]: ... Accordingly, we changed "effectiveness" to "practicability". Is this change okay? ==/ /** Dave [7]: I changed it back to effectiveness **/* Furthermore, many of the DRT for web services framework components can be automated. To make DRT for web services more efficient and practical, we developed a prototype that will be described in Section 3.3.

3.2 Guidelines for Parameter Setting

Our previous work [1] found that the DRT performance can be influenced by the number of partitions and the parameter ε. We next explore these impacts through a theoretical analysis, which, to be mathematically tractable, has the following assumptions:

1 The failure rate θ_i of each partition s_i ($i = 1, 2, \ldots, m$, and $m > 1$) is unknown, but can be estimated.
2 Each failure rate θ_i ($i = 1, 2, \ldots, m$, and $m > 1$) remains unchanged throughout the testing process (faults are not removed after their detection).
3 Test cases are selected with replacement, which means that same test cases may be selected more than once.

A principle of the DRT strategy is to increase the selection probabilities (by amount ε) of partitions with larger failure rates. In addition to the impact of the parameter ε, the number of partitions also influences the speed of updating the testing profile (Formulas 1 to 4). Therefore, for a given number of partitions, we are interested in investigating what values of ε yield the best DRT performance.

Letting θ_M denote the maximum failure rate, and s_M denote partitions with that failure rate, then p_i^n denotes the probability of executing the n^{th} test case from partition s_i. As testing proceeds, the probability p_M of partition s_M being selected is expected to increase:

$$p_M^{n+1} > p_M^n \tag{6}$$

Initially, the testing profile is $\{\langle s_1, p_1^0\rangle, \langle s_2, p_2^0\rangle, \ldots, \langle s_m, p_m^0\rangle\}$, which, after n test cases have been executed, is then updated to $\{\langle s_1, p_1^n\rangle, \langle s_2, p_2^n\rangle, \ldots, \langle s_m, p_m^n\rangle\}$. During the testing process, p_i^n is increased or decreased by the value ε, which is relatively small (set to 0.05 in previous studies [16], [18]). Because the initial p_i^0 is larger than ε, and the adjustment of p_i is relatively small (Formulas 1 to 4), the following two situations are rare, and thus not considered here: $p_i < \varepsilon/(m-1)$ or $p_i < \varepsilon$ ($i = 1, 2, \ldots, m$).

As part of exploring the relationship between p_i^{n+1} and p_i^n, we calculate the conditional probability, $p(i|\delta)$, of the following four situations (denoted δ_1, δ_2, δ_3, and δ_4, respectively):

Situation 1 (δ_1): **If** $t_n \notin s_i$ **and a fault is detected by** t_n, then $p(i|\delta_1)$ is calculated according to Formula 1:

$$p(i|\delta_1) = \sum_{i \neq j} \theta_j \left(p_i^n - \frac{\varepsilon}{m-1}\right).$$

Situation 2 (δ_2): **If** $t_n \in s_i$ **and a fault is detected by** t_n, then $p(i|\delta_2)$ is calculated according to Formula 2:

$$p(i|\delta_2) = \theta_i (p_i^n + \varepsilon).$$

Situation 3 (δ_3): **If $t_n \in s_i$ and no fault is detected by** t_n, then $p(i|\delta_3)$ is calculated according to Formula 3:

$$p(i|\delta_3) = (1-\theta_i)(p_i^n - \varepsilon).$$

Situation 4 (δ_4): **If $t_n \notin s_i$ and no fault is detected by** t_n, then $p(i|\delta_4)$ is calculated according to Formula 4:

$$p(i|\delta_4) = \sum_{i \neq j}(1-\theta_j)(p_i^n + \frac{\varepsilon}{m-1}).$$

Therefore, p_i^{n+1} for all cases together is:

$$\begin{aligned}
p_i^{n+1} =& p_i^n\theta_i(p_i^n+\varepsilon) + p_i^n(1-\theta_i)(p_i^n-\varepsilon) \\
&+ \sum_{j\neq i} p_j^n\theta_j(p_i^n - \frac{\varepsilon}{m-1}) \\
&+ \sum_{j\neq i} p_j^n(1-\theta_j)(p_i^n + \frac{\varepsilon}{m-1}) \\
=&(p_i^n)^2\theta_i + p_i^n\theta_i\varepsilon + (p_i^n)^2 - p_i^n\varepsilon - (p_i^n)^2\theta_i + p_i^n\theta_i\varepsilon \\
&+ (p_i^n - \frac{\varepsilon}{m-1})\sum_{j\neq i} p_j^n\theta_j + (p_i^n + \frac{\varepsilon}{m-1})\sum_{j\neq i} p_j^n \\
&- (p_i^n + \frac{\varepsilon}{m-1})\sum_{j\neq i} p_j^n\theta_j \\
=&(p_i^n)^2 + 2p_i^n\theta_i\varepsilon - p_i^n\varepsilon + (p_i^n - \frac{\varepsilon}{m-1} - p_i^n \\
&- \frac{\varepsilon}{m-1})\sum_{j\neq i} p_j^n\theta_j + (p_i^n + \frac{\varepsilon}{m-1})(1-p_i^n) \\
=&p_i^n + (p_i^n)^2 - (p_i^n)^2 + 2p_i^n\theta_i\varepsilon - p_i^n\varepsilon + \frac{\varepsilon}{m-1} - \\
&\frac{\varepsilon}{m-1}p_i^n - \frac{2\varepsilon}{m-1}\sum_{j\neq i} p_j^n\theta_j \\
=&p_i^n + \frac{\varepsilon}{m-1}(2p_i^n\theta_i m - p_i^n m - 2p_i^n\theta_i + 1) \\
&- \frac{2\varepsilon}{m-1}\sum_{j\neq i} p_j^n\theta_j \\
=&p_i^n + Y_i^n,
\end{aligned} \tag{7}$$

where

$$\begin{aligned}
Y_i^n =& \frac{\varepsilon}{m-1}(2p_i^n\theta_i m - p_i^n m - 2p_i^n\theta_i + 1) \\
&- \frac{2\varepsilon}{m-1}\sum_{j\neq i} p_j^n\theta_j.
\end{aligned} \tag{8}$$

From Formula 8, we have:

$$\begin{aligned}
Y_M^n - Y_i^n =& \frac{\varepsilon}{m-1}(2p_M^n\theta_M m - p_M^n m - 2p_M^n\theta_M + 1) \\
&- \frac{2\varepsilon}{m-1}\sum_{j\neq M} p_j^n\theta_j - \frac{\varepsilon}{m-1}(2p_i^n\theta_i m - p_i^n m \\
&- 2p_i^n\theta_i + 1) + \frac{2\varepsilon}{m-1}\sum_{j\neq i} p_j^n\theta_j \\
=& \frac{\varepsilon}{m-1}(2m(p_M^n\theta_M - p_i^n\theta_i) - m(p_M^n - p_i^n) - \\
&2(p_M^n\theta_M - p_i^n\theta_i)) - \sum_{j\neq M} p_j^n\theta_j + \sum_{j\neq i} p_j^n\theta_j \\
=& \frac{2\varepsilon}{m-1}(m(p_M^n\theta_M - p_i^n\theta_i) - \frac{m(p_M^n - p_i^n)}{2} - \\
&(p_M^n\theta_M - p_i^n\theta_i)) + \frac{2\varepsilon}{m-1}(p_M^n\theta_M - p_i^n\theta_i) \\
=& \frac{2\varepsilon}{m-1}(m(p_M^n\theta_M - p_i^n\theta_i) - \frac{m(p_M^n - p_i^n)}{2}).
\end{aligned} \tag{9}$$

Before presenting the final guidelines, we need the following lemma.

Lemma 1. If $p_i^n - p_M^n > 2(p_i^n\theta_i - p_M^n\theta_M)$, then $p_M^{n+1} > p_M^n$.

Proof: The condition $p_i^n - p_M^n > 2(p_i^n\theta_i - p_M^n\theta_M)$ can be equivalently expressed as:

$$\frac{p_M^n - p_i^n}{2} < p_M^n\theta_M - p_i^n\theta_i. \tag{10}$$

From Formula 10, $(p_M^n\theta_M - p_i^n\theta_i) - \frac{p_M^n - p_i^n}{2} > 0$, and because $0 < \varepsilon < 1$, and $m > 1$, therefore:

$$\frac{2m\varepsilon}{m-1}((p_M^n\theta_M - p_i^n\theta_i) - \frac{p_M^n - p_i^n}{2}) > 0. \tag{11}$$

Furthermore:

$$\frac{2\varepsilon}{m-1}(m(p_M^n\theta_M - p_i^n\theta_i) - \frac{m(p_M^n - p_i^n)}{2}) > 0. \tag{12}$$

According to Formulas 12 and 9, if $p_i^n - p_M^n > 2(p_i n\theta_i - p_M^n\theta_M)$, then $Y_M^n - Y_i^n > 0$.

Also, because $\sum_{i=1}^m p_i^{n+1} = 1$, and $\sum_{i=1}^m p_i^n = 1$, therefore $Y_M^n > 0$, and thus $\sum_{i=1}^m Y_i^n = 0$.

According to Formula 7, $p_M^{n+1} = p_M^n + Y_M^n$. Because $Y_M^n > 0$, therefore $p_M^{n+1} > p_M^n$. □

Accordingly, we can now present the following theorem that states a sufficient condition for achieving $p_M^{n+1} > p_M^n$.

Theorem 1. For failure rate $\theta_{min} = min\{\theta_1, \ldots, \theta_m\}$, $\theta_M > \theta_{min}$, if $0 < \theta_{min} < \frac{1}{2}$, the following condition is sufficient to guarantee that $p_M^{n+1} > p_M^n$:

$$\frac{2m\theta_{min}^2}{1-2\theta_{min}} < \varepsilon < \frac{(m-1)m\theta_{min}}{2(m+1)}. \tag{13}$$

Proof: In order to guarantee $p_M^{n+1} > p_M^n$, we consider the following three situations (where $i \in \{1, 2, \ldots, m\}$ and $i \neq M$).

Situation 1 ($p_i^n = p_M^n$): Because $\theta_i < \theta_M$, therefore $(p_i^n\theta_i - p_M^n\theta_M) < 0$.

Therefore, $(p_i^n - p_M^n) > 2(p_i^n\theta_i - p_M^n\theta_M)$.

According to Lemma 1, we have $p_M^{n+1} > p_M^n$.

Situation 2 ($p_i^n > p_M^n$): According to Formula 13, we have the following:

$$\varepsilon > \frac{2m\theta_{min}^2}{1-2\theta_{min}}.$$

Because

$$\frac{2m\theta_{min}^2}{1-2\theta_{min}} = \frac{\theta_{min}}{1/2m\theta_{min}-1/m},$$

we have the following:

$$\varepsilon > \frac{\theta_{min}}{1/2m\theta_{min}-1/m}.$$

Because $\theta_{min} < 1/2$, therefore $1/2m\theta_{min} - 1/m > 0$ and $\varepsilon(1/2m\theta - 1/m) > \theta_{min}$, which gives $\varepsilon/2m\theta_{min} > \theta_{min} + \varepsilon/m$.

Because $\varepsilon > 0$, and $m > 1$, therefore

$$\frac{1}{2\theta_{min}} > \frac{(\theta_{min}+\varepsilon/m)}{(\varepsilon/m)}.$$

$(1/2\theta_{min})(p_i^n - p_M^n) > (p_i^n - p_M^n)(\theta_{min} + \varepsilon/m)/(\varepsilon/m)$ as $p_i^n > p_M^n$, and

$$p_i^n - p_M^n > 2\theta_{min}(p_i^n - p_M^n)\frac{\theta_{min}+\varepsilon/m}{\varepsilon/m}.$$

Because $(\theta_{min} + \varepsilon/m)/(\varepsilon/m) > 1$, therefore

$$2\theta_{min}(p_i^n - p_M^n)\frac{\theta_{min}+\varepsilon/m}{\varepsilon/m} > 2\theta_{min}(p_i^n - p_M^n).$$

Because $\theta_{min} < \theta_M$, therefore

$$2\theta_{min}(p_i^n - p_M^n) > 2(p_i^n\theta_{min} - p_M^n\theta_M).$$

Thus,

$$p_i^n - p_M^n > 2(p_i^n\theta_{min} - p_M^n\theta_M).$$

According to Lemma 1, we have $p_M^{n+1} > p_M^n$.

Situation 3 ($p_i^n < p_M^n$): For this proof, we make the assumption that $\frac{1}{2} < \theta_M < 1$.

Because we have

$$\varepsilon < \frac{(m-1)m\theta_{min}}{2(m+1)}$$

and

$$\frac{(m-1)m\theta_{min}}{2(m+1)} = \frac{2m-(m+1)}{2(m+1)}m\theta_{min},$$

thus

$$\varepsilon < (\frac{m}{m+1} - \frac{1}{2})m\theta_{min}.$$

Obviously, $\varepsilon/m < (m/(m+1) - 1/2)\theta_{min}$ as $m > 1$.

Furthermore, we have

$$-\frac{\varepsilon}{m} > (\frac{1}{2} - \frac{m}{m+1})\theta_{min}$$

and

$$\frac{m\theta_{min}}{m+1} - \frac{\varepsilon}{m} + \frac{2\varepsilon}{m} > \frac{\theta_{min}}{2} + \frac{2\varepsilon}{m}$$

which means that

$$\frac{m\theta_{min}}{m+1} + \frac{\varepsilon}{m} > \frac{1}{2}(\theta_{min} + \frac{4\varepsilon}{m}).$$

It follows that

$$(m\theta_{min}/(m+1) + \varepsilon/m)/(4\varepsilon/m + \theta_{min}) > 1/2$$

for any $m > 1, \varepsilon > 0$, and $0 < \theta_{min} < 1$.

Because $\frac{1}{2} < \theta_M < 1$, therefore $(m\theta_{min}/(m+1) + \varepsilon/m)/(4\varepsilon/m + \theta_{min}) > 1/2\theta_M$.

Thus, we have

$$2(p_M^n - p_i^n)\theta_M \frac{\frac{\varepsilon}{m} + \frac{m\theta_{min}}{m+1}}{\frac{4\varepsilon}{m} + \theta_{min}} > p_M^n - p_i^n$$

as $p_M^n > p_i^n$.

Because $\varepsilon/m < 4\varepsilon/m$, and $m\theta_{min}/(m+1) < \theta_{min}$, therefore

$$\frac{\frac{\varepsilon}{m} + \frac{m\theta_{min}}{m+1}}{\frac{4\varepsilon}{m} + \theta_{min}} < 1$$

and

$$2(p_M^n - p_i^n)\theta_M > 2(p_M^n - p_i^n)\theta_M \frac{\frac{\varepsilon}{m} + \frac{m\theta_{min}}{m+1}}{\frac{4\varepsilon}{m} + \theta_{min}}$$

Hence we have

$$2(p_M^n - p_i^n)\theta_M > p_M^n - p_i^n,$$

which can be equivalently expressed as

$$p_i^n - p_M^n > 2(p_i^n - p_M^n)\theta_M.$$

Because $\theta_{min} < \theta_M$, therefore $2(p_i^n - p_M^n)\theta_M > 2(p_i^n\theta_{min} - p_M^n\theta_M)$, and thus

$$p_i^n - p_M^n > 2(p_i^n\theta_{min} - p_M^n\theta_M).$$

According to Lemma 1, we have $p_M^{n+1} > p_M^n$. □

In summary, when $\frac{1}{2} < \theta_M < 1$, there is always an interval E:

$$\varepsilon \in (\frac{2m\theta_{min}^2}{1-2\theta_{min}}, \frac{(m-1)m\theta_{min}}{2(m+1)}) \tag{14}$$

where $\theta_{min} \leq \theta_i, i \in \{1, 2, \ldots, m\}$, and $\theta_i \neq 0$, which can guarantee $p_M^{n+1} > p_M^n$.

From the proof above, it is clear that the value of θ_M affects the upper bound (E_{upper}) of E. When $\theta_{min} < \theta_M < \frac{1}{2}$, the value of E_{upper} should close to the lower bound of E. In practice, we should set

$$\varepsilon \approx \frac{2m\theta_{min}^2}{1-2\theta_{min}}. \tag{15}$$

3.3 Prototype

Figure 2 shows a screenshot of a prototype tool that partially automates DRT for web services. To start, testers input the address of the web service being tested (the URL of the WSDL), and press the Parse button to analyze the input and output formats. Next, an operation is selected from the operation list (in the bottom left). The tool provides two options for the partitions and test suites: either to manually specify the partitions (and test cases); or to upload the pre-defined partitions and test suites. Before beginning testing (by pressing the Test button), testers must set the maximum number of tests (Test Repetition Limit). During the testing, if a failure is detected before having executed the maximum

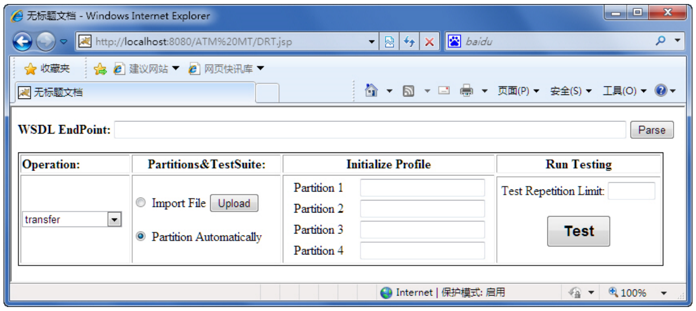


Fig. 2. Prototype interface

number of tests, then the tool suspends testing and asks for the tester's instruction. Testers can choose to remove defects and continue testing, or to stop testing. When all tests have completed, the test report is summarized and output in a file.

4 Empirical Study

We conducted a series of empirical studies to evaluate the performance of DRT.

4.1 Research Questions

In our experiments, we focused on addressing the following three research questions:

RQ1 How effective is DRT at detecting web service faults?
Fault-detection effectiveness is a key criterion for evaluating the performance of a testing technique. In our study, we chose three commonly used, real-life web services as subject programs, and applied mutation analysis to evaluate the effectiveness.

RQ2 How do the number of partitions and the DRT parameter ε impact on the failure detection effectiveness and efficiency of DRT?
In our earlier work [1], we found that the DRT parameter ε had a significant effect on DRT efficiency, and that the optimal value of the parameter could be related to the number of partitions. The relationship between ε and the number of partitions is examined through theoretical analysis, and verified through the empirical studies.

RQ3 What is the actual test case generation overhead when using the DRT strategy?
In Section 2.1, we have showed that DRT only requires linear time to generate test case. We wish to validate this theoretical finding through empirical examination of the actual test case generation and execution.

4.2 Subject Web Services

We selected three real-life web services as the subject programs for our study: `Aviation Consignment Management Service (ACMS)`, `China Unicom billing service (CUBS)`, and `Parking billing service (PBS)`. We used mutation analysis to generate a total of 1563 mutants. After removing equivalent mutants, we then also removed mutants that were too easily detected — deleting mutants that could be detected with less than 20 randomly generated test cases. Table 1 summarizes the basic information of the used web services and their mutants. A detailed description of each web service is given in the following.

TABLE 1
Studied Web Services

Web service	LOC	Number of mutants
ACMS	116	3
CUBS	131	11
PBS	129	4

TABLE 5
Plan C

	Plan details	Month charge (CNY)		
		$option_1$	$option_2$	$option_3$
Basic	Free calls (min)	260	380	550
	Free data (MB)	40	60	80
	Free data (MB)	Domestic (including video calls)		
Extra	Incoming calls (CNY/min)	0.25	0.20	0.15
	Data (CNY/KB)	0.0003		
	Video calls (CNY/min)	0.60		

4.2.1 Aviation Consignment Management Service (ACMC)

ACMS helps airline companies check the allowance (weight) of free baggage, and the cost of additional baggage. Based on the destination, flights are categorised as either domestic or international. For international flights, the baggage allowance is greater if the passenger is a student (30kg), otherwise it is 20kg. Each aircraft offers three cabins classes from which to choose (economy, business, and first), with passengers in different classes having different allowances. The detailed price rules are summarized in Table 2, where $price_0$ means economy class fare.

4.2.2 China Unicom Billing Service (CUBS)

CUBS provides an interface through which customers can know how much they need to pay according to cell-phone plans, calls, and data usage. The details of several cell-phone plans are summarized in Tables 3, 4, and 5.

4.2.3 Parking Billing Service (PBS)

Consider a parking billing service that accepts the parking details for a vehicle, including the vehicle type, day of the week, discount coupon, and hours of parking. This service rounds up the parking duration to the next full hour, and then calculates the parking fee for according to the hourly rates in Table 6. If a discount voucher is presented, a 50% discount off the parking fee is applied.

To facilitate better parking management, at the time of parking, customers may provide an estimation of parking duration, in terms of three different time ranges ($(0.0, 2.0]$, $(2.0, 4.0]$, and $(4.0, 24.0]$). If the estimation and actual parked hours fall into the same time range, then the customer will receive a 40% discount; but if they are different ranges, then a 20% markup is applied. A customer may choose to either use a discount coupon, or provide an estimation of parking duration, but may not do both.

TABLE 2
ACMC Baggage Allowance and Pricing Rules

	Domestic flights			International flights		
	First class	Business class	Economy class	First class	Business class	Economy class
Carry on (kg)	5	5	5	7	7	7
Free checked-in (kg)	40	30	20	40	30	20/30
Additional baggage pricing (kg)	$price_0 * 1.5\%$			$price_0 * 1.5\%$		

TABLE 3
Plan A

Plan details		Month charge (CNY)										
		$option_1$	$option_2$	$option_3$	$option_4$	$option_5$	$option_6$	$option_7$	$option_8$	$option_9$	$option_{10}$	$option_{11}$
Basic	Free calls (min)	50	50	240	320	420	510	700	900	1250	1950	3000
	Free data (MB)	150	300	300	400	500	650	750	950	1300	2000	3000
	Free incoming calls	Domestic (including video calls)										
Extra	Incoming calls (CNY/min)	0.25	0.20	0.15								
	Data (CNY/KB)	0.0003										
	Video calls (CNY/min)	0.60										

TABLE 4
Plan B

Plan details		Month charge (CNY)					
		$option_1$	$option_2$	$option_3$	$option_4$	$option_5$	$option_6$
Basic	Free calls (min)	120	200	450	680	920	1180
	Free data (MB)	40	60	80	100	120	150
	Free incoming calls	Domestic (including video calls)					
Extra	Incoming calls (CNY/min)	0.25	0.20	0.15			
	Data (CNY/KB)	0.0003					
	Video calls (CNY/min)	0.60					

TABLE 6
Hourly Parking Rates

Actual parking hours	Hourly parking rates					
	Weekday			Saturday and sunday		
	Motorcycle	Car: 2-door coupe	Car: others	Motorcycle	Car: 2-door coupe	Car: others
(0.0, 2.0]	$4.00	$4.50	$5.00	$5.00	$6.00	$7.00
(2.0, 4.0]	$5.00	$5.50	$6.00	$6.50	$7.50	$8.50
(4.0, 24.0]	$6.00	$6.50	$7.00	$8.00	$9.00	$10.00

(Obviously, a customer may also choose to neither provide an estimation, nor use a discount coupon.) No vehicles are allowed to remain parked for two consecutive days on a continuous basis.

4.3 Variables

4.3.1 Independent Variables

The independent variable in this study is the testing technique, DRT. RPT and RT were used as baseline techniques for comparison.

4.3.2 Dependent Variables

The dependent variable for RQ1 is the metric for evaluating the fault-detection effectiveness. Several effectiveness metrics exist, including: the P-measure [33] (the probability of at least one fault being detected by a test suite); the E-measure [34] (the expected number of faults detected by a test suite); the F-measure [35] (the expected number of test case executions required to detect the first fault); and the T-measure [36] (the expected number of test cases required to detect all faults). Since the F- and T-measures have been widely used for evaluating the fault-detection efficiency and effectiveness of DRT-related testing techniques [7], [9], [16]–[18], [36], they are also adopted in this study. We use F and T to represent the F-measure and the T-measure of a testing method. As shown in Algorithm 1, the testing process may not terminate after the detection of the first fault. Furthermore, because the fault detection information can lead to different probability profile adjustment mechanisms, it is also important to see what would happen after the first fault is revealed. For our study, therefore, we introduce the F2-measure [35], which is the number of additional test cases required to reveal the second fault after detection of the first fault. We use $F2$ to represent the F2-measure of a testing method, and $SD_{measure}$ to represent the standard deviation of metrics (where $measure$ can be F, $F2$, or T).

An obvious metric for RQ3 is the time required to detect faults. Corresponding to the T-measure, in this study we used T-$time$, the time required to detect all faults. F-$time$ and $F2$-$time$ denote the time required to detect the first fault, and the additional time needed to detect the second fault (after detecting the first), respectively.

For each of these metrics, smaller values indicate a better performance.

TABLE 7
Numbers of Partitions per Scheme for Three Subject Programs

Web service	Scheme 1	Scheme 2
ACMS	24	7
CUBS	20	3
PBS	18	3

4.4 Experimental Settings

4.4.1 Partitioning

In our study, we set the partitions by making use of a decision table (DT) [37]. */** Dave [8]: can we include the decision table in the paper? **/* A DT presents a large amount of complex decisions in a simple, straightforward manner, representing a set of decision rules under all exclusive conditional scenarios in a pre-defined problem. Typically, a DT consists of four parts:

1) The upper-left part lists the conditions denoted C_i ($i = 1, \ldots, n$, where n is the number of conditions in the pre-defined problem, and $n \geq 1$). Each condition C_i contains a set of possible options $O_{i,q} \in CO_i = \{O_{i,1}, \ldots, O_{i,t_i}\}$, where t_i is the number of possible options for C_i, and $q = \{1, \ldots, t_i\}$.
2) The upper-right part shows the condition space, which is a Cartesian product of all the CO_i ($SP(C) = CO_1 \times CO_2 \times \ldots \times CO_n$). Each element in the $SP(C)$ is a condition entry (CE) with the ordered n-tuple. */** Dave [9]: please rephrase the last sentence (I'm not sure of the intended meaning) **/*
3) The lower-left part shows all possible actions, represented A_j ($j = 1, \ldots, m$, where m is the number of possible actions and $m \geq 1$). Similar to CO_i, an action A_j contains a set of possible options $O'_{j,p} \in AO_j = \{O'_{j,1}, \ldots, O'_{j,k_j}\}$, where k_j is the number of alternatives for A_j, and $p = \{1, \ldots, k_j\}$.
4) The lower-right part shows the action space $SP(A)$, which is also a Cartesian product of all the AO_j ($SP(A) = AO_1 \times AO_2 \times \ldots \times AO_m$). Similar to *CE*, each element in the $SP(A)$ is an action entry (*AE*) with the ordered m-tuple. */** Dave [10]: please rephrase the last sentence (I'm not sure of the intended meaning) **/*

A DT *rule* is composed of a *CE* and its corresponding *AE*. With the constructed DT, it is possible to obtain partition schemes with different granularities. For fine-grain partition schemes, each *CE* of a DT *rule* corresponds to a partition; while for coarse-grained schemes, a partition corresponds to the union of a group of partitions of which all *CE* of DT *rules* have the same *AE*. Accordingly, we obtained two partition schemes for each subject web service: the numbers of partitions within these partition schemes are summarized in Table 7.

4.4.2 Initial Test Profile

Because test cases may be generated randomly during the test process, a feasible method is to use a uniform probability distribution as the initial testing profile. On the other hand, testers may also use past experience to guide a different probability distribution as the initial profile.

4.4.3 Constants

In the experiments, we were interested in exploring the relationship between the number of partitions and the DRT strategy parameter ε, and therefore selected a set of parameter values: $\varepsilon \in \{1.0E\text{-}05, 5.0E\text{-}05, 1.0E\text{-}04, 5.0E\text{-}04, 1.0E\text{-}03, 5.0E\text{-}03, 1.0E\text{-}02, 5.0E\text{-}02, 1.0E\text{-}01, 2E\text{-}01, 3E\text{-}01, 4E\text{-}01, 5E\text{-}01\}$. It should be noted that $\varepsilon = 5E\text{-}01$ is already a large value. Consider the following scenario. For PBS, when the test is carried out under partition scheme 2, if $\varepsilon = 7.5E\text{-}01$ and a uniform probability distribution is used as the testing profile (that is, $p_i = 1/3$) , then suppose that the first test case belonging to c_1 is executed and does not reveal any faults, then, according to Formula 3, the value of p_1 would become 0. It is important, therefore, that the initial value of ε should not be set too large.

4.5 Experimental Environment

Our experiments were conducted on a virtual machine running the Ubuntu 11.06 64-bit operating system, with two CPUs, and a memory of 2GB. The test scripts were written in Java. To ensure statistically reliable values [38] of the metrics (F-measure, F2-measure, T-measure, F-time, F2-time, and T-time), each testing session was repeated 30 times with 30 different seeds, and the average value calculated.

4.6 Threats To Validity

4.6.1 Internal Validity

A threat to internal validity is related to the implementations of the testing techniques, which involved a moderate amount of programming work. However, our code was cross-checked by different individuals, and we are confident that all techniques were correctly implemented.

4.6.2 External Validity

The possible threat to external validity is related to the subject programs and seeded faults under evaluation. Although the three subject web services are not very complex, they do implement real-life business scenarios of diverse application domains. Furthermore, 18 distinct faults were used to evaluate the performance. These faults cover different types of mutation operators and require an average of more than 20 randomly generated test cases to be detected. Although we have tried to improve the generalisability of the findings by applying different partitioning granularities, and 13 kinds of parameters, we cannot be certain as to whether or not similar results would be observed in other types of web services.

4.6.3 Construct Validity

The metrics used in our study are simple in concept and straightforward to apply, and hence there should be little threat to the construct validity.

4.6.4 Conclusion Validity

As has been reported for empirical studies in the field of software engineering [38], at least 30 observations are necessary to ensure the statistical significance of results. Accordingly, we have run a sufficient number of trials to ensure the reliability of our experimental results. Furthermore, as will be discussed in Section 5, further statistical tests were also conducted to confirm their significance.

TABLE 8
ACMS Results

Strategy		Partition scheme 1						Partition scheme 2					
		F	SD_F	F2	SD_{F2}	T	SD_T	F	SD_F	F2	SD_{F2}	T	SD_T
RT		13.30	10.34	14.90	24.64	41.80	27.13	13.30	10.34	14.90	24.64	41.80	27.13
RPT		12.04	11.13	12.26	19.08	35.31	25.95	9.03	8.24	15.13	16.08	34.29	29.53
DRT	1.0E-5	11.42	10.16	11.93	18.62	34.93	22.46	8.73	9.88	13.28	18.83	36.19	28.96
	5.0E-5	12.42	10.63	12.05	20.27	36.90	25.39	8.65	9.30	13.53	19.07	35.76	29.86
	1.0E-4	11.34	10.65	11.66	21.27	35.19	26.01	7.36	8.49	13.09	17.89	33.00	27.94
	5.0E-4	12.16	12.13	11.50	19.36	34.19	26.18	7.80	8.37	13.45	17.36	33.59	27.67
	1.0E-3	11.46	11.10	11.39	19.01	34.86	23.17	7.68	8.11	15.07	19.85	33.65	29.95
	5.0E-3	11.02	9.67	12.24	18.83	32.92	20.75	7.47	8.65	15.40	18.81	35.14	29.01
	1.0E-2	10.48	9.60	9.46	14.17	29.59	19.10	7.66	9.18	14.93	18.69	34.30	30.15
	5.0E-2	8.75	6.59	7.06	10.35	23.05	11.67	7.26	7.74	14.70	18.15	34.81	28.14
	1.0E-1	8.59	6.66	6.37	9.41	21.36	10.81	6.67	7.34	16.26	17.54	34.27	27.48
	2.0E-1	8.50	6.21	5.80	9.07	22.10	10.56	5.71	6.04	16.63	10.33	33.58	30.78
	3.0E-1	9.23	6.84	7.12	10.34	22.57	11.13	5.43	6.28	17.60	10.39	33.86	30.33
	4.0E-1	9.22	7.03	6.72	9.36	22.57	10.44	5.14	5.19	17.56	19.11	33.83	28.94
	5.0E-1	8.61	6.41	7.70	10.45	22.64	10.95	5.86	6.50	16.18	17.06	33.31	27.40

TABLE 9
CUBS Results

Strategy		Partition scheme 1						Partition scheme 2					
		F	SD_F	F2	SD_{F2}	T	SD_T	F	SD_F	F2	SD_{F2}	T	SD_T
RT		21.93	20.37	38.17	38.17	4203.07	3219.40	21.93	20.37	38.17	38.17	4203.07	3219.40
RPT		21.96	20.36	28.74	27.56	2590.38	1768.49	23.66	21.92	27.31	26.99	4195.72	2777.89
DRT	1.0E-5	21.21	23.24	25.62	23.29	2720.70	2051.85	22.90	22.51	25.29	28.21	4106.81	2589.29
	5.0E-5	19.50	19.94	25.77	24.78	2503.45	1873.76	23.88	23.25	26.68	26.84	4130.03	2588.36
	1.0E-4	20.71	22.51	25.59	26.00	2516.91	1843.11	23.55	22.76	26.71	30.26	4196.01	2247.57
	5.0E-4	21.79	21.10	26.82	28.21	2519.39	1942.65	25.27	28.74	26.11	24.36	4190.61	2753.74
	1.0E-3	20.95	20.79	31.41	31.54	2532.84	1752.56	23.84	25.44	27.34	27.64	4291.41	2884.39
	5.0E-3	22.32	21.90	25.93	26.48	2535.97	1572.42	24.11	23.27	26.80	25.20	4218.74	2887.01
	1.0E-2	22.47	21.55	26.01	23.85	2550.88	1873.01	23.46	25.01	26.74	26.43	4117.11	2798.92
	5.0E-2	21.61	20.04	27.66	29.12	2559.56	1777.16	24.01	24.32	26.52	27.04	4105.51	2570.57
	1.0E-1	21.72	21.71	28.31	28.91	2533.08	1774.39	23.30	24.45	26.07	27.91	4271.32	3011.37
	2.0E-1	21.71	21.83	28.68	32.75	2552.29	1879.60	23.55	25.20	28.25	30.31	4170.32	2796.61
	3.0E-1	22.82	21.65	26.68	31.28	2623.00	1770.36	23.40	25.52	27.35	27.84	4138.49	2594.70
	4.0E-1	23.34	24.02	27.32	27.42	2664.34	1886.21	23.07	24.08	29.18	30.39	4192.68	2706.73
	5.0E-1	22.18	22.32	27.14	28.54	2599.40	1640.05	23.30	23.63	26.98	27.71	4195.45	2535.00

TABLE 10
PBS Results

Strategy		Partition scheme 1						Partition scheme 2					
		F	SD_F	F2	SD_{F2}	T	SD_T	F	SD_F	F2	SD_{F2}	T	SD_T
RT		20.17	16.32	16.50	13.20	252.80	191.54	20.17	16.32	16.50	13.20	252.80	191.54
RPT		17.71	16.78	16.72	25.12	178.91	147.96	16.49	15.76	15.45	21.76	182.66	130.43
DRT	1.0E-5	18.07	19.08	14.27	21.70	176.18	146.49	15.24	15.06	15.31	23.97	163.51	135.48
	5.0E-5	18.23	18.34	15.39	24.35	176.93	143.93	15.13	15.35	14.30	23.57	160.82	116.59
	1.0E-4	16.39	17.42	13.96	21.04	171.61	142.79	15.14	15.05	13.96	22.31	159.93	121.90
	5.0E-4	16.75	17.56	12.60	21.98	165.94	140.33	14.95	12.66	14.98	24.52	166.18	125.20
	1.0E-3	17.96	18.93	15.40	22.22	164.56	138.74	15.72	16.03	16.30	24.41	166.27	128.63
	5.0E-3	16.93	17.23	14.34	22.34	160.70	117.02	15.54	12.79	15.45	23.70	170.14	129.61
	1.0E-2	17.12	16.60	15.10	22.68	166.15	136.59	15.19	15.02	15.13	25.78	166.60	124.97
	5.0E-2	17.02	18.70	16.46	24.75	168.20	129.72	17.10	17.23	15.07	24.09	170.09	130.88
	1.0E-1	17.09	16.78	14.12	22.00	172.94	153.50	16.02	17.16	15.98	24.97	174.16	134.34
	2.0E-1	17.45	18.36	14.14	22.77	174.71	139.02	15.27	15.54	15.52	23.86	167.07	132.81
	3.0E-1	17.23	18.45	16.95	27.43	178.09	161.44	15.43	15.54	15.15	22.04	175.02	136.66
	4.0E-1	17.05	17.72	16.21	20.08	169.21	145.43	15.28	15.93	15.28	24.48	164.67	124.24
	5.0E-1	17.17	18.26	16.23	25.83	172.29	134.94	16.10	15.78	15.26	23.80	167.21	124.05

5 Experimental Results

5.1 RQ1: Fault Detection Effectiveness

The F-, F2-, and T-measure results are summarized in Tables 8 to 10, and their distributions for each program are displayed using boxplots in Figures 3 to 5. In each boxplot, the upper and lower bounds of the box represent the third and first quartiles of the metric, respectively; the middle line represents the median value; the upper and lower whiskers mark, respectively, the largest and smallest data within the range of $\pm 1.5 \times IQR$ (where IQR is the interquartile range); outliers beyond the IQR are denoted with hollow circles; and each solid circle represents the mean value of the metric.

It can observed from the tables and figures that, in general, DRT is the best performer, followed by RPT. We also conducted statistical testing to verify the significance of this observation, using the Holm-Bonferroni method [35] (with p-value equal to 0.05) to determine which pairs of testing techniques had significant differences. The statistical data are shown in Tables 11 to 13, where each cell gives the number of scenarios where the technique above (in the table) performed better than one to the left. Where the difference is significant, the number is underlined and in bold face. For example, the **75** in the top right cell of Table 13 indicates that, of 78 scenarios (13 parameters $\times$ two partition schemes $\times$ three web services), DRT had lower F2-measure scores than RT for 75, with the fault-detection capabilities of these two techniques being significantly different.

TABLE 11
Number of Scenarios Where the Technique on the Top Row Has a Lower F-measure Score Than That on the Left Column

	RT	RPT	DRT
RT	—	4	**60**
RPT	2	—	**62**
DRT	**18**	**16**	—

TABLE 12
Number of Scenarios Where the Technique on the Top Row Has a Lower F2-measure Score Than That on the Left Column

	RT	RPT	DRT
RT	—	4	**69**
RPT	2	—	**63**
DRT	**9**	**15**	—

TABLE 13
Number of Scenarios Where the Technique on the Top Row Has a Lower T-measure Score Than That on the Left Column

	RT	RPT	DRT
RT	—	6	**75**
RPT	0	—	**64**
DRT	**3**	**14**	—

Tables 11 to 13 clearly show that the difference between each pair of testing techniques is always significantly different.

5.2 RQ2: Relationship between Partition Number and ε

In 3.2, we analyzed the relationship between the number of partitions and the DRT strategy parameter ε. In this section, we show that our theoretical analysis provides useful guidance to testers to set the value of ε.

TABLE 14
Theoretical Optimal Values of DRT Parameter

Web service	Partition scheme	θ_{min}	ε^*
ACMS	1	5.452E-2	1.601E-1
	2	2.797E-3	1.102E-4
CUBS	1	1.193E-3	5.702E-5
	2	1.397E-3	1.734E-5
PBS	1	1.760E-3	1.118E-4
	2	1.492E-3	1.340E-5

We used three web services to validate our theoretical analysis. Before starting the test, the failure rate θ_i of partition s_i was obtained by executing (k) test cases from s_i until revealing a fault, then $\theta_i = k/k_i$, where k_i is the total number of test cases in s_i. According to Formula 19, the theoretically optimal values of ε in each scenario for each web service are shown in Table 14, where ε^* denotes the theoretical value of ε. We ran a series of experiments with the parameters set according to those in Table 14: The F-, F2-, and T-measure results for each program are shown in Figure 6, where ε_1^* and ε_2^* denote the theoretical values of parameter ε in the two different partition schemes, respectively. For ease of presentation and understanding, we used $log_{100}(1.0E05 \times \varepsilon)$ for the horizontal axis in Figure 6. Apart from the DRT strategy parameter ε, all other experimental settings remained the same as in Section 5.1.

From Figure 6, we have the following observations:

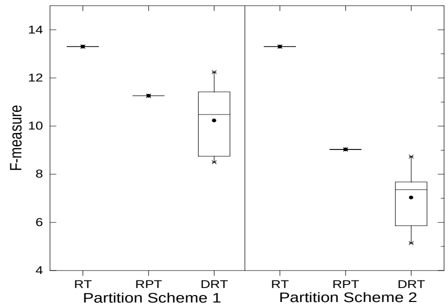
- In most scenarios, the DRT strategy with theoretically optimum parameter value performs best. Furthermore, the DRT strategy performs better when the parameter values are near the theoretically optimum value than when not.
- From Figure 6 (a), it can be observed that the DRT strategy with larger parameter values performs better than with the theoretically optimum value, in terms of the F-measure. The main reason for this is that, for this scenario, the maximum failure rate ($\theta_M = 4.781E-3$) is large and the number of partitions is small: When the parameter value is large, the probability of selecting partitions with lower failure rates is quickly reduced, and the probability of selecting partitions with larger failure rates is quickly increased, according to Formulas 3 and 4.

5.3 RQ3: Selection Overhead

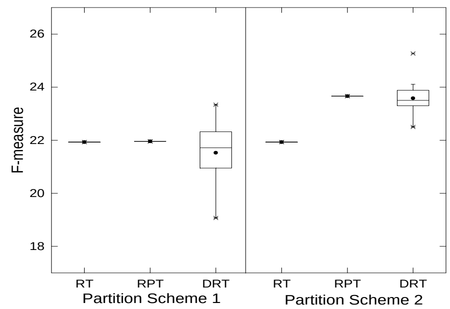
Tables 15 to 17 summarize the F-, F2-, and T-time results, respectively, and their distributions for each web service is shown in Figures 7 to 9. It can be observed from the figures that, in general, DRT had the best performance, and RPT just marginally outperforms RT.

As was done for the F-, F2-, and T-measure data, we used the Holm-Bonferroni method to check the difference between each pair of testing strategies in terms of F-time, F2-time, and T-time, as shown in Tables 18 to 20.

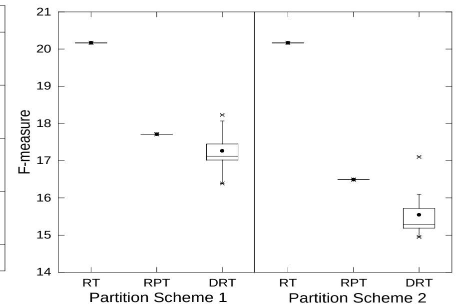
In Table 20, four entries (″**61**″ & ″**17**″ for DRT vs. RT, ″**57**″ vs. ″**21**″ for DRT vs. RPT) are in bold font and underline, meaning that, in terms of T-time, DRT was significantly



(a) ACMS

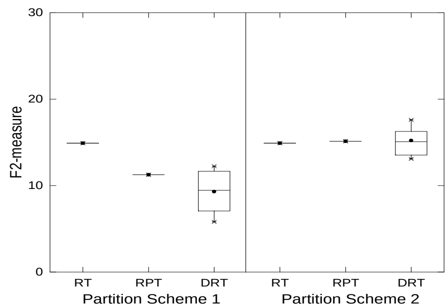


(b) CUBS

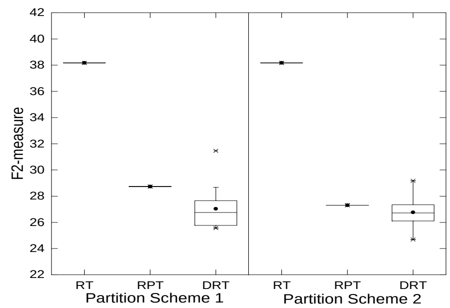


(c) PBS

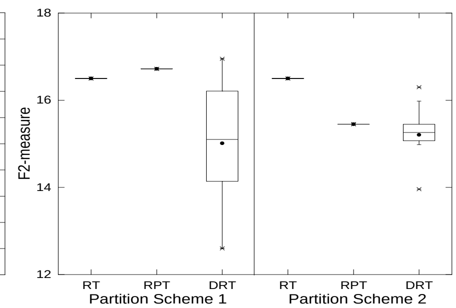
Fig. 3. F-measure boxplots for each program



(a) ACMS

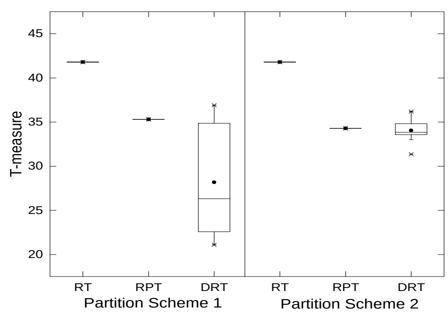


(b) CUBS

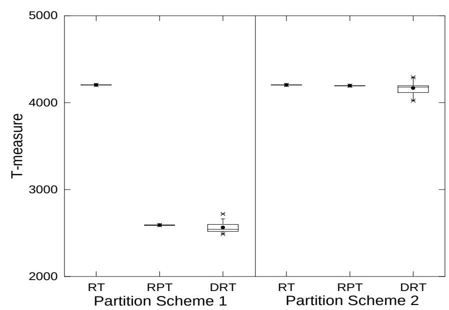


(c) PBS

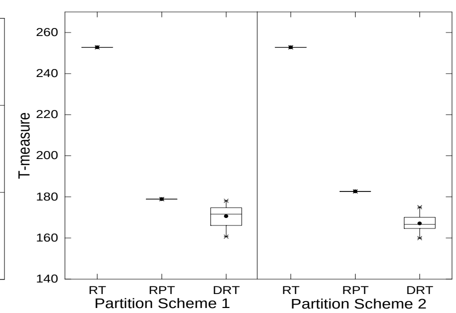
Fig. 4. F2-measure boxplots for each program



(a) ACMS

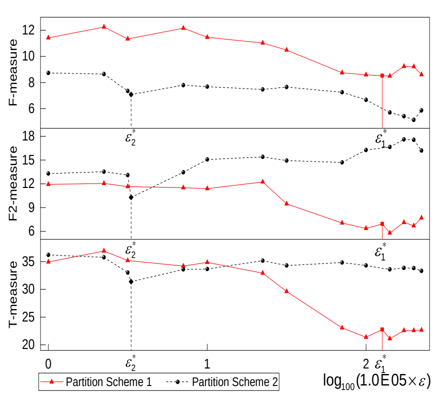


(b) CUBS

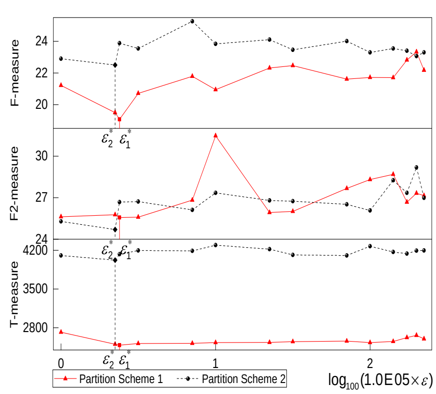


(c) PBS

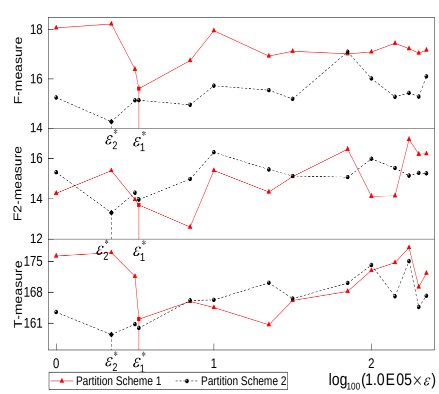
Fig. 5. T-measure boxplots for each program



(a) ACMS

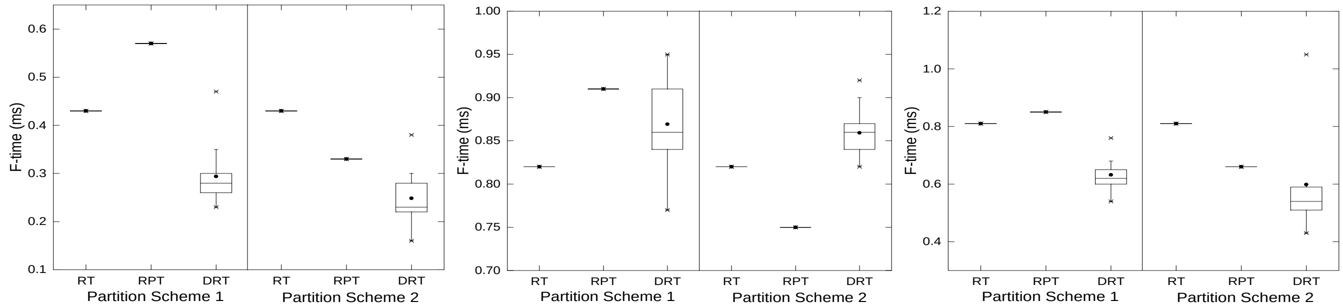


(b) CUBS



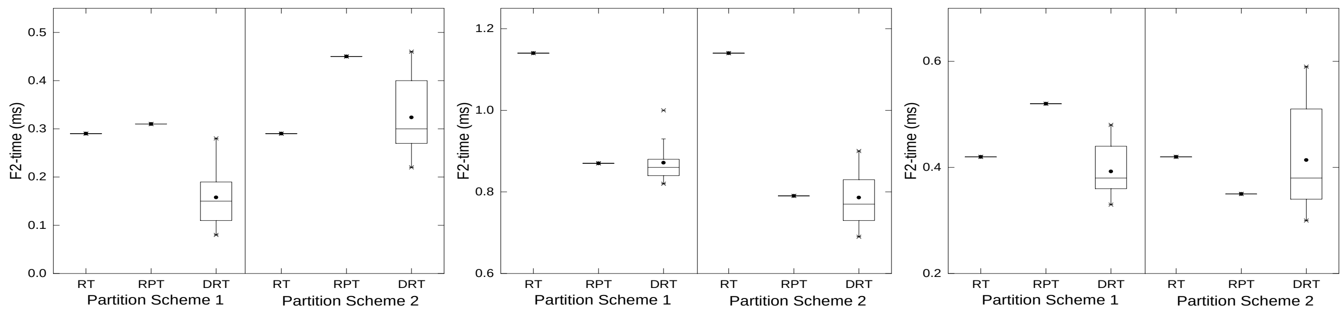
(c) PBS

Fig. 6. Line charts of F-measure, F2-measure, and T-measure values for each program (for both the theoretically optimum parameter value, and other values



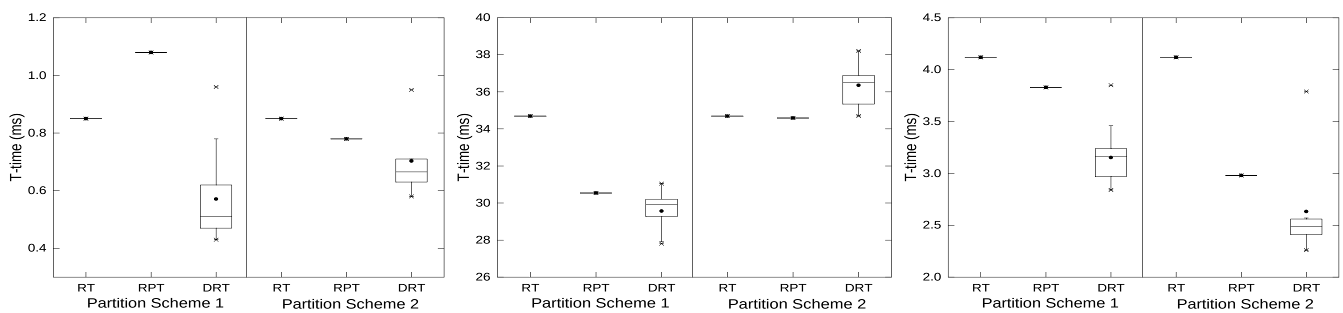
(a) ACMS (b) CUBS (c) PBS

Fig. 7. F-time boxplots for each program



(a) ACMS (b) CUBS (c) PBS

Fig. 8. F2-time boxplots for each program



(a) ACMS (b) CUBS (c) PBS

Fig. 9. T-time boxplots for each program

better than RT, and DRT only marginally outperformed RPT. Similar observations can be made regarding the F-time and F2-time results. In other words, the additional computation incurred in DRT by updating the test profile is compensated for in terms of test execution savings.

It can also be observed from Tables 18 to 20 that DRT only slightly outperformed RPT, but that DRT was significantly better than RT, especially in term of $T\text{-}time$.

In summary, the DRT strategy is considered the best testing technique across all six metrics, and RPT marginally outperformed RT.

6 Related Work

In this section, we describe related work from two perspectives: related to testing techniques for web services; and related to improving RT and PT.

6.1 Testing Techniques for Web Services

In recent years, a lot of effort has been made to test web services [5], [15], [39], [40]. Test case generation, involving both the generation and selection of test cases, is core to testing web services, and model-based [41] and specification-based [42] techniques are two common approaches. Before making services available on the Internet, testers can use model-based techniques to verify whether or not the behavior of the WSUT meets their requirements. In these techniques, test data can be generated from a data model that specifies the inputs to the software—this data model can be built before, or in parallel to, the software development process. Verification methods using technologies such as theorem-proving [43], models [44] and Petri-Nets [45] also exist. Sinha et al. [43], for example, used theorem-proving to generate test cases, making use of existing test generation methods based on extended finite state machine (EFSM) specifications. Paradkar et al. [44] proposed a model-based

TABLE 15
F-time, F2-time and T-time for Web Service ACMS (in ms)

Strategy		Partition scheme 1			Partition scheme 2		
		F-time	F2-time	T-time	F-time	F2-time	T-time
RT		0.43	0.29	0.85	0.43	0.29	0.85
RPT		0.57	0.31	1.08	0.33	0.45	0.78
DRT	1.0E-5	0.47	0.28	0.96	0.38	0.41	0.95
	5.0E-5	0.35	0.21	0.78	0.30	0.27	0.71
	1.0E-4	0.33	0.19	0.66	0.24	0.28	0.65
	5.0E-4	0.30	0.17	0.60	0.27	0.25	0.67
	1.0E-3	0.29	0.19	0.61	0.22	0.24	0.60
	5.0E-3	0.28	0.23	0.62	0.21	0.31	0.64
	1.0E-2	0.24	0.17	0.53	0.23	0.29	0.66
	5.0E-2	0.28	0.13	0.49	0.23	0.35	0.71
	1.0E-1	0.26	0.08	0.43	0.23	0.29	0.63
	2.0E-1	0.23	0.12	0.43	0.30	0.46	0.89
	3.0E-1	0.24	0.11	0.45	0.18	0.40	0.69
	4.0E-1	0.27	0.10	0.47	0.16	0.31	0.58
	5.0E-1	0.28	0.11	0.47	0.28	0.45	0.86

TABLE 16
F-time, F2-time and T-time for Web Service CUBS (in ms)

Strategy		Partition scheme 1			Partition scheme 2		
		F-time	F2-time	T-time	F-time	F2-time	T-time
RT		0.82	1.14	34.69	0.82	1.14	34.69
RPT		0.91	0.87	30.54	0.75	0.79	34.59
DRT	1.0E-5	0.91	0.87	30.14	0.87	0.83	36.49
	5.0E-5	0.95	0.86	30.21	0.82	0.75	34.70
	1.0E-4	0.77	0.89	29.27	0.86	0.69	35.34
	5.0E-4	0.79	1.00	31.05	0.87	0.90	36.85
	1.0E-3	0.86	0.88	29.93	0.92	0.77	36.44
	5.0E-3	0.86	0.83	30.28	0.83	0.83	36.27
	1.0E-2	0.95	0.88	29.33	0.84	0.72	35.23
	5.0E-2	0.84	0.82	30.56	0.88	0.72	37.07
	1.0E-1	0.88	0.93	29.45	0.82	0.76	37.14
	2.0E-1	0.83	0.83	30.16	0.86	0.73	36.79
	3.0E-1	0.95	0.86	27.81	0.84	0.84	36.89
	4.0E-1	0.84	0.84	28.23	0.86	0.82	35.22
	5.0E-1	0.87	0.84	27.91	0.90	0.86	38.21

test data generation technique for semantic web services, where test cases are generated using pre-defined fault-models and the IOPE (Inputs, Outputs, Preconditions, Effects paradigm) information from semantic specifications. Dong et al. [46] proposed a web service testing technique based on fault coverage, in which a High-level Petri Net (HPN) is first constructed by analyzing the WSDL specification for each operation, and HPNs for all operations are merged into one for the service. Then, a UIO (Unique Input Output) sequence is generated based on the resulting HPN through graph transformation. Finally, a test case for each UIO sequence is generated to meet the fault-coverage requirements. When testing web services, because it is often only the service's specification that users can receive, specification-based testing is a natural choice. Typically, the web service specification is contained in the web service description language (WSDL) document, which provides information about the available operations and parameters. Many methods proposed for WSDL-based test data generation are based on the XML schema data type. Hanna and Munro proposed a framework that can be used to test the robustness of a web service [47]. Their framework analyzes the WSDL documents of web services to identify what faults could impact the robustness, facilitating the design of test cases to detect those faults.

The approaches listed above all aim to generate test cases without consideration of the impact of test case execution order on test efficiency. In contrast, Bertolino et al. [48] proposed using the category-partition method [49] with XML schemas to perform XML-based partition testing. Because PT aims to find subsets of all possible test cases to adequately test a system, it can help reduce the required number of test cases. Our proposed approach involved using software cybernetics with PT: In DRT, selection of a partition is done according to the testing profile, which is updated throughout the test process. An advantage of DRT is that partitions with larger failure rates have higher probabilities of selection. Zhu and Zhang [50] proposed a collaborative testing framework, where test tasks are completed using collaborating test services—a test service is a service assigned to perform a specific testing task. Our framework (Section 3.1) aims to find more faults in the

TABLE 17
F-time, F2-time and T-time for Web Service PBS (in ms)

Strategy		Partition scheme 1			Partition scheme 2		
		F-time	F2-time	T-time	F-time	F2-time	T-time
RT		0.81	0.42	4.12	0.81	0.42	4.12
RPT		0.85	0.52	3.83	0.66	0.35	2.98
DRT	1.0E-5	0.76	0.48	3.85	1.05	0.51	3.44
	5.0E-5	0.73	0.35	3.02	0.91	0.52	3.79
	1.0E-4	0.54	0.36	2.97	0.49	0.34	2.26
	5.0E-4	0.68	0.34	3.20	0.50	0.30	2.31
	1.0E-3	0.62	0.38	2.87	0.52	0.38	2.42
	5.0E-3	0.60	0.38	2.99	0.43	0.51	2.40
	1.0E-2	0.58	0.42	2.84	0.51	0.34	2.41
	5.0E-2	0.64	0.44	3.27	0.59	0.36	2.49
	1.0E-1	0.65	0.38	3.46	0.60	0.38	2.56
	2.0E-1	0.60	0.36	3.22	0.54	0.59	2.47
	3.0E-1	0.59	0.44	3.24	0.57	0.46	2.57
	4.0E-1	0.61	0.33	2.90	0.51	0.33	2.54
	5.0E-1	0.62	0.44	3.16	0.56	0.36	2.56

TABLE 18
Number of Scenarios Where The Technique on the Top Row Has a Lower F-time Than That on the Left Column

	RT	RPT	DRT
RT	—	5	**53**
RPT	1	—	**59**
DRT	**25**	**19**	—

TABLE 19
Number of Scenarios Where the Technique on the Top Row Has a Lower F2-time Than That on the Left Column

	RT	RPT	DRT
RT	—	3	**62**
RPT	3	—	**57**
DRT	**16**	**21**	—

TABLE 20
Number of Scenarios Where the Technique on the Top Row Has a Lower T-time Than That on the Left Column

	RT	RPT	DRT
RT	—	5	**61**
RPT	1	—	**57**
DRT	**17**	**21**	—

WSUT, with the result of the current test case execution providing feedback to the control system so that the next test case selected has a greater chance to reveal faults.

Most existing web service testing techniques assume that the computed output for each test case is verifiable, something that is not always true in practice. The oracle problem [30], [31] refers to those situations where the test case output is not verifiable, and has meant that many testing techniques may not be applicable in some situations. To address the outstanding oracle problem for testing web services, a metamorphic testing [51], [52] technique has been proposed that not only alleviates the oracle problem, but also presents a feasible and efficient option for testing web services. Sun et al. proposed a metamorphic testing framework for web services [32] and conducted a case study that showed that up to 94.1% of seeded faults could be detected without the need for oracles.

6.2 Improving RT and PT

Based on the observation that failure-causing inputs tend to cluster into contiguous regions within the input domain [13], [14], significant work has been done aiming to improve RT [7], [8]. Adaptive random testing [8] is a family of advanced techniques based on random testing that aim to improve the failure-detection effectiveness by evenly spreading test cases throughout the input domain. One well-known ART approach, FSCS-ART */** Dave [11]: Perhaps cite: T. Y. Chen, H. Leung, and I. K. Mak. "Adaptive Random Testing". In Michael J. Maher, editor, ASIAN, volume 3321 of Lecture Notes in Computer Science, pages 320?329. Springer, 2004. **/* , selects a next test input from the fixed-size candidate set of tests that is farthest from all previously executed tests [53]. Many other ART algorithms have also been proposed, including RRT [54], [55] */** Dave [12]: can we (also) cite: Kwok Ping Chan, Tsong Yueh Chen, and Dave Towey (2006), "Restricted Random Testing: Adaptive Random Testing by Exclusion", International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol. 16, No. 4, pp.553-584. **/* , DF-FSCS [22], and ARTsum [56], with their effectiveness examined and validated through simulations and experiments.

Adaptive testing (AT) [9], [57], [58] takes advantage of feedback information to control the execution process, and has been shown to outperforms RT and RPT in terms of the T-measure and the number of detected faults, which means that AT has higher efficiency and effectiveness than RT and RPT. */** Dave [13]: I modified the previous sentence. Please confirm that I have the correct intended meaning. **/* However, AT may require a very long execution time in practice. To alleviate this, Cai et al. [7] proposed DRT, which uses testing information to dynamically adjust the testing profile. There are several things that can impact on DRT's test efficiency. Yang et al. [17] proposed A-DRT, which adjusts parameters during the testing process. Li et al. [18] developed O-DRT, which uses an objective function and a pre-defined parameter f which, when exceeded during the testing process (by the objective function), results in the test profile being adjusted to a theoretically optimal one. Lv et al. [16] introduced two parameters for adjusting the selection probability of different partitions: */** Dave [14]: Please confirm that my change in the previous part of this sentence ("adjusting the selection probability") is correct **/* ε adjusts the probability of partitions where a test case detects a fault; and δ adjusts the probability of partitions where a test case does not detect a fault. (ε should be greater than δ). Furthermore, they recommended setting the ε and δ values to have an interval of ε/δ. However, it is unclear how to set ε and δ individually. */** Dave [15]: are we sure that it is unclear how to set ε and δ individually? **/* In this study, we only considered one parameter (i.e. ε */** Dave [16]: is our ε a different meaning to that of Lv et al. [16]? If so, should we explain explicitly here? **/*) in the DRT algorithm for adjusting the probability of different partitions, and provided detailed settings guidance.

7 Conclusion

In this paper, to address the challenges of testing SOA-based applications, we have presented a dynamic random testing (DRT) technique for web services. Our technique uses random testing to generate test cases, and selects test cases for execution from different partitions in accordance with a testing profile that is dynamically updated in response to the test data collected. In this way, the proposed test technique includes benefits form both random testing and partition testing.

We proposed a framework that examines key issues when applying DRT to testing web services, and developed a prototype to make the method feasible and effective. To help guide testers to correctly set the DRT parameters, we used a theoretical analysis to identify the relationships between the number of partitions (m) and the probability adjusting factor (ε). Three real web services were used as experimental subjects to validate the feasibility and effectiveness of our approach. The results of the empirical study show that, in general, DRT has better performance than both RT and RPT, in terms of the F-, F2-, and T-measures, and always outperforms when the ε settings follow our guidelines. In other words, our theoretical analysis can provide genuinely useful guidance when DRT is used.

In our future work, we plan to conduct experiments on more web services to further validate its effectiveness, and identify the limitations of our method.

Acknowledgment

This research is supported by the National Natural Science Foundation of China (under Grant No. 61872039), the Beijing Natural Science Foundation (Grant No. 4162040), the Aeronautical Science Foundation of China (Grant No. 2016ZD74004), and the Fundamental Research Funds for the Central Universities (Grant No. FRF-GF-17-B29).

References

[1] C.-A. Sun, G. Wang, K.-Y. Cai, and T. Y. Chen, "Towards dynamic random testing for web services," in *Proceedings of the 36th IEEE International Computer Software and Applications Conference (COMPSAC'12)*, 2012, pp. 164–169.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223–255, 2008.

[3] C.-A. Sun, E. el Khoury, and M. Aiello, "Transaction management in service-oriented systems: requirements and a proposal," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 167–180, 2011.

[4] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti, "Whitening soa testing," in *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2009, pp. 161–170.

[5] G. Canfora and M. D. Penta, "Service-oriented architectures testing: a survey," *Lecture Notes in Computer Science*, vol. 5413, pp. 78–105, 2009.

[6] R. Hamlet, "Random testing," in *Encyclopedia of Software Engineering*. John Wiley and Sons, Inc., 2002.

[7] K.-Y. Cai, H. Hu, C. Jiang, and F. Ye, "Random testing with dynamically updated test profile," in *Proceedings of the 20th International Symposium On Software Reliability Engineering (ISSRE'09)*, 2009, pp. 1–2.

[8] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: the art of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[9] K.-Y. Cai, B. Gu, H. Hu, and Y.-C. Li, "Adaptive software testing with fixed-memory feedback," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1328–1348, 2007.

[10] T. Y. Chen, F.-C. Kuo, and H. Liu, "Adaptive random testing based on distribution metrics," *Journal of Systems and Software*, vol. 82, no. 9, pp. 1419–1433, 2009.

[11] T. Y. Chen, F.-C. Kuo, H. Liu, and W. E. Wong, "Code coverage of adaptive random testing," *IEEE Transactions on Reliability*, vol. 62, no. 1, pp. 226–237, 2013.

[12] J. Chen, L. Zhu, T. Y. Chen, D. Towey, F.-C. Kuo, R. Huang, and Y. Guo, "Test case prioritization for object-oriented software: an adaptive random sequence approach based on clustering," *Journal of Systems and Software*, vol. 135, pp. 107–125, 2018.

[13] P. E. Ammann and J. C. Knight, "Data diversity: an approach to software fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 418–425, 1988.

[14] G. B. Finelli, "NASA software failure characterization experiments," *Reliability Engineering and System Safety*, vol. 32, no. 1, pp. 155–169, 1991.

[15] Y.-F. Li, P. K. Das, and D. L. Dowe, "Two decades of web application testingła survey of recent advances," *Information Systems*, vol. 43, pp. 20–54, 2014.

[16] J. Lv, H. Hu, and K.-Y. Cai, "A sufficient condition for parameters estimation in dynamic random testing," in *Proceedings of the 35th IEEE Annual International Computer Software and Applications Conference Workshops (COMPSACW'11)*, 2011, pp. 19–24.

[17] Z. Yang, B. Yin, J. Lv, K.-Y. Cai, S. S. Yau, and J. Yu, "Dynamic random testing with parameter adjustment," in *Proceedings of the 38th IEEE Annual International Computer Software and Applications Conference Workshops (COMPSACW'14)*, 2014, pp. 37–42.

[18] Y. Li, B. B. Yin, J. Lv, and K.-Y. Cai, "Approach for test profile optimization in dynamic random testing," in *Proceedings of the 39th IEEE Annual International Computer Software and Applications Conference (COMPSAC'15)*, 2015, pp. 466–471.

[19] C.-A. Sun, M. Li, J. Jia, and J. Han, "Constraint-based model-driven testing of web services for behavior conformance," in *Proceedings of the 16th International Conference on Service Oriented Computing (ICSOC'18)*, 2018, pp. 543–559.

[20] C.-A. Sun, Z. Wang, Q. Wen, P. Wu, and T. Y. Chen, "An empirical study on iterative metamorphic testing for web services," *International Journal on Software Tools for Technology Transfer, under review*.

[21] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[22] C. Mao, T. Y. Chen, and F.-C. Kuo, "Out of sight, out of mind: a distance-aware forgetting strategy for adaptive random testing," *Science China Information Sciences*, vol. 60, no. 9, pp. 092 106:1–092 106:21, 2017.

[23] J. Chen, F.-C. Kuo, T. Y. Chen, D. Towey, C. Su, and R. Huang, "A similarity metric for the inputs of oo programs and its application in adaptive random testing," *IEEE Transactions on Reliability*, vol. 66, no. 2, pp. 373–402, 2017.

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

[25] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 703–711, 1991.

[26] K.-Y. Cai, T. Jing, and C.-G. Bai, "Partition testing with dynamic partitioning," in *Proceedings of the 29th International Computer Software and Applications Conference (COMPSAC'05)*, 2005, pp. 113–116.

[27] T. Y. Chen and Y.-T. Yu, "On the relationship between partition and random testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, pp. 977–980, 1994.

[28] ——, "On the expected number of failures detected by subdomain testing and random testing," *IEEE Transactions on Software Engineering*, vol. 22, no. 2, pp. 109–119, 1996.

[29] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[30] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: a survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[31] K. Patel and R. M. Hierons, "A mapping study on testing non-testable systems," *Software Quality Journal*, vol. 26, no. 4, pp. 1373–1413, 2018.

[32] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "Metamorphic testing for web services: framework and a case study," in *Proceedings of the 9th IEEE International Conference on Web Services (ICWS'11)*, 2011, pp. 283–290.

[33] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *IEEE transactions on Software Engineering*, no. 4, pp. 438–444, 1984.

[34] T. Y. Chen and Y.-T. Yu, "Optimal improvement of the lower bound performance of partition testing strategies," *IEE Proceedings-Software Engineering*, vol. 144, no. 5, pp. 271–278, 1997.

[35] C.-A. Sun, H. Dai, H. Liu, T. Y. Chen, and K.-Y. Cai, "Adaptive partition testing," *IEEE Transactions on Computers*, 2018.

[36] L. Zhang, B.-B. Yin, J. Lv, K.-Y. Cai, S. S. Yau, and J. Yu, "A history-based dynamic random software testing," in *Proceedings of the 38th International Computer Software and Applications Conference Workshops (COMPSACW'14)*, 2014, pp. 31–36.

[37] D. Gettys, "If you write documentation, then try a decision table," *IEEE Transactions on Professional Communication*, no. 4, pp. 61–64, 1986.

[38] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, 2011, pp. 1–10.

[39] M. Bozkurt, M. Harman, Y. Hassoun *et al.*, "Testing web services: a survey," *Department of Computer Science, Kings College London, Tech. Rep. TR-10-01*, 2010.

[40] D. Qiu, B. Li, S. Ji, and H. Leung, "Regression testing of web service: a systematic mapping study," *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–21, 2015.

[41] S. R. Dalal, A. Jain, N. Karunanithi, J. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, 1999, pp. 285–294.

[42] Z. J. Li, J. Zhu, L.-J. Zhang, and N. M. Mitsumori, "Towards a practical and effective method for web services test case generation," in *Proceedings of the 4th International Workshop on Automation of Software Test (AST'09), Co-located with the the 31st International Conference on Software Engineering (ICSE'09)*, 2009, pp. 106–114.

[43] A. Sinha and A. Paradkar, "Model-based functional conformance testing of web services operating on persistent data," in *Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications, Co-located with the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'06)*, 2006, pp. 17–22.

[44] A. M. Paradkar, A. Sinha, C. Williams, R. D. Johnson, S. Outterson, C. Shriver, and C. Liang, "Automated functional conformance test generation for semantic web services," in *Proceedings of the International Conference on Web Services (ICWS'07)*, 2007, pp. 110–117.

[45] D. Xiang, N. Xie, B. Ma, and K. Xu, "The executable invocation policy of web services composition with petri net," *Data Science Journal*, vol. 14, 2015.

[46] W. L. Dong and Y. Hang, "Web service testing method based on fault-coverage," in *Proceedings of the Intanational Workshop on Models for Enterprise Computing (IWMEC'06), Co-located with the 10th IEEE International Enterprise Distributed Object Conference (E-DOC'06)*, 2006, pp. 43–43.

[47] S. Hanna and M. Munro, "An approach for wsdl-based automated robustness testing of web services," in *Information Systems Development*, 2009, pp. 1093–1104.

[48] A. Bertolino, J. Gao, E. Marchetti, and A. Polini, "Automatic test data generation for xml schema-based partition testing," in *Proceedings of the 2nd International Workshop on Automation of Software Test (AST'07), Co-located with the 29th International Conference on Software Engineering (ICSE'07)*, 2007, p. 4.

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

[50] H. Zhu and Y. Zhang, "Collaborative testing of web services," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 116–130, 2012.

[51] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep., 1998.

[52] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: a review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, p. 4, 2018.

[53] T. Y. Chen, H. Leung, and I. Mak, "Adaptive random testing," in *Proceedings of the 9th Annual Asian Computing Science Conference (ASIAN'04)*. Springer, 2004, pp. 320–329.

[54] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing," in *Proceedings of the 7th European Conference on Software Quality (ECSQ'02)*. Springer, 2002, pp. 321–330.

[55] ——, "Restricted random testing: Adaptive random testing by exclusion," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 04, pp. 553–584, 2006.

[56] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, R. Merkel, and G. Rothermel, "A cost-effective random testing method for programs with non-numeric inputs," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3509–3523, 2016.

[57] H. Hu, W. E. Wong, C.-H. Jiang, and K.-Y. Cai, "A case study of the recursive least squares estimation approach to adaptive testing for software components," in *Proceedings of the 5th International Conference on Quality Software (QSIC'05)*, 2005, pp. 135–141.

[58] H. Hu, C.-H. Jiang, and K.-Y. Cai, "An improved approach to adaptive testing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 05, pp. 679–705, 2009.

Chang-ai Sun is a Professor in the School of Computer and Communication Engineering, University of Science and Technology Beijing. Before that, he was an Assistant Professor at Beijing Jiaotong University, China, a postdoctoral fellow at the Swinburne University of Technology, Australia, and a postdoctoral fellow at the University of Groningen, The Netherlands. He received the bachelor degree in Computer Science from the University of Science and Technology Beijing, China, and the PhD degree in Computer Science from Beihang University, China. His research interests include software testing, program analysis, and Service-Oriented Computing.

Hepeng Dai is a PhD student in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. He received the master degree in Software Engineering from University of Science and Technology Beijing, China and the bachelor degree in Information and Computing Sciences from China University of Mining and Technology, China. His current research interests include software testing and debugging.

Guan Wang is a masters student at the School of Computer and Communication Engineering, University of Science and Technology Beijing. He received a bachelor degree in Computer Science from University of Science and Technology Beijing. His current research interests include software testing and Service-Oriented Computing.

Dave Towey is an associate professor in the School of Computer Science, University of Nottingham Ningbo China. He received his BA and MA degrees from The University of Dublin, Trinity College, PgCertTESOL from The Open University of Hong Kong, MEd from The University of Bristol, and PhD from The University of Hong Kong. His current research interests include technology-enhanced teaching and learning, and software testing, especially metamorphic testing and adaptive random testing. He is a member of both the IEEE and the ACM.

Tsong Yueh Chen is a Professor of Software Engineering at the Department of Computer Science and Software Engineering in Swinburne University of Technology. He received his PhD in Computer Science from The University of Melbourne, the MSc and DIC from Imperial College of Science and Technology, and BSc and MPhil from The University of Hong Kong. His current research interests include software testing and debugging, software maintenance, and software design.

Kai-Yuan Cai received the BS, MS, and PhD degrees from Beihang university, Beijing, China, in 1984, 1987, and 1991, respectively. He has been a full professor at Beihang University since 1995. He is a Cheung Kong Scholar (chair professor), jointly appointed by the Ministry of Education of China and the Li Ka Shing Foundation of Hong Kong in 1999. His main research interests include software testing, software reliability, reliable flight control, and software cybernatics.