Process Feedback Driven Approach Towards Selection of Source Test Cases and Metamorphic Relations for Metamorphic Testing

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. In this paper, we propose a dynamic random testing (DRT) technique for web services, which 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

Test result verification is an important part of software testing. A test oracle [1] is a mechanism that can exactly decide whether the output produced by a programs is correct. However, there are situations where it is difficult to decide whether the result of the software under test (SUT) agrees with the expected result. This situation is known as oracle problem [2], [3]. In order to alleviate the oracle problem, several techniques have been proposed, such as N-version testing [4], metamorphic testing (MT) [5], [6], assertions [7], and machine learning [8]. Among of them, MT obtains metamorphic relations (MRs) according to the properties of SUT. Then, MRs are used to generate new test cases called follow-up test cases from original test cases known as source test cases. Next, both source and follow-up test cases are executed and their result are verified against the corresponding MRs.

MT has been receiving increasing attention in the software testing community, since this technique not only alleviates the oracle problem but also generates new test cases from existing test cases. MT has been successfully applied in a number of application domains and paradigms, including healthcare [9], bioinformatics [10], air traffic control [11], the web service [12], the RESTful web APIs [13], and testing of deep learning system [14]. The applications of MT emphasizes that MT is a new and promising strategy that complements the existing testing approaches.

The successes in applying MT to multiple application domains and paradigms continues to stimulate researchers to develop the theoretical foundations for MT. There are two broad categories of methods to improve the effectiveness of MT: source test cases generating and MRs identifying. The former employs different test case selection strategies to generate source test cases for MT [15], [16]. The latter creates MRs by combining existing relations, or by generating them automatically [17]–[21].

Random testing (RT) that randomly selects test cases from input domain (which refers to the set of all possible inputs of SUT), which is most commonly used strategies in traditional MT [22]. Although RT is simple to implement, RT does not make use of any execution information about SUT or the test history. Thus, traditional MT may be ineffectiveness in some situations. Barus et al. [16] employed adaptive random testing [23] that is a class of testing method aimed to improve the performance of RT by increasing the diversity across a programs input domain of the test cases executed to generate source test cases for MT. The Fixed-Size Candidate Set ART technique (FSCS-ART) [23] was the first ART method, and is also the most widely studied, which selects a next test input from the fixed-size candidate set of tests that is farthest from all previously executed tests. Many other ART algorithms have also been proposed, including RRT [24], [25], DF-FSCS [26], and ARTsum [27], with their effectiveness examined and validated through simulations

A preliminary version of this paper was presented at the 36th Annual IEEE Computer Software and Applications Conference (COMPSAC 2012) [?].

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.

and experiments.

In contrast to RT, partition testing (PT) attempts to generate test cases in a more systematic way, aiming to use fewer test cases to detect more faults. When conducting PT, the input domain of 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, that is, test cases in the same partition should have similar software execution behavior: If one input can detect a fault, then all other inputs in the same partition should also be able to detect a fault. RT and PT have their own characteristics. Therefore, it is a nature thought to investigate the integration of them for developing new testing techniques. Cai et al. [28] proposed dynamic random testing (DRT) that takes advantages of testing information to control the testing process, with the goal of benefitting from the advantages of RT and PT. Sun et al. proposed adaptive partition testing [29], where test cases are randomly selected from some partition whose probability of being selected is adaptively adjusted along the testing process, and developed two algorithms, Markov-chain based adaptive partition testing and reward-punishment based adaptive partition testing, to implement the proposed approach.

The set of identified MRs is the other foremost component of MT, which impacts the fault-detection effectiveness of MT, and that a common opinion has been identification of these MRs appears to be a manual and tough task [30]: "Even experts may find it difficult to discover metamorphic relations". To address this question, a number of methods have been proposed such as MR composition [31], [32], Machine-learning based MR detection [33], Graph-kernel based MR prediction [34], Search-based MR inference [35], data-mutation-directed MR acquisition [36]. More recently, a more systematical and effective method (METRIC) was proposed, which make use of complete test frame constructed by categories and choices [37]. Those methods help test engineers create a set of MRs, while cannot guide select the next MR during the test.

In previous and existing work, researchers investigated the effectiveness of MT by either choosing different MRs alone, or choosing different test case selection strategies to generate source test cases alone. Different from the previous methods, this paper proposed an innovative method (PFMT) that makes use of feedback information to select both source test case and corresponding MR, aiming to maximize the fault-detection effectiveness of MT.

PFMT is built on top of four insights: i) source test cases; ii) there are no work that relates to select MR; iii) select both source test case and MR, which may be more effectiveness than one; iv) analyzing the feedback information can guide select "good" source test case and MR.

In short, PFMT collects feedback information during the test, and makes use of those information to select source test case and .

In this study, we investigate how to make use of feedback information in the previous tests to control the execution process of MT in terms of both selecting source test cases and MRs. As a result, a process-feedback metamorphic testing framework is proposed to improve the fault-detecting efficiency of MT. An empirical study was conducted to evaluate the efficiency of the proposed technique. Main contributions made in this paper are the following:

1 From a new perspective, we proposed a process feedback metamorphic testing method. This includes a universal framework (PFMT) that indicates how to make use of history testing information to select next source test case and MR.
2 We particular developed two kinds of algorithms, partition based PFMT (P-PFMT) and random based PFMT (R-PFMT), to implement the proposed framework. P-PFMT includes two methods: i) An MRs-centric P-PFMT (MP-PFMT), which first randomly selects an MR to generate source and follow-up test cases of related input partitions, and then updates the test profile of input partitions according to the result of test execution. Second, a partition is selected according to updated test profile, and an MR is selected based on proposed strategies from the set of MRs whose source test cases belong to selected partition; ii) A partition-centric P-PFMT (PP-PFMT), which leverages MRs as a mechanism for verifying the test results. First, PP-PFMT selects a partition according to the test profile. Second, a test case is randomly selected in the selected partition, its follow-up test cases are generated based on the involved MRs, and their results are verified against the involved MRs. Third, PP-AMT updates the test profile based on the test results. R-PFMT include one method: we employed ARTsum to select source test cases and then an MR is selected based on proposed strategies from the set of MRs whose source test cases is selected source test case.
3 In order to support proposed methods, we proposed two algorithms to update the test profile, and two algorithms to select an MR form the candidate MRs.
4 We evaluated the performance of PFMT through a series of empirical studies on 12 programs. These studies show that PFMT has significantly higher fault-detection efficiency than traditional MT that randomly selects source test cases and a revised MT method that employed ART method to select source test cases, ignoring to select a "good" MR.

The rest of this paper is organized as follows. Section 2 introduces the underlying concepts for DRT, web services and mutation analysis. Section **??** presents the DRT framework for web services, guidelines for its parameter settings, and a prototype that partially automates DRT. Section **??** 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 **??** concludes the paper.

2 Background

In this section, we present some of the underlying concepts for MT, DRT, and ART.

2.1 Metamorphic Testing (MT)

MT is a novel technique to alleviate the oracle problem: Instead of applying an oracle, MT uses a set of MRs (corresponding to some specific properties of the SUT) to verify the test results. MT is normally conducted according to the following steps:

Step1. Identify an MR from the specification of the SUT.

Step2. Generate the source test case stc using the traditional test cases generation techniques.
Step3. Derive the follow-up test case ftc from the stc based on the MR.
Step3. execute stc and ftc and get their outputs O_s and O_f.
Step4. Verify stc, ftc, O_s, and O_f against the MR: If the MR does not hold, a fault is said to be detected.

The above steps can be repeated for a set of MRs.

Let us use a simple example to illustrate how MT works. For instance, consider the mathematic function $f(x, y)$ that can calculate the maximal value of two integers x and y. There is a simple yet obvious property: the order of two parameters x and y does not affect the output, which can be described as the follow metamorphic relation (MR): $f(x, y) = f(y, x)$. In this MR, (x, y) is source test case, and (y, x) is considered as follow-up test case. Suppose `P` denotes a program that implements the function $f(x, y)$, `P` is executed with a test cases $(1, 2)$ and $(2, 1)$. Then we check $P(1, 2) = P(2, 1)$: If the equality does not hold, then we consider that `P` at least has one fault.

2.2 METRIC

2.3 Dynamic Random Testing (DRT)

Cai et al [28] proposed dynamic random testing (DRT), which adjusts the test profile according to the result of current test. In DRT, the input domain is first divided into m partitions (denoted $s_1, s_2, \ldots, s_m$), and each s_i is assigned a probability p_i. During the test process, the selection probabilities of partitions are dynamically updated. Suppose that a test case tc from s_i ($i = 1, 2, \ldots, m$) is selected and executed. The process of DRT adjusting the value of p_i is as follows: If tc detects a fault, $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we then set

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

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 tc does not detect a fault, we set

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

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

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

The detailed DRT algorithm is given in Algorithm 1. In DRT, a test case is selected from a partition that has been randomly selected according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$ (Lines 2 to 4). If a fault is detected, then Formulas 1 and 2 are used to adjust the values of p_i (Line 6), otherwise Formulas 3 and 4 are used (Line 8). This process is repeated until a termination condition is satisfied (Line 1). Termination conditions could be set as "when the testing resources are consumed exhausted", "when a certain number of test cases have been executed", or "when the first fault is detected".

In AMT, when a source test case and corresponding follow-up test case belong to same partition s_i, DRT is employed to update the test profile as follows: If a fault is detected (i.e., their results violate the related MR), then Formulas 1 and 2 are used to adjust the values of p_i, otherwise Formulas 3 and 4 are used. During the testing process, there is the other situation where a source test case and corresponding follow-up test case do not belong to same partition. In order to adjust the test profile in this situation, we proposed a new strategy that is described in Section 3.3.

Algorithm 1 DRT

```
Input: $\epsilon, p_1, p_2, \ldots, p_m$
1: while termination condition is not satisfied do
2:     Select a partition $s_i$ according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$.
3:     Select a test case $tc$ from $s_i$.
4:     Test SUT using $tc$.
5:     if a fault is detected by $tc$ then
6:         Update test profile according to Formulas 1 and 2.
7:     else
8:         Update test profile according to Formulas 3 and 4.
9:     end_if
10: end_while
```

2.4 Adaptive Random Testing (ART)

To improve the fault-detection efficiency of RT, Chen et al. proposed adaptive random testing [23] based on the intuition that two test cases close to each other were more likely to have the same failure behavior than two test cases that were widely separated [38], that means faults tend to cause erroneous behaviour to occur across contiguous regions of the input domain. Hence, if a method that spread test cases more evenly would identify failures using fewer test cases.

There are many possible ways to apply the principle of ART. One that has been widely used is distance-based ART, also known as Fixed-Size Candidate Set ART (FSCS-ART) [23]. It makes use of the Euclidean distance as a distance measure to ensure test cases are spread out evenly. At first, two types of sets are defined: candidate set and executed set. The candidate set contains a predefined number n candidate inputs, whereas the later set stores all test cases that have been executed. Both sets are initially empty. Then n elements of candidate set are selected randomly from the existing input domain. For the first round, a random test case is selected, executed, and if the execution does not reveal any failures, it would be added to the executed set whilst the candidate set is reset as empty. For subsequent iterations, n test cases are randomly selected and placed again in the candidate set. Then, the minimum distance of each candidate test case to the executed test cases is calculated, that is the nearest executed test case. The candidate having maximum minimum distance is then selected as the next test case. This is known as the max-min criterion. The testing stops whenever an executed test case reveals a failure.

3 Process Feedback Driven Approach to Select Source Test Cases and MRs

In this section, we describe a framework for applying feedback mechanism to select source test cases and MRs for MT.

3.1 Motivation

Since MT was first published, a considerable number of studies have been reported from various aspects [22]. To improve the efficiency of MT, most of studies have paid their attention to identify the better MRs, which are more likely to be violated. For the efficiency of MRs, several factors such as the difference between the source and follow-up test cases [17], [18] and the the detecting-faults capacity of MRs compared to existing test oracles [39], have been investigated.

Since the follow-up test cases are generated based on source test cases and MRs, in addition to the so-called good MRs, source test cases also have an impact on the efficiency of MT. However, 57% of existing studies employed RT to select test cases, and 34% of existing studies used existing test suites according to a survey report by Segura et al. [22]. In this study, we investigate the strategies of selection test cases and MRs, and its impacts on the fault-detecting efficiency of MT.

It has been pointed out that fault-detecting inputs tend to cluster into "continuous regions" [40], [41], that is, the test cases in some partitions are more likely to detect faults than the test cases in other partitions. Inspired by the observation, AMT takes full advantage of feedback information to update the test profile, aiming at increasing the selection probabilities of partitions with larger failure rates. Accordingly, the MRs whose sources test cases belonging to the partitions with larger failure rates, are more likely to be selected and violated. Therefore, AMT is expected to detect faults more efficient than traditional MT.

3.2 Framework

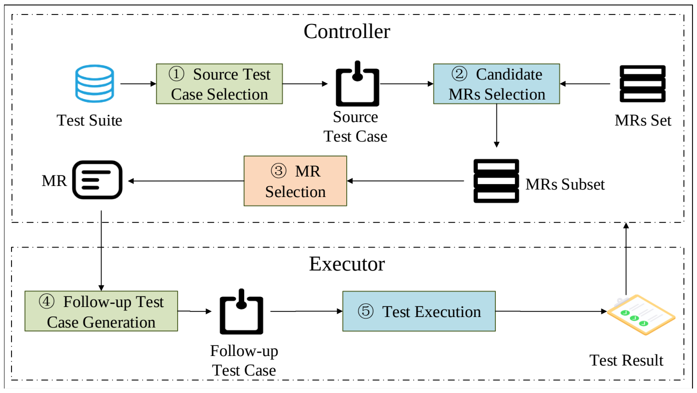

Fig. 1. framework

1 *Source Test Case Selection.*
2 *candidate MRs Selection.*
3 *MR Selection.*
4 *Follow-up Test Case Generation.*
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).

3.3 Updating Test Profile Strategies

When the source test case stc and follow-up test case ftc related to an metamorphic relation MR belong to the same partition s_i, DRT is suitable to adjust the test profile. However, during the test process, there are some scenarios where the stc is not in the same partition as ftc. In such scenarios, DRT cannot be used to adjust the test profile. To solve this problem, we proposed metamorphic dynamic random testing (MDRT).

Suppose that source test case stc and follow-up test case ftc related to an metamorphic relation MR, belong to s_i and s_f ($f \in \{1, 2, \ldots, m\}, f \neq i$), respectively. If their results violate the MR, $\forall j = 1, 2, \ldots, m$ and $j \neq i, f$, we set

$$p'_j = \begin{cases} p_j - \dfrac{2\epsilon}{m-2} & \text{if } p_j \geq \dfrac{2\epsilon}{m-2} \\ 0 & \text{if } p_j < \dfrac{2\epsilon}{m-2} \end{cases}, \tag{5}$$

where ϵ is a probability adjusting factor, and then

$$p'_i = p_i + \frac{(1 - \sum_{j=1, j\neq i,m}^{m} p'_j) - p_i - p_f}{2}, \tag{6}$$

$$p'_f = p_f + \frac{(1 - \sum_{j=1, j\neq i,m}^{m} p'_j) - p_i - p_f}{2}. \tag{7}$$

Alternatively, if their results hold the MR_h, we set

$$p'_i = \begin{cases} p_i - \epsilon & \text{if } p_i \geq \epsilon \\ 0 & \text{if } p_i < \epsilon \end{cases}, \tag{8}$$

$$p'_f = \begin{cases} p_f - \epsilon & \text{if } p_f \geq \epsilon \\ 0 & \text{if } p_f < \epsilon \end{cases}, \tag{9}$$

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

$$p'_j = p_j + \frac{(p_i - p'_i) + (p_f - p'_f)}{m-2} \tag{10}$$

The detailed MDRT algorithm is given in Algorithm 2. In MDRT, the source test case is selected from a partition that has been randomly selected according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$, and an metamorphic relation is select according to some strategies (Lines 2 to 4 in Algorithm 1). During the testing, if the source and follow-up test cases are not in same partition, then the test profile is updated by changing the p_i: If a fault is detected, then Formulas 5, 6, and 7 are used to adjust the values of p_i (Line 8), otherwise Formulas 8, 9, and 10 are used (Line 10). The testing process is stopped as long as the termination condition is satisfied.

Algorithm 2 MDRT

```
Input: $\epsilon, p_1, p_2, \ldots, p_m, MR_1, MR_2, \ldots, MR_n$
1:  while termination condition is not satisfied do
2:    Select a partition $s_i$ according to the test profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$.
3:    Select a source test case $stc_i$ from $s_i$, and an metamorphic relation $MR_h$ ($h \in \{1, 2, \ldots, n\}$).
4:    Based on the $MR_h$, follow-up test case $ftc_i$ is generated from $stc_i$, belonging to partition $s_f$.
5:    Test the SUT using $stc_i$ and $ftc_i$.
6:    if $i \neq f$ then
7:      if the results of $stc_i$ and $ftc_i$ violate the MR then
8:        Update $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$ according to Formulas 5, 6, and 7.
9:      else
10:       Update $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$ according to Formulas 8, 9, and 10.
11:     end_if
12:   end_if
13: end_while
```

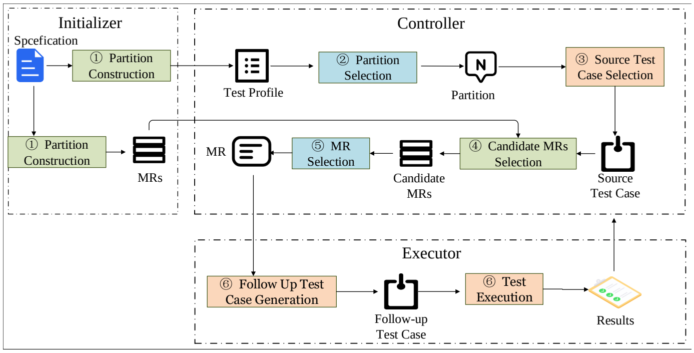

Fig. 2. framework

3.4 Selecting MR Strategies

3.5 Partition Based PFMT

3.5.1 A partition-centric P-PFMT

3.5.2 A MRs-centric P-PFMT

3.6 Random Based PFMT

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 efficient is SDMT at detecting faults?
Fault-detection efficiency is a key criterion for evaluating the performance of a testing technique. In our study, we chose three real-life programs, and applied mutation analysis to e-valuate the fault-detecting efficiency.

RQ2 What is the actual test case selection overhead when using the M-AMT strategy?

Algorithm 3 PP-PFMT

```
Input: $\epsilon, p_1, \ldots, p_m, MR_{s_1}, \ldots, MR_{s_m}, \mathcal{R}, counter$
1:  Initialize $counter = 1$.
2:  while termination condition is not satisfied do
3:    if $counter = 1$ then
4:      Randomly select an $MR$ from $\mathcal{R}$, and $MR \in MR_{s_i}$.
5:      Increment counter by 1.
6:      Randomly select a source test case from the partition $s_i$, and generate the follow-up test case belonged to partition $s_f$ ($f \in 1, 2, \ldots, m$).
7:      Execute the source and follow-up test cases.
8:      if the results of source and follow-up test cases violate $MR$ then
9:        if $i = f$ then
10:         Update the test profile according to Formulas 1 and 2.
11:       else
12:         Update the test profile according to Formulas 5, 6, and 7.
13:       end_if
14:     else
15:       if $i = f$ then
16:         Update the test profile according to Formulas 3 and 4.
17:       else
18:         Update the test profile according to Formulas 8, 9, and 10
19:       end_if
20:     end_if
21:   else
22:     Select a partition $s_i$ according to the updated test profile, and then select an $MR^{'}$ from $MR_{s_i}$.
23:     Randomly select a source test case from the partition $s_i$, and generate the follow-up test case belonged to partition $s_f$ ($f \in 1, 2, \ldots, m$).
24:     Execute the source and follow-up test cases.
25:     if the results of source and follow-up test cases violate $MR^{'}$ then
26:       if $i = f$ then
27:         Update the test profile according to Formulas 1 and 2.
28:       else
29:         Update the test profile according to Formulas 5, 6, and 7.
30:       end_if
31:     else
32:       if $i = f$ then
33:         Update the test profile according to Formulas 3 and 4.
34:       else
35:         Update the test profile according to Formulas 8, 9, and 10
36:       end_if
37:     end_if
38:   end_if
39: end_while
```

Algorithm 4 PM-PFMT

```
Input: $\epsilon, p_1, \ldots, p_m, MR_{s_1}, \ldots, MR_{s_m}, \mathcal{R}, counter$
1: Initialize $counter = 1$.
2: while termination condition is not satisfied do
3:   if $counter = 1$ then
4:     Randomly select an $MR$ from $\mathcal{R}$, and $MR \in MR_{s_i}$.
5:     Increment counter by 1.
6:     Randomly select a source test case from the partition $s_i$, and generate the follow-up test case belonged to partition $s_f$ ($f \in 1, 2, \ldots, m$).
7:     Execute the source and follow-up test cases.
8:     if the results of source and follow-up test cases violate $MR$ then
9:       if $i = f$ then
10:        Update the test profile according to Formulas 1 and 2.
11:      else
12:        Update the test profile according to Formulas 5, 6, and 7.
13:      end_if
14:    else
15:      if $i = f$ then
16:        Update the test profile according to Formulas 3 and 4.
17:      else
18:        Update the test profile according to Formulas 8, 9, and 10
19:      end_if
20:    end_if
21:  else
22:    Select a partition $s_i$ according to the updated test profile, and then select an $MR^{'}$ from $MR_{s_i}$.
23:    Randomly select a source test case from the partition $s_i$, and generate the follow-up test case belonged to partition $s_f$ ($f \in 1, 2, \ldots, m$).
24:    Execute the source and follow-up test cases.
25:    if the results of source and follow-up test cases violate $MR^{'}$ then
26:      if $i = f$ then
27:        Update the test profile according to Formulas 1 and 2.
28:      else
29:        Update the test profile according to Formulas 5, 6, and 7.
30:      end_if
31:    else
32:      if $i = f$ then
33:        Update the test profile according to Formulas 3 and 4.
34:      else
35:        Update the test profile according to Formulas 8, 9, and 10
36:      end_if
37:    end_if
38:  end_if
39: end_while
```

We evaluate the test case selection overhead of M-AMT and compare with traditional MT in detecting software faults.

4.2 Object Programs

In order to evaluate the fault-detection effectiveness of proposed methods in different scales, different implementation languages, and different fields, we chose to study four sets of object programs: three laboratorial programs that were developed according to corresponding specifications, seven programs from the Software-artifact Infrastructure Repository (SIR) [42], the regular expression processor component of the larger utility program `GUN grep`, and a Java library developed by Alibaba, which can be used to convert Java Objects into their JSON representation, and convert a JSON string to an equivalent Java object. The details of twelve programs are summarized in Table 1.

There four sets of programs present complementary strengths and weaknesses as experiment objects. The Laboratorial programs implement simple functions and their interfaces are easy to understand. The test engineers can easily generate source test cases, and identify MRs for testing laboratorial programs. However, these programs are small and there are a limited number of faulty versions available for each programs. The SIR repositories provides object programs, including a number of pre-existing versions with seeded faults, as well as a test suite in which test cases were randomly generated. It is a challenge task to partition the input domain of those programs. The `grep` program is a much larger system for which mutation faults could be generated. The `FastJson` is also a much larger system, and the faults of that are obtained on GitHub. We provide further details on each of those sets of object programs next.

4.2.1 Laboratorial Programs

Based on three real-lift specifications, We developed three systems, respectively. `China Unicom Billing System (CUBS)` provides an interface through which customers can know how much they need to pay according to plans, month charge, calls, and data usage. The details of two cell-phone plans are summarized in Tables 2 and 3. `Aviation Consignment Management System (ACMS)` aims to help airline companies check the allowance (weight) of free baggage, and the cost of additional baggage. Based on the destination, flights are categorized 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 4, where $price_0$ means economy class fare. `Expense Reimbursement System (ERS)` assists the sales Supervisor of a company with handling the following tasks: i) Calculating the cost of the employee who use the cars based on their titles and the number of miles actually traveled; ii) accepting the requests of reimbursement that include airfare, hotel accommodation, food and cell-phone expenses of the employee.

TABLE 1
Twelve Programs as Experimental Objects

programs	Source	Language	LOC	All Faults	Used Faults	Number of MRs	Number of Partitions
`CUBS`	Laboratory	Java	107	187	4	184	8
`ACMS`	Laboratory	Java	128	210	9	142	4
`ERS`	Laboratory	Java	117	180	4	1130	12
`grep`	GUN	c	10,068	20	20	12	–
`printtokens`	SIR	c	483	7	7	3	–
`printtokens2`	SIR	c	402	10	10	3	–
`replace`	SIR	c	516	31	31	3	–
`schedule`	SIR	c	299	9	9	3	–
`schedule2`	SIR	c	297	9	9	3	–
`tcas`	SIR	c	138	41	41	3	–
`totinfo`	SIR	c	346	23	23	3	–
`FastJson`	Alibaba	Java	204125	6	6	–	–

TABLE 4
ACMS 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 2
Plan A

Plan details		Month charge (CNY)			
		$option_1$	$option_2$	$option_3$	$option_4$
Basic	Free calls (min)	50	96	286	3000
	Free data (MB)	150	240	900	3000
Extra	Dialing calls (CNY/min)	0.25	0.15	0.15	0.15
	Data (CNY/KB)	0.0003			

TABLE 3
Plan B

Plan details		Month charge (CNY)			
		$option_1$	$option_2$	$option_3$	$option_4$
Basic	Free calls (min)	120	450	680	1180
	Free data (MB)	40	80	100	150
Extra	Dialing calls (CNY/min)	0.25	0.15	0.15	0.15
	Data (CNY/KB)	0.0003			

4.2.2 SIR Programs

The seven object programs selected from the SIR [24] repository were `printtokens`, `printtokens2`, `replace`, `schedule`, `schedule2`, `tcas`, and `totinfo`. These programs were originally compiled by Hutchins et al. [43] at Siemens Corporate Research. We used these programs for several reasons:

1 Faulty versions of the programs are available.
2 The programs are of manageable size and complexity for an initial study.
3 All programs and related materials are available from the SIR, and the MRs of each program are defined in [44].

The `printtokens` and `printtokens2` are independent implementations of the same specification. They each implement a lexical analyzer. Their input files are split into lexical tokens according to a tokenizing scheme, and their output is the separated tokens. The `replace` program is a command-line utility that takes a search string, a replacement string, and an input file. The search string is a regular expression. The replacement string is text, with some metacharacters to enable certain features. The `replace` program searches for occurrences of the search string in the input file, and produces an output file where each occurrence of the search string is replaced with the replacement string. The `schedule` and `schedule2` programs are also independent implementations of the same specification. They each implement a priority scheduler that takes three non-negative integers and an input file. The integers indicate the number of initial processes at the three available scheduling priorities. The `schedule` and `schedule2` programs then take as input a command file that specifies certain actions to be taken. The `tcas` program is a small part of a much larger aeronautical collision avoidance system. It performs simple analysis on data provided by other parts of the system and returns an integer indicating what action, if any, the pilot should take to avoid collision.

4.2.3 GUN grep

The used version of the `grep` programs is 2.5.1a [45]. This program searches one or more input files for lines containing a match to a specified pattern. By default, grep prints the matching lines. We chose `grep` for our study for several reasons:

1 The `grep` program is wide used in Unix system, providing a opportunity to demonstrate the real world relevance of our techniques.
2 The `grep` program, and its input format, are of greater complexity than the the programs in the other test sets, but still manageable as a target for automated test case generation.

The inputs of the `grep` were categorize into three components: options, which consist of a list of commands to modify the searching process, pattern, which is the regular expression to be searched for, and files, which refers to the input files to be searched.

The scope of functionality of this program is larger, which leads to construct test infrastructure to test all of functionality would have been impractical. Therefore, we restricted our focus to the regular expression analyzer of the `grep`.

4.2.4 *Real-World Popular Programs*

The `FastJson`

4.3 Variables

4.3.1 *Independent Variables*

The independent variable 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 [46] (the probability of at least one fault being detected by a test suite); the E-measure [**?**] (the expected number of faults detected by a test suite); the F-measure [29] (the expected number of test case executions required to detect the first fault); and the T-measure [**?**] (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 [**?**], [**?**], [28], [47]–[49], 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 **??**, 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 revealing the first fault. Therefore, we introduce the F2-measure [29] as 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 $T\text{-}time$, the time required to detect all faults. $F\text{-}time$ and $F2\text{-}time$ 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.

4.4 Experimental Settings

4.4.1 *Partitioning*

4.4.2 *Initial Test Profile*

4.4.3 *Constants*

4.5 Experimental Environment

4.6 Experiment Architecture

4.7 Threats To Validity

4.7.1 *Internal Validity*

4.7.2 *External Validity*

4.7.3 *Construct Validity*

4.7.4 *Conclusion Validity*

5 Experimental Results

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.

Acknowledgment

This research is supported by the National Natural Science Foundation of China (Grant Nos. 61872039 and 61872167), 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] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[2] 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.

[3] 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.

[4] S. S. Brilliant, J. C. Knight, and P. Ammann, "On the performance of software testing using multiple versions," in *Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS'90)*, 1990, pp. 408–415.

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

[6] 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.

[7] K. Y. Sim, C. S. Low, and F.-C. Kuo, "Eliminating human visual judgment from testing of financial charting software," *Journal of Software*, vol. 9, no. 2, pp. 298–312, 2014.

[8] W. K. Chan, S.-C. Cheung, J. C. Ho, and T. Tse, "Pat: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs," *Journal of Systems and Software*, vol. 82, no. 3, pp. 422–434, 2009.

[9] C. Murphy, M. S. Raunak, A. King, S. Chen, C. Imbriano, G. Kaiser, I. Lee, O. Sokolsky, L. Clarke, and L. Osterweil, "On effective testing of health care simulation software," in *Proceedings of the 3rd workshop on software engineering in health care*. ACM, 2011, pp. 40–47.

[10] T. Y. Chen, J. W. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC bioinformatics*, vol. 10, no. 1, p. 24, 2009.

[11] Z. Hui, S. Huang, Z. Ren, and Y. Yao, "Metamorphic testing integer overflow faults of mission critical program: A case study," *Mathematical Problems in Engineering*, vol. 2013, 2013.

[12] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 221–243, 2012.

[13] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of restful web apis," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1083–1099, 2018.

[14] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*. ACM, 2018, pp. 303–314.

[15] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang, "Metamorphic testing and testing with special values." in *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'04)*, 2004, pp. 128–134.

[16] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, and H. W. Schmidt, "The impact of source test case selection on the effectiveness of metamorphic testing," in *Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16)*. ACM, 2016, pp. 5–11.

[17] T. Y. Chen, D. Huang, T. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th lberoAmerican Symposium on Software Engineering and Knowledge Engineer-ing (JIISIC'04)*, 2004, pp. 569–583.

[18] Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in *Proceedings of the 13th International Conference on Quality Software (QSIC'13)*, 2013, pp. 153–162.

[19] 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.

[20] T. Chen, P. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 190–14, 2014.

[21] X. Xie, J. Li, C. Wang, and T. Y. Chen, "Looking for an mr? try metwiki today," in *Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 1–4.

[22] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

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

[24] 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.

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

[26] 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.

[27] 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.

[28] K. Cai, H. Hu, C.-h. 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.

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

[30] A. Groce, T. Kulesza, C. Zhang, S. Shamasunder, M. Burnett, W.-K. Wong, S. Stumpf, S. Das, A. Shinsel, F. Bice *et al.*, "You are the only possible oracle: Effective test selection for end users of interactive machine learning systems," *IEEE Transactions on Software Engineering*, vol. 40, no. 3, pp. 307–323, 2014.

[31] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *Proceedings of the 12th International Conference on Quality Software (QSIC'12)*, 2012, pp. 59–68.

[32] G.-W. Dong, B.-W. Xu, L. Chen, C.-H. Nie, and L.-L. Wang, "Case studies on testing with compositional metamorphic relations," *Journal of Southeast University (English Edition)*, vol. 24, no. 4, pp. 437–443, 2008.

[33] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013, pp. 1–10.

[34] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels," *Software testing, verification and reliability*, vol. 26, no. 3, pp. 245–269, 2016.

[35] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14)*, 2014, pp. 701–712.

[36] C.-A. Sun, Y. Liu, Z. Wang, and W. K. Chan, "μmt: A data mutation directed metamorphic relation acquisition methodology," in *Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 12–18.

[37] T. Y. Chen, P.-L. Poon, and T. Tse, "A choice relation framework for supporting category-partition test case generation," *IEEE transactions on software engineering*, vol. 29, no. 7, pp. 577–593, 2003.

[38] 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.

[39] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, 2014.

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

[41] G. B. Finelli, "Nasa software failure characterization experiments," *Reliability Engineering & System Safety*, vol. 32, no. 1-2, pp. 155–169, 1991.

[42] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[43] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow-and control-flow-based test adequacy criteria," in *Proceedings of 16th International conference on Software engineering*. IEEE, 1994, pp. 191–200.

[44] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Information and Software Technology*, vol. 55, no. 5, pp. 866–879, 2013.

[45] T. G. Project, "Grep home page. http://www.gnu.org/software/grep," 2006.

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

[47] 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.

[48] 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.

[49] Z. Yang, B. Yin, J. Lv, K. Cai, S. S. Yau, and J. Yu, "Dynamic random testing with parameter adjustment," in *Proceedings of the 6th IEEE International Workshop on Software Test Automationthe, Co-located with the 38th IEEE Annual International Computer Software and Applications Conference (COMPSAC14)*, 2014, pp. 37–42.

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. He is the inventor of metamorphic testing and adaptive random testing.

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.