AMT: An Efficient and Novel Method to Improve the Fault-detecting Efficiency of Metamorphic Testing

Chang-ai Sun, *Senior Member, IEEE,* Hepeng Dai, Huai Liu, *Member, IEEE,* and Tsong Yueh Chen, *Member, IEEE,*

Abstract—Metamorphic testing (MT) is a promising technique to alleviate the oracle problem, which first defines metamorphic relations (MRs) that are then used to generate new test cases (i.e. follow-up test cases) from the original test cases (i.e. source test cases), and verify the results of source and follow-up test cases. Many efforts have been reported to improve MT's efficiency by either generating better MRs that are more likely to be violated or selecting different test case selection strategies to generate source test cases. Unlike these efforts, we investigate how to improve the efficiency of MT in terms of test executions. Furthermore, traditional MT techniques often employ the random testing strategy (RT) to select source test cases for execution, which could be inefficient because the feedback information during the testing process is not leveraged. Consequently, we propose ~~an~~ adaptive metamorphic testing (AMT) technique to improve the efficiency of MT through controlling the execution process of MT. We conducted an empirical study to evaluate the efficiency of the proposed technique with three real-life programs. Empirical results show that AMT outperforms traditional MT in terms of fault-detecting efficiency.

Index Terms—metamorphic testing, control test process, feedback, random testing, adaptive random testing, partition testing, adaptive partition testing

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 drawing 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. The technique 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 ~~artificial intelligence system~~ [14]–[16]. ~~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 [17], [18]. The latter creates MRs by combining existing relations, or by generating them automatically [19]–[23].

The fault-detecting effectiveness of MT relies on the quality of MRs and the source test cases. There are astronomically large number of studies to investigate generate good MRs [19]–[23] or the source test cases [17], [24], [25]. However, Researchers ignore the impact of test execution on the efficiency of MT.

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 [26]. 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. [18] employed adaptive random testing (ART) [27] that is a class of testing method aimed to improve the performance of RT by increasing the diversity across a program's input domain of the test cases executed to generate source test cases for MT. In terms of the choice of metamorphosis relationship, it is difficult for practitioners to choose the MRs based on the existing work

C.-A. Sun, and H. Dai are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China. E-mail: casun@ustb.edu.cn.
H. Liu is with the College of Engineering and Science, Victoria University, Melbourne VIC 8001, Australia. E-mail: Huai.Liu@vu.edu.au.
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.

or need to consume a lot of resources to choose those [6], [17], [19], [28]–[31].

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 (AMT) that makes use of feedback information to select both source test case and corresponding MR based on software cybernetics [32], [33], aiming to maximize the fault-detection effectiveness of MT.

AMT is built on top of three insights: i) source test cases is one of factors that effects the fault-detecting efficiency of MT. However, random testing is commonly used to generate source test cases, and only a little researches related to use appropriate test cases selection strategies to generate source test cases [26]; ii) most of existing work help testers create MRs, aiming to improve the fault-detection efficiency of MT, while there seems to be no investigation to study how to choose an MR based on the generated set of MRs; iii) an appropriate test cases selection strategy can generate "better" test cases (i.e. those test cases have a higher probability to detect a fault). In MT, if source test cases cannot detect a fault, and follow-up test case generated by an MR detect a fault, we still can conclude that this execution detect a fault. Therefore, based on a selected source test case, choosing a "better" MR can improve the fault-detection efficiency.

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

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 adaptive metamorphic testing method. This includes a ~~universal~~ framework (AMT) 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 AMT (P-AMT) and random based AMT (R-AMT), to implement the proposed framework. P-AMT includes two methods: i) An MRs-centric P-AMT (MP-AMT), 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-AMT (PP-AMT), which leverages MRs as a mechanism for verifying the test results. First, PP-AMT 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-AMT 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 AMT through a series of empirical studies on 12 programs. These studies show that AMT 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 MT, APT and CPM. Section 3 presents the AMT framework, the strategies of test case selection, and the strategies of MRs selection. Section 4 describes an empirical study where the proposed AMT is used to test four laboratory programs, a Gun program, and an Alibaba program, 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, and APT.

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 result. 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 Adaptive Partition Testing (APT)

Based on software cybernetics, Sun [34] propose a new testing approach, adaptive partition testing, where test cases are randomly selected from some partition whose probability of being selected is adaptively adjusted along the testing process. Furthermore, they particularly develop two algorithms, Markov-chain based adaptive partition testing and reward-punishment based adaptive partition testing, to implement the proposed approach. A detailed description of each algorithm is given in the following.

2.2.1 Markov-Chain Based Adaptive Partition Testing (MAPT)

According to the concept of Markov chain, given two states i and j, the probability of transitioning from i to j is represented by $p_{i,j} = Pr\{j|i\}$. Suppose that a test case from c_i $(i = 1, 2, \dots, m)$ is selected and executed. In MAPT, we consider each partition as a state in the Markov matrix. If a partition c_i is selected for conducting a test, the probability of selecting c_j for conducting the next test will be $p_{i,j}$. MAPT will adaptively adjust the value of each $p_{i,j}$ according to the following procedure.

Suppose that a test case from c_i is selected and executed. If this test case reveals a fault, $\forall j = 1, 2, \dots, m$ and $j \neq i$, we set

$$p'_{i,j} = \begin{cases} p_{i,j} - \dfrac{\gamma \times p_{i,i}}{m-1} & \text{if } p_{i,j} > \dfrac{\gamma \times p_{i,i}}{m-1} \\ p_{i,j} & \text{if } p_{i,j} \leq \dfrac{\gamma \times p_{i,i}}{m-1} \end{cases}, \tag{1}$$

and then we set

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

Otherwise, that is, the test case does not reveal a fault, $\forall j = 1, 2, \dots, m$ and $j \neq i$, we set

$$p'_{i,j} = \begin{cases} p_{i,j} + \dfrac{\tau \times p_{i,j}}{m-1} & \text{if } p_{i,i} > \dfrac{\tau \times (1 - p_{i,i})}{m-1} \\ p_{i,j} & \text{if } p_{i,i} \leq \dfrac{\tau \times (1 - p_{i,i})}{m-1} \end{cases}, \tag{3}$$

and then we have

$$p'_{i,i} = \begin{cases} p_{i,i} - \dfrac{\tau \times (1 - p_{i,i})}{m-1} & \text{if } p_{i,i} > \dfrac{\tau \times (1 - p_{i,i})}{m-1} \\ p_{i,i} & \text{if } p_{i,i} \leq \dfrac{\tau \times (1 - p_{i,i})}{m-1} \end{cases}. \tag{4}$$

The details of MAPT is given in Algorithm 1. In MAPT, the first test case is selected from a partition that is randomly selected according to the initial probability profile $\{p_1, p_2, \dots, p_m\}$ (Lines 5 and 6 in Algorithm 1). After the execution of a test case, the Markov matrix P will be updated through changing the values of $p_{i,j}$ (Line 13): If a fault is revealed, Formulas 1 and 2 will be used; otherwise, Formulas 3 and 4 will be used. The updated matrix will be used to guide the random selection of the next test case (Lines 8 and 9). Such a process is repeated until the termination condition is satisfied (refer to Line 3). The termination condition here can be either "testing resource has been exhausted", or "a certain number of test cases have been executed", or "a certain number of faults have been detected", etc. Note that after a fault is detected (Line 11), the testing process continues only when the termination condition is not satisfied (Line 3); otherwise, the testing process is stopped.

Algorithm 1 MAPT

Input: $\gamma, \tau, p_1, p_2, \dots, p_m$
1: Initialize Markov matrix P by setting $p_{i,j} = p_j$
2: Set $noTC = 0$
3: **while** termination condition is not satisfied
4: **if** noTC = 0
5: Select a partition c_i according to profile $\{p_1, p_2, \dots, p_m\}$
6: Select a test case t from c_i
7: **else**
8: Given that the previous test case is from c_i, select a partition c_j according to profile $\{p_{i,1}, p_{i,2}, \dots, p_{i,m}\}$
9: Select a test case t from c_j
10: **end_if**
11: Test the software using t
12: Increment noTC by 1
13: Update P based on the testing result according to Formulas 1 to 4
14: **end_while**

2.2.2 reward-punishment based adaptive partition testing (RAPT)

Based on the reward and punishment mechanism, RAPT attempts to be quicker at selecting the fault-revealing test cases. Two parameters Rew_i and Pun_i are used in RAPT to determine to what extent a partition c_i can be rewarded and punished, respectively. If a test case in c_i reveals a fault, Rew_i will be incremented by 1 and Pun_i will become 0, and test cases will be repeatedly selected from c_i until a non-fault-revealing test case is selected from c_i. If a test case selected from c_i does not reveal a fault, Rew_i will become 0 and Pun_i will be incremented by 1. If Pun_i reaches a preset bound value Bou_i, c_i will be regarded to have a very low failure rate, and its corresponding p_i will become 0. Basically, the higher value Rew_i has, the larger p_i the partition c_i has. The p_i's adjustment mechanism of RAPT is as follows. Suppose that a test case from c_i is selected and executed. If this test case reveals a fault, $\forall j = 1, 2, \dots, m$ and $j \neq i$, we set

$$p'_j = \begin{cases} p_j - \dfrac{(1 + \ln Rew_i) \times \epsilon}{m-1} & \text{if } p_j \geq \dfrac{(1 + \ln Rew_i) \times \epsilon}{m-1} \\ 0 & \text{if } p_j < \dfrac{(1 + \ln Rew_i) \times \epsilon}{m-1} \end{cases}, \tag{5}$$

and then we have

$$p'_i = 1 - \sum_{\substack{j=1 \\ j \neq i}}^{m} p'_j. \tag{6}$$

Otherwise, that is, the test case does not reveal a fault, we set

$$p_i' = \begin{cases} p_i - \delta & \text{if } p_i \geq \delta \\ 0 & \text{if } p_i < \delta \text{ or } Pun_i = Bou_i \end{cases}, \tag{7}$$

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

$$p_j' = \begin{cases} p_j + \dfrac{\delta}{m-1} & \text{if } p_i \geq \delta \\ p_j + \dfrac{p_i'}{m-1} & \text{if } p_i < \delta \text{ or } Pun_i = Bou_i \end{cases}. \tag{8}$$

The details of RAPT is given in Algorithm 2. Like MAPT, RAPT selects the first test case from a partition that is randomly selected according to the initial profile $\{p_1, p_2, \ldots, p_m\}$ (Lines 5 and 6 in Algorithm 2). If a test case reveals a fault, the same partition will be used for selecting the next test case until a non-fault-revealing test case is selected (refer to the while loop from Lines 7 to 12). Otherwise, the probability profile for p_i will be updated according to Formulas 5 to 8 (Lines 14 to 19). Note that the values of Rew_i and Pun_i are adaptively adjusted during the testing process. After a fault is detected (Line 11), the testing process continues only when the termination condition is not satisfied (Line 3); otherwise, the testing process is stopped.

Algorithm 2 RAPT

Input: $\epsilon, \delta, p_1, p_2, \ldots, p_m, Bou_1, Bou_2, \ldots, Bou_m$

1: Initialize $Rew_i = 0$ and $Pun_i = 0$ for all $i = 1, 2, \ldots, m$
2: Set $noTC = 0$
3: **while** termination condition is not satisfied
4: Select a partition c_i according to the profile $\{p_1, p_2, \ldots, p_m\}$
5: Select a test case t from c_i
6: Test the software using t
7: **while** a fault is revealed by t
8: Increment Rew_i by 1
9: Set $Pun_i = 0$
10: Select a test case t from c_i
11: Test the software using t
12: **end_while**
13: Increment Pun_i by 1
14: **if** $Rew_i \neq 0$
15: Update p_j ($j = 1, 2, \ldots, m$ and $j \neq i$) and p_i according to Formulas 5 and 6, respectively
16: Set $Rew_i = 0$
17: **else**
18: Update p_i and p_j ($j = 1, 2, \ldots, m$ and $j \neq i$) according to Formulas 7 and 8, respectively
19: **end_if**
20: **end_while**

2.3 Category Partition Method (CPM)

The proposed AMT selects both source test cases and corresponding MRs by making use of information which is collected during the test process that means, we need to establish a connection between the source test cases and MRs. The connection provides a guideline for us to obtain the set of candidate MRs according to a source test case or select a source test case for an MR.

Category partition method (CPM) is specification-based testing technique ~~developed by Ostrand and Balcer~~ [35]. It helps software testers create test cases by refining the functional specification of a program into test specifications. The method consists of the following steps.

Step1. Decompose the functional specification into functional units that can be tested independently, then Identify the parameters (the explicit inputs to a functional unit) and environment conditions (the state of the system at the time of execution) that are known as *categories*.

Step2. Partition each *category* into *choices*, which include all the different kinds of values that are possible for that *category*.

Step3. Determine the constraints among the *choices* of different *categories*, and Write the test specification (which is a list of categories, choices, and constraints in a predefined format) using the test specification language TSL.

Step4. Use a generator to produce *test frames* from the test specification. Each generated *test frame* is a set of choices. Then, create a test case by selecting a single element from each *choice* in each generated *test frame*.

In this study, we made use of CPM to create test cases. During the process of testing, the choice information can be used as the basis for choosing the MRs.

3 Adaptive Metamorphic Testing

In this section, we describe a framework for process-feedback MT, present the motavation ~~of this paper~~ and argorithms about selection source test case and MRs.

3.1 Motivation

Since MT was first published, a considerable number of studies have been reported from various aspects [26]. 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 [19], [20] and the the detecting-faults capacity of MRs compared to existing test oracles [36], 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. [26]. 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" [37], [38], 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

Considering the principles of software cybernetics and the features of MT, we propose an adaptive metamorphic texting framework, as illustrated in Figure 1. Interactions between the controller, MT, and testers are depicted in the framework. We next discuss the individual framework components.

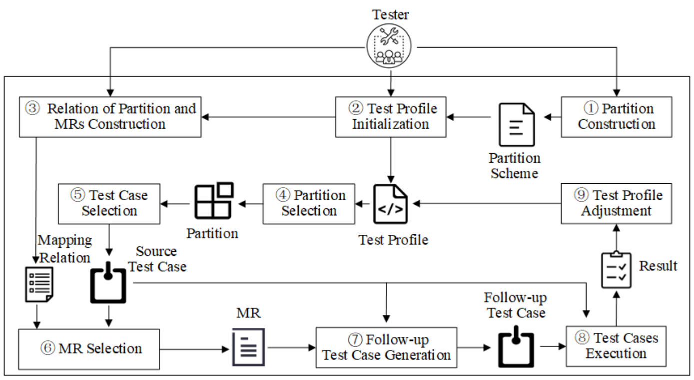


Fig. 1. The Framework of AMT

1 *Partition Construction*. Partition testing (PT) refers to a class of testing techniques that classify the input domain into a number of partitions [39]. Because APT 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 [39], [40]. The input domain of SUT can be partitioned based on the specification. 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.
2 *Test Profile Initialization*. Testers need to initialize the test profile, a simple way of doing which would be the use of a uniform probability distribution $p_1 = p_2 = \ldots = p_k$, where k denotes the number of partitions, and $p_i(i = 1, 2, \ldots, k)$ denotes the probability of selecting the i^{th} partition.
3 *Relation of Partition and MRs Construction*. The tester needs to create a subset $R_i(i = 1, 2, \ldots, k)$ of MRs for each partition s_i so that any test case belonging to partition p_i can generate follow-up test case driven by any MR belonging to R_i.
4 *Partition Selection*. AMT randomly selects a partition s_i according to the test profile.
5 *Test Case Selection*. AMT selects a test case from the selected partition s_i according to a uniform distribution.
6 *MR Selection*. The relevant AMT component firstly finds the subset R_i of MRs, and then selects an MR according to certain strategies (The detailed MR selection strategies are shown in section 3.4).
7 *Follow-up Test Case Generation*. The follow-up test case is derived from the selected source test case based on the selected MR.
8 *Test Cases Execution*. The relevant AMT component receives the generated source test case and follow-up test case, and executes them on SUT.
9 *Test Profile Adjustment*. Upon completion of each test, its pass or fail status is determined by comparing the actual and expected results (a pass status if both are the same). The pass or fail status is then used to adjust the (partition) probability distribution accordingly.

3.3 Source Test Case Selection

3.3.1 MAPT*

Suppose that source test case tc and corresponding follow-up test case tc' are belonging to partition s_s and s_f, respectively. if their results vioate the related MR, $\forall i = 1, 2, \ldots, m$ and $s, f \neq i$, we update test profile using the following equations. When source test case and follow-up test case belong to same partition ($s = f$), we set

$$p'_{s,i} = \begin{cases} p_{s,i} - \dfrac{\gamma \times p_{s,s}}{m-1} & \text{if } p_{s,i} > \dfrac{\gamma \times p_{s,s}}{m-1} \\ p_{s,i} & \text{if } p_{s,i} \leq \dfrac{\gamma \times p_{s,s}}{m-1} \end{cases} \quad (9)$$

and then,

$$p'_{s,s} = 1 - \sum_{i=0, i \neq s}^{m} p'_{s,i}. \quad (10)$$

Alternatively, if $s \neq f$, we set

$$p'_{s,i} = \begin{cases} p_{s,i} - \dfrac{\gamma \times p_{s,s}}{m-1} & \text{if } p_{s,i} > \dfrac{\gamma \times p_{s,s}}{m-1} \\ p_{s,i} & \text{if } p_{s,i} \leq \dfrac{\gamma \times p_{s,s}}{m-1} \end{cases} \quad (11)$$

$$p'_{f,i} = \begin{cases} p_{f,i} - \dfrac{\gamma \times p_{f,f}}{m-1} & \text{if } p_{f,i} > \dfrac{\gamma \times p_{f,f}}{m-1} \\ p_{f,i} & \text{if } p_{f,i} \leq \dfrac{\gamma \times p_{f,f}}{m-1} \end{cases} \quad (12)$$

and then,

$$p'_{s,s} = 1 - \sum_{i=0, i \neq f}^{m} p_{s,i} \quad (13)$$

$$p'_{f,f} = 1 - \sum_{i=0, i \neq f}^{m} p_{f,i} \quad (14)$$

When source test case and corresponding follow-up test case donot detect a fault, we employ the following equations to update test profile. If source test case and correspnding follow-up test case belong to same partition ($s = f$), we set

$$p'_{s,i} = \begin{cases} p_{s,i} + \dfrac{\tau \times p_{s,i}}{m-1} & \text{if } p_{s,s} > \dfrac{\tau \times (1 - p_{s,s})}{m-1} \\ p_{s,i} & \text{if } p_{s,s} \leq \dfrac{\tau \times (1 - p_{s,s})}{m-1} \end{cases} \quad (15)$$

and then,

$$p'_{s,s} = \begin{cases} p_{s,s} - \dfrac{\tau \times (1 - p_{s,s})}{m-1} & \text{if } p_{s,s} > \dfrac{\tau \times (1 - p_{s,s})}{m-1} \\ p_{s,s} & \text{if } p_{s,s} \leq \dfrac{\tau \times (1 - p_{s,s})}{m-1} \end{cases} \tag{16}$$

Alternatively, if $s \neq f$, we set

$$p'_{s,i} = \begin{cases} p_{s,i} + \dfrac{\tau \times p_{s,s}}{m-1} & \text{if } p_{s,s} > \dfrac{\tau \times (1 - p_{s,s})}{m-1} \\ p_{s,i} & \text{if } p_{s,s} \leq \dfrac{\tau \times (1 - p_{s,s})}{m-1} \end{cases} \tag{17}$$

$$p'_{s,s} = \begin{cases} p_{s,s} - \dfrac{\tau \times (1 - p_{s,s})}{m-1} & \text{if } p_{s,s} > \dfrac{\tau \times (1 - p_{s,s})}{m-1} \\ p_{s,s} & \text{if } p_{s,s} \leq \dfrac{\tau \times (1 - p_{s,s})}{m-1} \end{cases} \tag{18}$$

$$p'_{f,i} = \begin{cases} p_{f,i} + \dfrac{\tau \times p_{f,f}}{m-1} & \text{if } p_{f,f} > \dfrac{\tau \times (1 - p_{f,f})}{m-1} \\ p_{f,i} & \text{if } p_{f,f} \leq \dfrac{\tau \times (1 - p_{f,f})}{m-1} \end{cases} \tag{19}$$

$$p'_{f,f} = \begin{cases} p_{f,f} - \dfrac{\tau \times (1 - p_{f,f})}{m-1} & \text{if } p_{f,f} > \dfrac{\tau \times (1 - p_{f,f})}{m-1} \\ p_{f,f} & \text{if } p_{f,f} \leq \dfrac{\tau \times (1 - p_{f,f})}{m-1} \end{cases} \tag{20}$$

3.3.2 RAPT*

When source test case and follow-up test case belong to same partition ($s = f$), we set,

$$p'_i = \begin{cases} p_i - \dfrac{(1 + lnRe_{w_i}) \times \epsilon}{m-1} & \text{if } p_i > \dfrac{(1 + lnRe_{w_i}) \times \epsilon}{m-1} \\ 0 & \text{if } p_i \leq \dfrac{(1 + lnRe_{w_i}) \times \epsilon}{m-1} \end{cases} \tag{21}$$

and then we have

$$p'_s = 1 - \sum_{i=0, i \neq s}^{m} p'_i \tag{22}$$

Alternativelt, if $s \neq f$, we set

$$p'_i = \begin{cases} p_i - \dfrac{(1 + lnRe_{w_i}) \times \epsilon}{m-2} & \text{if } p_i > \dfrac{(1 + lnRe_{w_i}) \times \epsilon}{m-2} \\ 0 & \text{if } p_i \leq \dfrac{(1 + lnRe_{w_i}) \times \epsilon}{m-2} \end{cases} \tag{23}$$

and then we have

$$p'_s = p_s + \frac{1 - \sum\limits_{i=0, i \neq s, i \neq f}^{m} p'_i - p_s - p_f}{2} \tag{24}$$

$$p'_f = p_f + \frac{1 - \sum\limits_{i=0, i \neq s, i \neq f}^{m} p'_i - p_s - p_f}{2} \tag{25}$$

When source test case and corresponding follow-up test case donot detect a fault, we employ the following equations to update test profile. If source test case and correspnding follow-up test case belong to same partition ($s = f$), we set

$$p'_s = \begin{cases} p_s - \delta & \text{if } p_s > \delta \\ 0 & \text{if } p_s \leq \delta \text{ or } Pun_i = Bou_i, \end{cases} \tag{26}$$

and then

$$p'_i = \begin{cases} p_i + \dfrac{\delta}{m-1} & \text{if } p_s > \delta \\ p_i + \dfrac{p_s}{m-1} & \text{if } p_s \leq \delta \text{ or } Pun_i = Bou_i, \end{cases} \tag{27}$$

Alternatively, if $s \neq f$, we have

$$p'_s = \begin{cases} p_s - \delta & \text{if } p_s > \delta \\ 0 & \text{if } p_s \leq \delta \text{ or } Pun_i = Bou_i, \end{cases} \tag{28}$$

$$p'_f = \begin{cases} p_f - \delta & \text{if } p_f > \delta \\ 0 & \text{if } p_f \leq \delta \text{ or } Pun_i = Bou_i, \end{cases} \tag{29}$$

and then

$$p'_i = p_i + \frac{(p_s - p'_s) + (p_f - p'_f)}{m-2} \tag{30}$$

In AMT, once a source test case is selected, the corresponding candidate MRs are available. We first proposed a simple strategy based on random mechanism to select an MR from the set of candidate MRs, then we proposed an innovative strategy to select an MR that can make the execution of follow-up test case as different as possible from the source test case.

3.4 ~~MR Selection~~

3.4.1 Randomly MR-Selection Strategy (RMRS)

RSMR randomly selects an from the set of candidate MRs, which is a straightforward method without considering extra information.

3.4.2 Properties-Based strategy of MR selection (PBMR)

The fault-detection efficiency of MT highly dependent on the specific MRs that are used, and selecting effective MRs is thus a critical step when applying MT. Chen et al. reported that good metamorphic relations are those that can make the execution of the source-test case as different as possible to its follow-up test case. Moreover, They defined the "difference among execution" as any aspects of program runs (e.g., paths traversed). Before presenting the properties-based strategy of MR selection (PBMR), we first introduce a new metric called category-partition-based metric (CP-distance) that measures the distance between the source test cases and corresponding follow-up test cases. CP-distance makes use of the concepts of categories and choices from the CPM method described in Section 2.3. CP-distance have the capacity to reflect the difference among the source test cases and follow-up test cases, since the categories and choices are defined based on the software functionalities.

More formally, let us denote the set of categories by $A = \{A_1, A_2, \ldots, A_g\}$, where g denotes the total number of categories. For each A_i, its choices are denotes by $P_i^{A_i} = \{p_1^{A_i}, p_2^{A_i}, \ldots, p_h^{A_i}\}$, where h denotes the number of choices for A_i. For input x, let us denote the corresponding non-empty subset by $A(x) = \{A_1^x, A_2^x, \ldots, A_q^x\}$, where q refers to the number of categories associated with x. Since categories are distinct and their choices are disjoint, input x in fact consists of values chosen from a non-empty subset of choices, denoted as $P(x) = \{p_1^x, p_2^x, \ldots, p_q^x\}$, where $p_i^x (i = 1, 2, \ldots, q)$ is the choice of the category A_i^x for x.

For any two inputs x and y, we define $DP(x, y)$ as the set that contains elements in either $P(x)$ or $P(y)$ but not both. That is,

$$DP(x, y) = (P(x) \cup P(y))(P(x) \cap P(y)),$$

where "n" is the set difference operator. Now, we define

$$DA(x, y) = \{A_m | A_i if p_j^i \in DP(x, y)\}.$$

In other words, $DA(x, y)$ is the set of categories in which inputs x and y have different choices. Then, the distance measure between x and y is defined as $|DA(x, y)|$ (the size of $DA(x, y)$); that is, the number of categories that appear in either x or y but not both, or in which the choices in x and y differ.

obviously, a greater value of CP-distance representing more dissimilar execution. After selecting a source test case stc_i and obtaining a set of candidate MRs $\mathcal{R} = \{r_1^{stc_i}, r_2^{stc_i}, \ldots, r_g^{stc_i}\}$ (g is the number of MRs whose source test case could be stc_i), PBMR generate a set of candidate folow-up test cases $FC = \{ftc_{r_1^{stc_i}}, ftc_{r_2^{stc_i}}, \ldots, ftc_{r_g^{stc_i}}\}$ according to every MR belonged to FC. Then, the distance $CP_{i,h}$ ($h \in \{1, 2, \ldots, g\}$) between stc_i and each follow-up test case $ftc_{r_h^{stc_i}}$ is calculated. Finally, the MR $r_h^{stc_i}$ is selected as long as the following condition is hold:

$$CP_{i,h} = max\{CP_{i,1}, CP_{i,2}, \ldots, CP_{i,g}\}.$$

The details of PBMR is given in Algorithm 3.

Algorithm 3 PBMR

Input: $stc_i, \mathcal{R} = \{r_1^{stc_i}, r_2^{stc_i}, \ldots, r_g^{stc_i}\}, CParray = null$
Output: CP_{max} ($max \in \{1, 2, \ldots, g\}$)

```
for h = 1 → h = g do
    Generate the follow-up test case ftc_h according to the stc_i and r_h^{stc_i}
    Calculate the distance CP_{i,h} between the stc_i and ftc_h
    Add CP_{i,h} to the list CParray
end for
Calculate the maximal value CP_max (max ∈ {1, 2, ..., g}) in the list CParray
```

4 Empirical Study

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

4.1 Research Questions

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

RQ1 How efficient is AMT 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 evaluate the fault-detecting efficiency.

RQ2 What is the actual test case selection overhead when using the AMT technique?

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 three sets of object programs: four laboratorial programs that were developed according to corresponding specifications[1], the regular expression processor component of the larger utility program `GNU grep`, and a Java library developed by Alibaba[2], which can be used to convert Java Objects into their JSON representation, and convert a JSON string to an equivalent Java object. Table 1 ~~detailedly~~ summarized the basic information of the object programs, giving the developer of object programs (*Developer*), the name of object program (*Program*), the size of the object program (*LOC*), the number of total faults for each object program (*Number of All Faults*), the number of used faults in each object program (*Number of Used Faults*), test cases in the associated test pool (*Size of test suite*), the number of identified MRs for each object program (*Number of MRs*), and the number of partition for each object program (*Number of Partitions*).

There three 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 `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 ~~four~~ real-lift specifications~~. We developed four systems: `China Unicom Billing System (CUBS)`, `Aviation Consignment Management System (ACMS)`, `Expense Reimbursement System (ERS)`, `Meal Ordering System (MOS)`. These systems~~ were implemented using Java programming language.

`CUBS` provides an interface through which customers can know how much they need to pay according to plans, month charge, calls, and data usage.

`ACMS` aims to help airline companies check the allowance (weight) of free baggage, and the cost of additional baggage.

1. The implementations have been made available at: https://github.com/phantomDai/subjects4tsc.git
2. This project is available at https://github.com/alibaba/fastjson

TABLE 1
Twelve Programs as Experimental Objects

Developer	Program	Language	LOC	All Faults	Used Faults	Number of Test Cases	Number of MRs	Number of Partitions
Laboratory	`CUBS`	Java	107	187	3	—	184	8
	`ACMS`	Java	128	210	3	—	142	4
	`ERS`	Java	117	180	1	—	1130	12
	`MOS`	Java	135	215	1	—	3512	9
GNU	`grep`	C	10,068	20	8	101,193	12	550
Alibaba	`FastJson_v31`	Java	125,192	1	1	16,383	17	128
	`FastJson_v36`	Java	134,440	1	1	16,383	17	128
	`FastJson_v40`	Java	144,044	1	1	16,383	17	128
	`FastJson_v48`	Java	149,544	1	1	16,383	17	128

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.

`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.

`MOS` helps catering provider to determine the quantity for every type of meal and other special requests (if any) that need to be prepared and loaded onto the aircraft served by this provider. For each flight, `MOS` generates a Meal Schedule Report (MSR), containing the number of various types of meals (first class, business class, economy class, vegetarian, child, crew member, and pilot) and the number of bundles of flowers.

We employed the tool MuJava [41] to conduct mutation analysis [42], generating a total of 792 mutants. Each mutant was obtained by changing the original program's syntax (using one of all applicable mutation operators provided by MuJava). Equivalent mutants, and those that were too easily detected (requiring less than 20 randomly generated test cases), were removed. To ensure the statistical reliability, we obtained 30 different test suites using different random seeds, then tested all mutants with all test suites, calculating the average number of test cases needed to kill (detect) a mutant.

4.2.2 GNU grep

The used version of the `grep` programs is 2.5.1a [43]. 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 `grep` program is wide used in Unix system, providing a opportunity to demonstrate the real world relevance of our techniques.
2 `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.
3 `grep` program has been studied by several papers [18], [44], [45], that means, we can obtain its faults, source test cases and MRs freely and conveniently.

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`.

4.2.3 Real-World Popular Programs

`FastJson` is a Java library that can be used to convert Java Objects into their JSON representation (known as serialization). It can also be used to convert a JSON string to an equivalent Java object (known as deserialization). We chose `FastJson` for our study for several reason:

1 `FastJson` is wide used in real-word projects, providing a opportunity to demonstrate the real world relevance of our techniques.
2 `FastJson` is more complex than other programs, and it is a open source project, that is, we can obtain its faults freely and conveniently.

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 deserialization of the `FastJson`.

Two bigger version of `FastJson` are available, named `FastJson 1.1.*` (1.1.0 − 1.1.51) and `FastJson 1.2.*` (1.2.1 − 1.2.61), respectively. Since there are no failure reports for `FastJson 1.1.*`, we collected deserialization-related faults from the `FastJson 1.2.1-FastJson 1.2.48` (When we started this project, the latest version of FastJson was 1.2.48). We removed the faults that caused `FastJson` to throw an error information during execution.

4.3 Variables

4.3.1 Independent Variables

The independent variable in our experiment is the different strategies that improve the fault-detection efficiency. As choices for this variable, we include, of course, proposed

TABLE 2
The Details of Independent Variables

Techniques	Test case selection strategy	MRs selection strategy
MR-AMT	MAPT*	RT
MP-AMT	MAPT*	PBMR
RR-AMT	RAPT*	RT
RP-AMT	RAPT*	PBMR
MT	RT*	RT

adaptive metamorphic testing technique, which includes two test cases selection strategies (MAPT* and RAPT*) and two MRs selection strategies (RBMR and PBMR). As baseline techniques for use in comparison, we selected traditional MT, which randomly selects source test case and MRs.

Table 2 summarizes the details information of selecting source test cases and MRs of different techniques. Traditional MT is a natural baseline choice, because all proposed techniques is designed as an enhancement to MT, and assessing whether proposed techniques including MR-AMT (which uses MAPT* and RT strategies to select source test case and MR, respectively), MP-AMT (which uses MAPT* and PBMR strategies to select source test case and MR, respectively), RR-AMT (which uses RAPT* and RT strategies to select source test case and MR, respectively), and RP-AMT (which uses RAPT* and PBMR strategies to select source test case and MR, respectively) is more cost-effective than MT is important.

4.3.2 Dependent Variables

The choice of a metric to use in comparing the effectiveness of testing techniques is non-trivial.

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 [47] (the expected number of faults detected by a test suite); the F-measure [34] (the expected number of test case executions required to detect the first fault); and the T-measure [48] (the expected number of test cases required to detect all faults). Since the F-measures has been widely used for evaluating the fault-detection efficiency and effectiveness of MT, and APT testing techniques [18], [34], [45], they are also adopted in this study. In addition, as shown in Fig 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 revealing the first fault. Therefore, we introduce the F2-measure [34], [49] as the number of additional test cases required to reveal the second fault after detection of the first fault. We use F and $F2$ to represent the F-measure and the F2-measure of a testing technique.

We define $F\text{-}count$ as the number of test cases needed to detect a failure in a specific test run, $F2\text{-}count$ as the number of additional test cases needed to detect the second failure after detecting the first failure. The F-measure is the expected $F\text{-}count$ for a testing method:

$$F = \overline{F - count}. \tag{31}$$

The F2-measure is the expected $F2 - count$ for a testing method:

$$F = \overline{F2 - count}. \tag{32}$$

The F-measure is particularly appropriate for measuring the failure-detection efficience of MT. The value of F-measure indicates the speed of detecting failures for used testing techniques. The F2-measure can be further generalized to measure the number of test cases required for detecting the $i+1$th fault in the context of having the i th fault being detected in an iterative way. The value of F2-measure reflects the AMT's ability to detect failures with test history information.

Holm-Bonferroni method [50] to determine which pair of testing techniques have significant difference in terms of F. Across the whole study, for each pair of testing techniques, denoted by technique a and technique b, the null hypothesis (H_0) was that a and b had similar performance in terms of one metric; whereas the alternative hypothesis (H_1) was that a and b had significantly different performance in terms of F. All the null hypotheses were ordered by their corresponding p-values, from lowest to largest; in other words, for null hypotheses $H_0^1, H_0^2, \ldots, H_0^{h-1}$, we had $p_1 \leq p_2 \leq$ ... For the given confidence level $\alpha = 0.05$, we found the minimal index h such that $p_h > \frac{\alpha}{N+1-h}$ (where N is the total number of null hypotheses). Then, we rejected $H_0^1, H_0^2, \ldots, H_0^{h-1}$, that is, we regarded the pair of techniques involved in each of these hypotheses to have statistically significant difference in terms of a certain metric. On the other hand, we considered the pair of techniques involved in each of H_0^h, H_0^{h+1}, ... not to have significant difference with respect to one metric, as these hypotheses were accepted.

An obvious metric for RQ2 is the time required to detect faults. Corresponding to the F-measure and F2-measure, in this study we used $F\text{-}time$ and $F2\text{-}time$ denote the time required (i.e. test cases generation, test cases selection, and test cases execution) to detect the first fault. Obviously, a smaller values of $F\text{-}time$ indicate a better performance.

4.4 Experimental Settings

4.4.1 Generation of Categories and Choices for Object Programs

The categories and choices used for the laboratorial programs and Real-World Popular Programs (FastJson) considered in this study were designed by the authors. On the other hand, the categories and choices used for GNU Program was designed by Barus [45]. In large part, the selection of appropriate categories and choices is at a tester's discretion; we chose what we regarded as simple approaches for emulating that process. Precise details on the categories and choices used in our study are provided as following.

For each laboratorial program (`ACMS`, `CUMS`, `ERS`, `MOS`), categories and choices for its input domain was identified in advance based on its corresponding specification. Based on the identified categories and choices of each labboratorial program, we divided input domain into partitions, generated test cases, and identified MRs. The identified categories and corresponding choices for `ACMS`, `CUMS`, `ERS`, and `MOS` are shown in Table 3, 4, 5, and 6, respectively. Detailed description of categories and choices for laboratorial programs are available at https://github.com/phantomDai/AMT-material.git.

TABLE 3
Definition of Categories and Choices for `ACMS`

#	Categories	Associated Choices
1	class	First calss (1a) Business class (1b) Economy class (1c) Infant (1d)
2	region	Domestic flights (2a) International flights (2b)
3	isStudent	True (3a) False (3b)
4	luggage	Luggage $<$ Free checked-in (4a) Luggage $\leq$ Free checked-in (4b)
5	fee	Fee $= 0$ (5a) Fee ≤ 0 (5b)

TABLE 4
Definition of Categories and Choices for `CUBS`

#	Categories	Associated Choices
1	plan	A (1a) B (1b)
2	options	46CNY (2a) 96CNY (2b) 286CNY (2c) 886CNY (2d) 126CNY (2e) 186CNY (2f)
3	calls	calls $<$ free calls (3a) calls $\geq$ free calls (3b)
4	data	data $<$ free data (4a) data $\geq$ free data (4b)

TABLE 5
Definition of Categories and Choices for `ERS`

#	Categories	Associated Choices
1	title	senior sales manager (1a) sales manager (1b) sales executive (1c)
2	mileage	$0 \leq mileage \leq 3000$ (2a) $3000 \leq mileage \leq 4000$ (2b) $mileage \geq 4000$ (2c)
3	sales	$0 \leq sales \leq 50,000$ (3a) $50,000 \leq sales \leq 80,000$ (3b) $0 \leq sales \leq 3000$ (3c) $80,000 \leq sales \leq 100,000$ (3d) $sales \geq 100,000$ (3e)
4	airline ticket(AT)	$AT = 0$ (4a) $AT \leq 0$ (4b)
5	other expenses(OE)	$OE = 0$ (5a) $OE \leq 0$ (5b)

TABLE 6
Definition of Categories and Choices for `MOS`

#	Categories	Associated Choices
1	Aircraft Model	747200 (1a) 747300 (1b) 747400 (1c) 002000 (1d) 003000 (1e)
2	Change in the Number of Crews	Yes (2a) No (2b)
3	Number of Crews(NC)	$NC >$ default value (3a) $NC =$ default value (3b) $NC <$ default value (3c)
4	Change in the Number of pilots	Yes (4a) No (4b)
5	Number of pilots(NP)	$NP >$ default value (5a) $NP =$ default value (5b) $NP <$ default value (5c)
6	Number of Child Passengers	$childNum > 0$ (6a) $childNum = 0$ (6b)
7	Number of Bundles Flowers	$NF > 0$ (7a) $NF = 0$ (7b)

For `grep`, Barus et. al have made use the existing testing information about grep recorded in the SIR [51], conducting categories and choices. The exact details of categories and choices for `grep` are presented in [44]. Furthermore, they divided the categories into two groups — the independent and the dependent categories (The details of the dependent and independent categories are listed in Table 7). The independent group includes all categories that contain elements which can form a valid test case on their own. Dependent categories need the presence of elements of other categories to form valid test cases.

To define the categories and choices of `FastJson`, the specification and user documentation were first examined, however they were either not discovered or did not provide signicant detail to construct appropriate categories and choices. Therefore, we also made use the existing best practices, the set of test cases and source codes about `FastJson` recorded in the Alibaba repository[3]. As mentioned, we restrict our experimentation to testing the deserialization of `FastJson`. As a consequence, our categories-choices scheme considers only on this portion of `FastJson`. Deserialization is the process of translating a stored format (For example, a Json file or memory buffer) into an object state, which contains the values of all the variables in a program. Therefore, We consider the base member variables and their combinations. All identified categories and choices of `FastJson` are listed in Table 8. The Detailed description of categories and choices for `FastJson` are available at our repository: https://github.com/phantomDai/AMT-material.git.

3. The best practices and test suite are available at: https://github.com/alibaba/fastjson/wiki

TABLE 7
Table of Independent and Dependent categories for `grep`

Independent Categories	Dependent Categories
NormalChar	Bracket
WordSymbol	Iteration
DigitSymbol	Parentheses
SpaceSymbol	Line
NamedSymbol	Word
AnyChar	Edge
Range	Combine

TABLE 8
Definition of Categories and Choices for `FastJson`

#	Categories	Associated Choices
1	Float	Exist Inexistence
2	Double	Exist Inexistence
3	Short	Exist Inexistence
4	Byte	Exist Inexistence
5	Int	Exist Inexistence
6	Long	Exist Inexistence
7	Boolean	Exist Inexistence
8	Char	Exist Inexistence
9	Date	Exist Inexistence
10	String	Exist Inexistence
11	Enum	Exist Inexistence
12	Map	Exist Inexistence
13	List	Exist Inexistence
14	Set	Exist Inexistence

4.4.2 Generation of Test Cases for Object Programs

For `CUBS`, `ACMS`, and `ERS`, we used a generator that is based on the complete test frame devised for MT-related techniques selection. We systematically generated a test case for each complete test frame, which were collectively guaranteed to cover each category and choice. The final test suites contained 284, 1470, and 2260, respectively. All complete test frame are available at: https://github.com/phantomDai/AMT-material.git.

For `grep`, we used the test suite generated by Barus, who has described the test case generation process in detail [45]. The original test suite generated by Barus contains 171,634 inputs. During the test process, we found that some of the test frames containe the same choices, but in a different order. In practice, each test frame corresponds to a partition in which test cases cover some paths, thus, those test frames containe the same choices cover same or similar paths. we removed those test frames, and obtained 101,193 test frames, which are available at our repository: https://github.com/phantomDai/AMT-material.git.

As for `FastJson`, we used a generator that is based on the complete test frame devised for MT-related techniques selection. We systematically generated a test case for each complete test frame, which were collectively guaranteed to cover each category and choice. The final test suites contained 16383 Java object and 16383 conrrsponding Json files. All complete test frames are available at: https://github.com/phantomDai/AMT-material.git.

4.4.3 Partitioning

The proposed technique AMT based on the partition testing that is a mainstream family of software testing techniques, which can be realized in different ways, such as Intuitive Similarity, Equivalent Paths, Risk-Based, and Specified As Equivalent (two test values are equivalent if the specification says that the program handles them in the same way). In this study, we made use of the CPM to conduct the partitioning. Our previous work [34] found that there is not a strong correlation between the granularity level in partitioning and the performance of APT.

We randomly select two categories for `ACMS`, `CUBS`, `ERS`, and `MOS`, and partition their input domain according to the combinations of choices.We use an explanatory example to illustrate the method of partitioning. According to the description of Section 4.2.1, we identify categories and associated choices of `ACMS`, as shown in Table 9. With these categories/choices and constraints among choices, we further derive a set of complete test frames and each of them corresponds to a partition. Note that the number of partitions may vary with the granularity level. To ease the illustration, we derive partitions without considering the baggage weight category. Table 10 shows the resulting partitions for `ACMS`, where $weight_*$ indicates no classification on the baggage weight.

Barus has divided the categories into two groups — the independent and the dependent categories [44]. The independent group includes all categories that contain elements which can form a valid test case on their own. Dependent categories need the presence of elements of other categories to form valid test cases. We obtained two partitioning schemes by choosing independent categories to ignore dependent categories and choosing dependent categories to ignore independent categories. As a consequence we constructed 550 and 3380 partitions, respectively. We finally selected the first partition scheme (550 partitions), since the second partition scheme tends to result in the repeated selection of the same test case during testing.

4.4.4 Initial Test Profile

The proposed techniques M-AMT and R-AMT make use of feedback information to adjust the test profile. Then the updated test profile guide those techniques to select next source test case and MR. Before executing the test, a concrete test profile should be initialized. Because test cases may be generated randomly during the test process, a feasible method is to use a uniform probability distribution as the

TABLE 9
Categories and choices of ACMS

Category	Associated choices
aircraft cabin	*Economy, Business, First Class*
flight region	*International, Domestic*
baggage weight	*Above Limit, Below Limit*

TABLE 10
Partitions of ACMS

Partition	Complete Test Frame
c_1	$\{International, Economy, weight_*\}$
c_2	$\{International, Business, weight_*\}$
c_3	$\{International, FirstClass, weight_*\}$
c_4	$\{Domestic, Economy, weight_*\}$
c_5	$\{Domestic, Business, weight_*\}$
c_6	$\{Domestic, FirstClass, weight_*\}$

initial testing profile. On the other hand, in our previous work [34], we have compared the equal and proportional initial test profile in terms of associated performance metrics (F-measure, F2-measure [34], and T-measure). The comparison results that there was no significant difference between these two types of initial test profiles. In our experiment, we used a uniform probability distribution for the initial test profile. The initial test profiles of each subject program is summarized in Table 11, where $< s_i, p_i >$ means that the probability of selecting partition s_i is p_i.

4.4.5 Constants

Previous studies [34], [52]–[54] have given some guidelines on how to set ε and δ of RAPT. We followed these studies to set $\varepsilon = 0.05$ and calculate the proper δ according to the following formula (which is extracted from [54]):

$$\frac{1}{\theta_M} - 1 < \frac{\varepsilon}{\delta} < \frac{1}{\theta_\Delta} - 1, \tag{33}$$

where θ_M and θ_Δ are the largest and the second largest failure rates amongst all partitions, respectively. Note that the value of θ_Δ was different for different versions of `FastJson`. According to Formula 33, the theoretically values of δ in each program are shown in Table 12.

To set the values of γ and τ for MAPT, we followed our previous study [34], in which we conducted a series of preliminary experiments. We observed that γ and τ could neither be too large nor too small, which imply that the change in probability profile would be either very dramatic or very marginal. Both situations might result in the nfair? (either too big or too small) award/punishment to certain partitions. After several rounds of trials, we concluded that $\gamma = \tau = 0.1$ were fair settings. All faults in our experiments are non-trivial and thus not easy to be killed. Thus, the value of Boui could not be too small. We set $Bou_i = 70\% \times k_i$, where k_i is the number of test cases selected from c_i.

TABLE 11
Initial Test Profile for Subject Program

Program	Number of partitions	Initial test profile
`ACMS`	8	$\{< s_1, \frac{1}{8} >, < s_2, \frac{1}{8} >, \ldots, < s_8, \frac{1}{8} >\}$
`CUBS`	4	$\{< s_1, \frac{1}{4} >, < s_2, \frac{1}{4} >, \ldots, < s_4, \frac{1}{4} >\}$
`ERS`	12	$\{< s_1, \frac{1}{12} >, < s_2, \frac{1}{12} >, \ldots, < s_{12}, \frac{1}{12} >\}$
`MOS`	10	$\{< s_1, \frac{1}{10} >, < s_2, \frac{1}{10} >, \ldots, < s_{10}, \frac{1}{10} >\}$
`grep`	550	$\{< s_1, \frac{1}{550} >, < s_2, \frac{1}{550} >, \ldots, < s_{550}, \frac{1}{550} >\}$
`FastJson`	128	$\{< s_1, \frac{1}{128} >, < s_2, \frac{1}{128} >, \ldots, < s_{550}, \frac{1}{128} >\}$

TABLE 12
Theoretical Values of δ

Program	θ_M	θ_Δ	δ
`ACMS`	2.86E-1	1.04E-1	1.72E-2
`CUBS`	2.50E-1	2.78E-2	2.60E-2
`ERS`	6.00E-1	2.93E-1	1.98E-2
`MOS`	1.59E-1	1.57E-1	9.42E-3
`grep`	9.41E-1	8.82E-1	4.20E-1
`FastJson`	5.80E-2	5.33E-2	2.77E-3

4.5 Identification of MRs for Object Programs

4.5.1 The MRs of Laboratorial Programs

In our study, to obtain metamorphic relations of `ACMS`, `CUBS`, `ERS`, and `MOS`, we made use of METRIC [55], which is based on the category choice framework [35]. In METRIC, only two distinct complete test frames that are abstract test cases defining possible combinations of inputs, are considered by the tester to generate an MR. In general, METRIC has the following steps to identify MRs:

- Step1. Users select two relevant and distinct complete test frames as a *candidate pair*.
- Step2. Users determine whether or not the selected candidate pair is useful for identifying an MR, and if it is, then provide the corresponding MR description.
- Step3. Restart from step 1, and repeat until all candidate pairs are exhausted, or the predefined number of MRs to be generated is reached.

Following the above guidelines, we identified MRs for `CUBS`, `ACMS`, and `ERS`. Table 13 presents a part of MRs of those programs. In the table, O_1 and O_2 denote the output of source test case and follow-up test case, respectively. $NMC.0$ indicates the number of crew meals have not changed; $NMP.0$ indicates the number of passenger meals have not changed; $NMP.1$ indicates the number of passenger meals have increased; $NMCP.0$ indicates the number of child meals have not changed; $NMCP.1$ indicates the number of child meals have increased; $NMCP.-1$ indicates the number of passenger meals have descreased; $NF.0$ indicates the number of flowers have not changed; and $NF.1$ indicates the number of flowers have increased.

The development of the MRs for `grep` are restricted to the test cases of `grep`. Specifically, these MRs are only associated with the regular expression parameter of `grep`'s input. The details of MRs are provided in [56].

Based on the genearated test cases, best practice, and all faults reported in Alibaba repository, we consducted MRs for `FastJson`. Accordingly, we identified twenty MRs particularly for testing the deserialization of `FastJson`. The details of MRs are available at: https://github.com/phantomDai/AMT-material.git.

TABLE 13
Part of MRs of Laboratorial Programs

Program	MRs		
	Source Test Cases	Follow-up Test Cases	Relations
CUBS	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2a, 3a, 4b\}$	$O_1 \geq O_2$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2a, 3b, 4a\}$	$O_1 \geq O_2$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2a, 3b, 4b\}$	$O_1 \geq O_2$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2b, 3a, 4a\}$	$O_1 \geq O_2$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2b, 3a, 4b\}$	$O_1 \geq O_2$
ACMS	$\{1a, 2a, 3a, 4a, 5a\}$	$\{1a, 2a, 3a, 4a, 5b\}$	$O_1 = O_2 = 0$
	$\{1a, 2a, 3a, 4a, 5a\}$	$\{1a, 2a, 3a, 4b, 5a\}$	$O_1 = O_2 = 0$
	$\{1a, 2a, 3a, 4a, 5a\}$	$\{1a, 2b, 3a, 4a, 5a\}$	$O_1 = O_2 = 0$
	$\{1a, 2a, 3a, 4a, 5a\}$	$\{1b, 2a, 3a, 4a, 5a\}$	$O_1 = O_2 = 0$
	$\{1a, 2a, 3a, 4a, 5a\}$	$\{1c, 2a, 3a, 4a, 5a\}$	$O_1 = O_2 = 0$
CUBS	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2a, 3a, 4b\}$	$O_1 \leq O_2$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2a, 3b, 4a\}$	$O_1 = O_2 = 0$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2a, 3c, 4a\}$	$O_1 = O_2 = 0$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2b, 3a, 4a\}$	$O_1 = O_2 = 0$
	$\{1a, 2a, 3a, 4a\}$	$\{1a, 2c, 3a, 4a\}$	$O_1 \geq O_2$
MOS	$\{1a, 2b, 3b, 4b, 5b, 6b, 7b\}$	$\{1a, 2b, 3b, 4b, 5b, 6b, 7a\}$	$NMC.0, NMP.0, NMCP.0, NF.1$
	$\{1a, 2b, 3b, 4b, 5b, 6b, 7b\}$	$\{1a, 2b, 3b, 4b, 5b, 6a, 7b\}$	$NMC.0, NMP.0, NMCP.1, NF.0$
	$\{1a, 2b, 3b, 4b, 5b, 6b, 7b\}$	$\{1a, 2b, 3b, 4b, 5b, 6a, 7a\}$	$NMC.0, NMP.0, NMCP.1, NF.1$
	$\{1a, 2b, 3b, 4b, 5b, 6b, 7b\}$	$\{1a, 2b, 3b, 4a, 5c, 6b, 7b\}$	$NMC.0, NMP.0, NMCP.-1, NF.0$
	$\{1a, 2b, 3b, 4b, 5b, 6b, 7b\}$	$\{1a, 2b, 3b, 4a, 5a, 6b, 7b\}$	$NMC.0, NMP.1, NMCP.0, NF.0$

4.6 Experimental Environment

Our experiments were conducted on a virtual machine running the Ubuntu 18.04 64-bit operating system. In this system, there were two CPUs and a memory of 4GB. The test scripts were generated using Java, bash shell, and Python. In the experiments, we repeatedly ran the testing using each technique 30 times [57] with different random seeds to guarantee the statistically reliable mean values of the metrics.

4.7 Threats To Validity

4.7.1 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.

4.7.2 External Validity

The possible threat to external validity is related to the subject programs and seeded faults under evaluation. Although the four laboratory programs are not very complex, they do implement real-life business scenarios of diverse application domains. Furthermore, we chose two open source programs (`grep and FastJson`). Although we have tried to improve the generalisability of the findings by applying subject programs in different areas, we anticipate that the evaluation results may vary slightly with different subject web services.

4.7.3 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.

4.7.4 Conclusion Validity

As reported for empirical studies in the field of software engineering [57], 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, we also conducted statistical tests to confirm the significance of the results.

5 Experimental Results

5.1 RQ1: Fault-detection efficience

F-, and F2-results for `ACMS`, `CUBS`, `PBS`, `MOS`, `grep`, and `FastJson` are shown using boxplots in Figures 2 and 3, where AMT_1, AMT_2, AMT_3, and AMT_4 denotes $MR-AMT$, $MP-AMT$, and $RR-AMT$, and $RP-AMT$, respectively. 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.

From Figures 2 and 3, we can observe that in general, RP-AMT was the best performer in terms of F-measure and F2-measure, followed by MP-AMT, MR-AMT, RR-AMT, and MT in descending order. We further conducted hypothesis testing to verify the statistical significance of this observation. We used the Holm-Bonferroni method [58] to determine which pair of testing techniques have significant difference in terms of each metric. Across the whole study, for each pair of testing techniques, denoted by technique a and technique b, the null hypothesis (H_0) was that a and b had the similar performance in terms of one metric. All the null hypotheses were ordered by their corresponding p-values, from lowest to largest; in other words, for null hypotheses H_0^1, H_0^2, $\ldots$, we had $p_1 \leq p_2 \leq \ldots$. For the given confidence level $\alpha = 0.05$, we found the minimal index h such that $p_h > \frac{\alpha}{N+1-h}$ (where N is the total number of null hypotheses). Then, we rejected H_0^1, H_0^2, $\ldots$, H_0^{h-1}, that is, we regarded the pair of techniques involved in each of these hypotheses to have statistically significant difference in terms of a certain metric. On the other hand, we considered the pair of techniques involves in each of H_0^h, H_0^{h+1}, $\ldots$ not to have significant difference with respect to one metric, as these hypotheses were accepted. The statistical testing results are shown in Tables 14 and 15. Each cell in the tables gives the number of scenarios where the technique on the top row performed better than that on the left column. If the difference is significant, the number will be displayed in bold. For example, the "131" in the top right corner in Table 14 indicates that out of 180 scenarios (6 programs repeatedly ran), RP-AMT had smaller F-measure than MT for 131 scenarios. Correspondingly, the "49" in the bottom left corner in Table 14 means that the F-measure of MT was smaller than that of RP-AMT for only 49 scenarios. The corresponding null hypothesis was that MT and RP-AMT had similar performance in terms of F-measure. Since this hypothesis was rejected, we could say that the fault-detection capabilities of RAPT and RPT were significantly different in terms of F-measure. This statistically significance difference is represented by bold font of "131" and "49", which further indicates that RAPT was significantly better than RPT.

TABLE 14
Number of Scenarios where the Technique on the top row has a lower metric (F-measure) score than the technique on the left column

	MT	MR-AMT	MP-AMT	RR-AMT	RP-AMT
MT	—	**124**	**128**	**119**	**131**
MR-AMT	56	—	105	86	106
MP-AMT	52	75	—	92	102
RR-AMT	61	94	88	—	110
RP-AMT	49	74	78	70	—

5.2 RQ2: Selection Overhead

The detailed experimental results of F-time and F2-time are shown in Figures 4 and 5. From Figures 4 and 5, we can observe that in general, RP-AMT had the best performance, followed by RR-AMT, MR-AMT, and MP-AMT. Note that MR-AMT, MP-AMT, RR-AMT, and RP-AMT do not consistently delivered the best performance for all object programs in terms of both F-time and F2-time, while the larger the program, the better the AMT performs. As we can observe from figure 2, the execution of the test cases accounts for most of the overhead. AMT compensates for the overhead of selecting test cases by reducing the number of test cases executed.

TABLE 15
Number of Scenarios where the Technique on the top row has a lower metric (F2-measure) score than the technique on the left column

	MT	MR-AMT	MP-AMT	RR-AMT	RP-AMT
MT	—	**142**	**148**	**136**	**156**
MR-AMT	38	—	135	118	145
MP-AMT	32	45	—	121	139
RR-AMT	44	62	59	—	143
RP-AMT	24	35	41	37	—

We also conducted hypothesis testing to determine which pair of testing techniques have significant difference in terms of F-time and F2-time, as shown in Tables 16 and 17.

From Table 16 and 17, we can observe that fourteen entries ("102" & "78" for MR-AMT versus MT, "112" & "68" for MP-AMT versus MT, "91" & "89" for RR-AMT versus MT, and "120" & "60" for RP-AMT versus MT, "96" & "84" for MR-AMT versus MT, "114" & "66" for RR-AMT versus MT, and "128" & "52" for RP-AMT versus MT) are in bold font. These observations imply that in terms of F-time, AMT performed better than MT ("87" & "93" for MP-AMT versus MT). On the other hand, the performance of RP-AMT was significantly better than that of the other four techniques. In other words, the additional computation incurred in AMT for updating the probability profiles and selecting MRs is compensated with the saving of test executions.

TABLE 16
Number of Scenarios where the Technique on the top row has a lower metric (F-time) score than the technique on the left column

	MT	MR-AMT	MP-AMT	RR-AMT	RP-AMT
MT	—	**102**	**112**	**91**	**120**
MR-AMT	**78**	—	98	71	103
MP-AMT	**68**	82	—	69	97
RR-AMT	**89**	109	111	—	113
RP-AMT	**60**	77	83	67	—

TABLE 17
Number of Scenarios where the Technique on the top row has a lower metric (F2-time) score than the technique on the left column

	MT	MR-AMT	MP-AMT	RR-AMT	RP-AMT
MT	—	**96**	87	**114**	**128**
MR-AMT	**84**	—	105	116	104
MP-AMT	93	75	—	103	107
RR-AMT	**66**	64	77	—	118
RP-AMT	**52**	76	73	62	—

6 Related Work

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

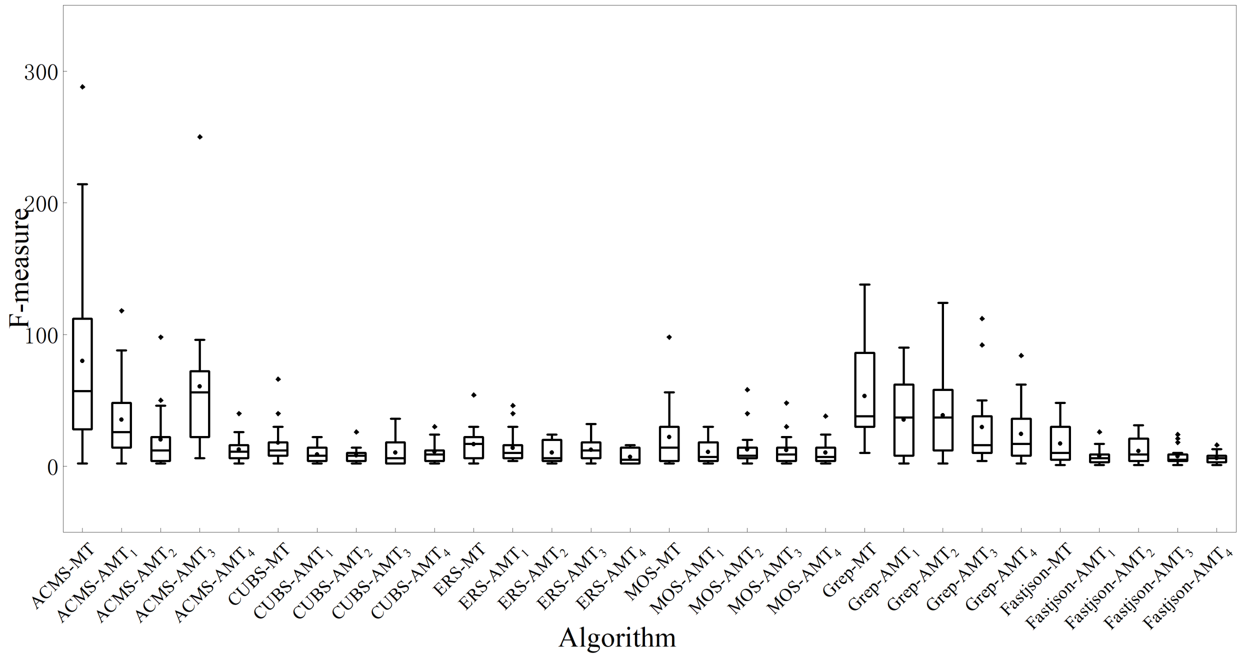


Fig. 2. F-measure boxplots for each program

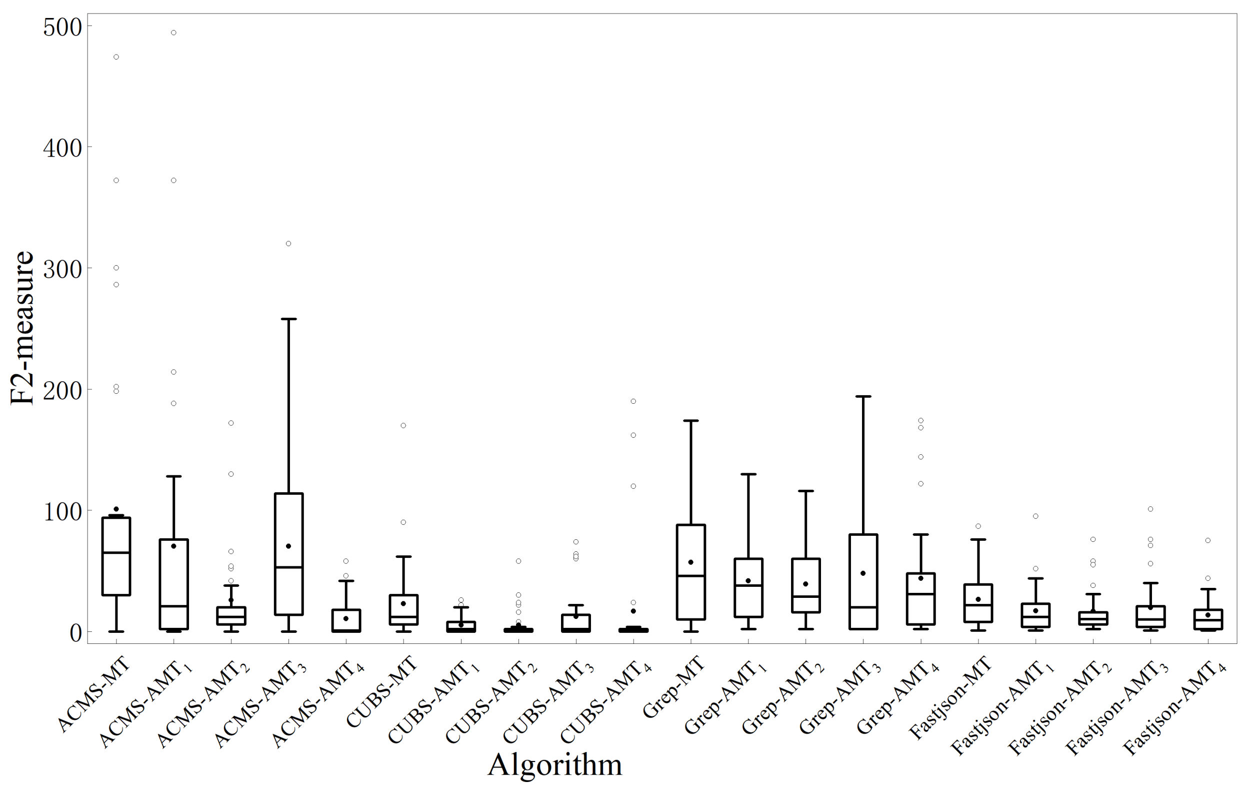
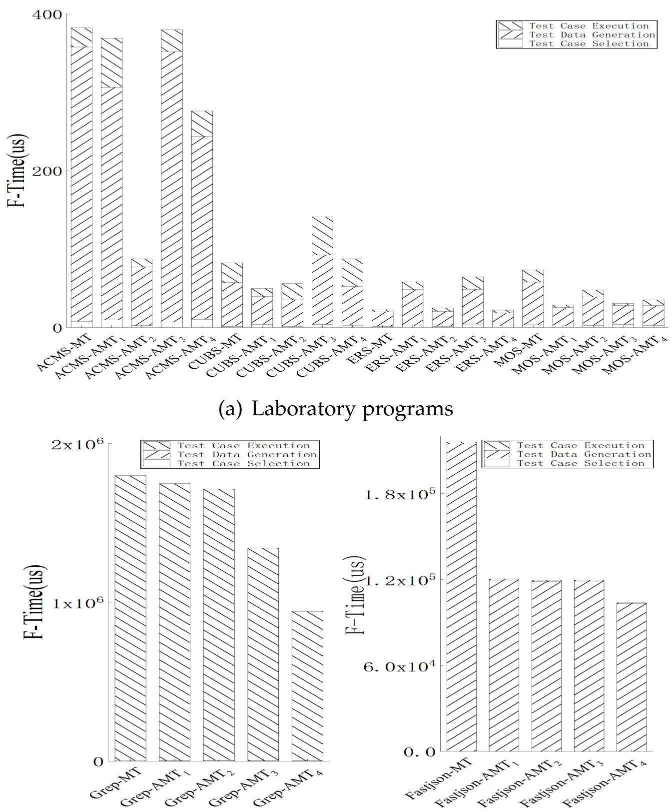


Fig. 3. F2-measure boxplots for each program

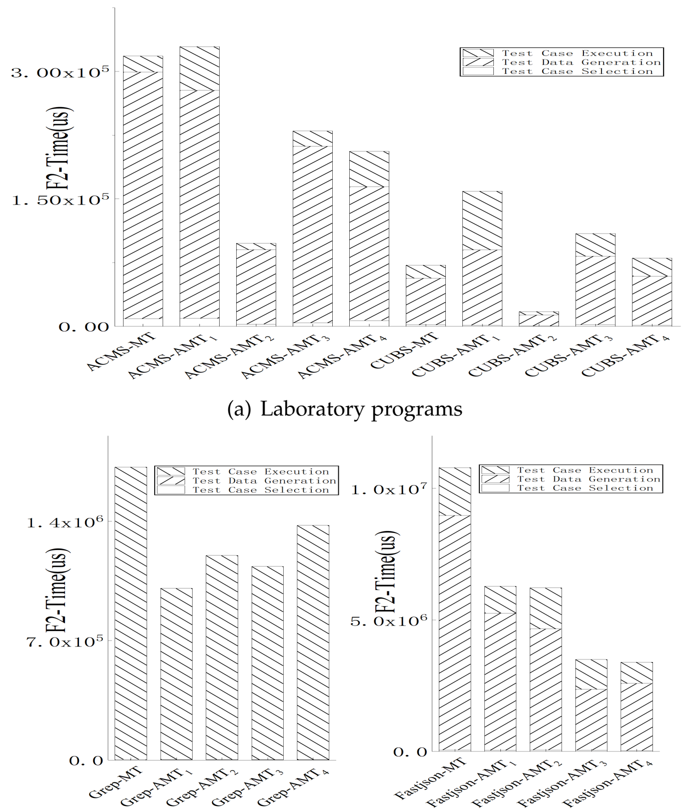


(a) Laboratory programs

(b) GNU grep

(c) Alibaba FastJson

Fig. 4. The F-time results of test case selection, generation, and execution.



(a) Laboratory programs

(b) GNU grep

(c) Alibaba FastJson

Fig. 5. The F2-time results of test case selection, generation, and execution.

6.1 Testing techniques of MT

When testing a software system, the oracle problem appears in some situations where either an oracle does not exist for the tester to verify the correctness of the computed results; or an oracle does exist but cannot be used. The oracle problem often occurs in software testing, which renders many testing techniques inapplicable [2]. To alleviate the oracle problem, Chen et al. [5] proposed a technique named metamorphic testing (MT) that has been receiving increasing attention in the software testing community [2], [6], [26]. The main contributions to MT in the literature focused on the following aspects: i) MT theory; ii) combination with other techniques; iii) application of MT.

1 *Theoretical development of MT:* The MRs and the source test cases are the most important components of MT. However, defining MRs can be difficult. Chen et al. [55] proposed a specification-based method and developed a tool called MR-GENerator for identifying MRs based on category-choice framework [35]. Zhang et al. [59] proposed a search-based approach to automatic inference of polynomial MRs for a software under test, where a set of parameters is used to represent polynomial MRs, and the problem of inferring MRs is turn into a problem of searching for suitable values of the parameters. Then, particle swarm optimization is used to solve the search problem. Sun et al. [60] proposes a data-mutation directed metamorphic relation acquisition methodology, in which data mutation is employed to construct input relations and the generic mapping rule associated with each mutation operator to construct output relations. Liu et al. [61] proposed to systematically construct MRs based on some already identified MRs.

~~Without doubt~~, "good" MRs can improve the fault detection efficiency of MT. Chen et al. [19] reported that good MRs are those that can make the execution of the source-test case as different as possible to its follow-up test case. This perspective has been confirmed by the later studies [24], [25]. Asrafi et al. [29] conduct a case study to analyze the relationship between the execution behavior and the fault-detection effectiveness of metamorphic relations by code coverage criteria, and the results showed a strong correlation between the code coverage achieved by a metamorphic relation and its fault-detection effectiveness. Source test cases also have a important impact on the fault detection effectiveness of MT. Chen et al. [17] compared the effects of source test cases generated by special value testing and random testing on the effectiveness of MT, and found that MT can be used as a complementary test method to special value testing. Batra and Sengupta [24] integrated genetic algorithms into MT to select source test cases maximising the the paths traversed in the software under test. Dong et al. [25] proposed a Path–Combination–Based MT method that first generates symbolic input for each executable paths and minis relationships among these symbolic inputs and their outputs, then constructs MRs on the basis of these relationships, and generates actual test cases corresponding to the symbolic inputs.

Different from the above investigates, we focused on performing test cases and MRs with fault revealing capabilities as quickly as possible by making use of feedback information. We first divided the input domain into disjoint partitions, and randomly selected an MR to generate follow-up test cases depended on source test case of related input partitions, then updated the test profile of input partitions according to the results of test execution. Next, a partition was selected according to updated test profile, and an MR was randomly selected from the set of MRs whose source test cases belong to selected partition.

2 *Combination with other techniques:* In order to improve the applicability and effectiveness of MT, it has been integrated into other techniques. Xie et al. [62] combined the MT with the spectrum-based fault localization (SBFL), extend the application of SBFL to the common situations where test oracles do not exist. Dong et al. [63] proposed a method for improving the efficiency of evolutionary testing (ET) by considering MR when fitness function is constructed. Liu et al. [64] introduced MT into fault tolerance and proposed a theoretical framework of a new technique called Metamorphic Fault Tolerance (MFT), which can handle system failure without the need of oracles during failure detection. In MFT, the trustworthiness of a test case depends on the number of violations or satisfactions of metamorphic relations. The more relations are satisfied and the less relations are violated, the more trustable test case is.

3 *Application of MT:* Sun et al. [21], [65] proposed a metamorphic testing framework for web services taking into account the unique features of SOA, in which MRs are derived from the description or Web Service Description Language (WSDL) [21] of the Web service, and on the basis on of MRs, follow-up test cses are generated depended on source test cases that are randomly generated according to the WSDL. Segura et al. [13] present a metamorphic testing approach for the detection of faults in RESTful Web APIs where they proposed six abstract relations called Metamorphic Relation Output Patterns (MROPs) that can then be instantiated into one or more concrete metamorphic relations. To evaluate this approach, they used both automatically seeded and real faults in six subject Web APIs.

6.2 Improving RT and PT

Based on the observation that failure-causing inputs tend to cluster into contiguous regions in the input domain [66], [67], much work has been done to improve RT [68]–[70]. Adaptive random testing [69], [70] is a family of 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, selects a next test input from the fixed-size candidate set of tests that is farthest from all previously executed tests [27]. Many other ART algorithms have also been proposed, including RRT [71], DF-FSCS [72], and ART-sum [45], with their effectiveness examined and validated through simulations and experiments.

Adaptive testing (AT) [73]–[75] takes advantage of feedback information to control the execution process, and has been shown to outperform 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. However, AT may require a rather long execution time in practice. To alleviate this, Cai et al. [68] 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. [76] proposed A-DRT, which adjusts parameters during the testing process.

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 (SEHC'11), Co-located with the 33th International Conference on Software Engineering (ICSE'11)*, 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, pp. 24–32, 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, no. 4, pp. 6–12, 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 (ICSE'18)*, 2018, pp. 303–314.

[15] D. Marijan, A. Gotlieb, and M. K. Ahuja, "Challenges of testing machine learning based systems," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest'19)*, 2019, pp. 101–102.

[16] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020, DOI:10.1109/TSE.2019.2962027.

[17] 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.

[18] 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)*, 2016, pp. 5–11.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] G. Batra and J. Sengupta, "An efficient metamorphic testing technique using genetic algorithm," in *Proceedings of the 3rd International Conference on Information Intelligence, Systems, Technology and Management (ICISTM'11)*, 2011, pp. 180–188.

[25] G. Dong, T. Guo, and P. Zhang, "Security assurance with program path analysis and metamorphic testing," in *Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science (ICSESS'13)*, 2013, pp. 193–197.

[26] 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.

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

[28] J. Mayer and R. Guderlei, "An empirical study on the selection of good metamorphic relations," in *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, 2006, pp. 475–484.

[29] M. Asrafi, H. Liu, and F.-C. Kuo, "On testing effectiveness of metamorphic relations: A case study," in *Proceedings of the 15th International Conference on Secure Software Integration and Reliability Improvement (SSIRI'11)*, 2011, pp. 147–156.

[30] 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, 2013.

[31] X. Xie, J. Tu, T. Y. Chen, and B. Xu, "Bottom-up integration testing with the technique of metamorphic testing," in *Proceedings of 14th International Conference on Quality Software (QSIC'14)*, 2014, pp. 73–78.

[32] J. W. Cangussu, K.-Y. Cai, S. D. Miller, and A. P. Mathur, "Software cybernetics," *Wiley Encyclopedia of Computer Science and Engineering*, 2007.

[33] H. Yang, F. Chen, and S. Aliyu, "Modern software cybernetics: New trends," pp. 157–169, 2017.

[34] C.-A. Sun, H. Dai, H. Liu, T. Y. Chen, and K.-Y. Cai, "Adaptive partition testing," *IEEE Transactions on Computers*, vol. 68, no. 2, pp. 157–169, 2019.

[35] 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.

[36] 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.

[37] 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.

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

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

[40] 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.

[41] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "Mujava: an automated class mutation system," *Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[42] 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.

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

[44] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, R. Merkel, and G. Rothermel, "A novel linear-order algorithm for adaptive random testing of programs with non-numeric inputs," *Technical Report TR-UNL-CSE-2014-0004*, 2014.

[45] ——, "A cost-effective random testing method for programs with non-numeric inputs," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3509–3523, 2016.

[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] 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.

[48] 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.

[49] C.-A. Sun, H. Dai, G. Wang, D. Towey, T. Y. Chen, and K.-Y. Cai, "Dynamic random testing of web services: A methodology and evaluation," *IEEE Transactions on Services Computing*, 2019, available online: https://doi.org/10.1109/TSC.2019.2960496.

[50] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian journal of statistics*, pp. 65–70, 1979.

[51] 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.

[52] 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.

[53] 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.

[54] 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 International Computer Software and Applications Conference (COMPSAC15)*, vol. 3, 2015, pp. 466–471.

[55] T. Y. Chen, P.-L. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 177–190, 2016.

[56] A. C. Barus, "An in-depth study of adaptive random testing for testing program with complex input types," Ph.D. dissertation, Ph. D. Thesis. Faculty of Information and Communication Technologies , 2010.

[57] 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.

[58] H. Sture, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, pp. 65–70, 1979.

[59] 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.

[60] C.-A. Sun, Y. Liu, Z. Wang, and W. Chan, "μmt: A data mutation directed metamorphic relation acquisition methodology," in *2016 IEEE/ACM 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.

[61] 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.

[62] 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.

[63] G. Dong, S. Wu, G. Wang, T. Guo, and Y. Huang, "Security assurance with metamorphic testing and genetic algorithm," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'10)*, vol. 3, 2010, pp. 397–401.

[64] H. Liu, I. I. Yusuf, H. W. Schmidt, and T. Y. Chen, "Metamorphic fault tolerance: An automated and systematic methodology for fault tolerance in the absence of test oracle," in *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, 2014, pp. 420–423.

[65] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "A metamorphic relation-based approach to testing web services without oracles," *International Journal of Web Services Research*, vol. 9, no. 1, pp. 51–73, 2012.

[66] 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.

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

[68] 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.

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

[70] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, and X. Xia, "A survey on adaptive random testing," *IEEE Transactions on Software Engineering*, 2019, DOI: 10.1109/TSE.2019.2942921.

[71] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing: Adaptive random testing by exclusion," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 04, pp. 553–584, 2006.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

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.

Huai Liu is a Lecturer of Information Technology at the College of Engineering & Science in Victoria University, Melbourne, Australia. Prior to joining VU, he worked as a research fellow at RMIT University and a research associate at Swinburne University of Technology. He received the BEng in physioelectronic technology and MEng in communications and information systems, both from Nankai University, China, and the PhD degree in software engineering from the Swinburne University of Technology, Australia. His current research interests include software testing, cloud computing, and end-user software engineering.