Adaptive Metamorphic Testing*

1st Chang-ai Sun, *Senior Member, IEEE,*
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, China
casun@ustb.edu.cn

2nd Hepeng Dai
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, China
daihepeng@sina.cn

3rd Tsong Yueh Chen
Faculty of Information and Communication Technologies
Swinburne University of Technology Australia
Melbourne, Australia
tychen@swin.edu.au

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

***Abstract*—Metamorphic testing (MT) is a promising technique to alleviate the oracle problem, which fist defines metamorphic relations (MRs) used to generate new test cases (i.e. follow-up test cases) from the original test cases (i.e. source test cases). Both source and follow-up test cases are executed and their results are verified against the relevant MRs. Up to now, in terms of improving the efficiency of MT, researchers have focused their efforts on generating or selecting better MRs, while ignoring the impact of source test cases. Most of MT techniques employ random testing strategy (RT) to select source test cases. Although RT is simple and easy to use, it may not be efficient because RT does not use any information of the software under test. This study aims to improve the efficiency of MT through controlling the execution process of MT, and proposes a adaptive metamorphic testing (AMT) technique based on partition. We conduct a empirical study where AMT is employed to test three real-life programs. The results of empirical study show the feasibility of AMT, and AMT outperforms RT on enhancing the efficiency of MT.**

***Index Terms*—metamorphic testing, control test process, partition**

I. Introduction

Test result verification is an important part of software testing. A test oracle [1] is mechanism that can exactly decide whether or not 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].

Metamorphic testing (MT) [4], [5] is one of several techniques to alleviate oracle problem. MT first define metamorphic relations (MRs) through using some 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 follow-up and source test cases are executed and their result are verified against the corresponding MRs.

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

Obviously, the efficiency of MT in detecting faults relay on the quality of MRs and the source test cases. There have been vast studies to investigate generate and select good MRs [6]–[11]. However, Researchers ignore the impact of source test cases 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), is most commonly used technique in MT [12]. Although RT is simple to implement, RT does not make use of any information about SUT, or the test history. Thus, RT may be inefficient in some situations.

In contrast to RT, partition testing (PT) attempts to generate test cases in a more systematic way, aiming to use fewer test cases to reveal more faults. When conducting PT, the input domain of the SUT is divided into disjoint partitions, with test cases then selected from each and every one. Each partition is expected to have a certain degree of homogeneityłtest cases in the same partition should have similar software execution behavior. Ideally, a partition should also be homogeneous in fault detection: If one input can reveal a fault, then all other inputs in the same partition should also be able to reveal a fault.

This paper proposes a adaptive metamorphic testing framework based on PT to improve the efficiency of MT. We investigate how to make use of feedback information to control the execution process of MT, and conduct an empirical study. The paper makes the following contributions:

- *An adaptive metamorphic testing framework* which shows how to control the execution process of MT. The framework combines the basic principle of MT with the software cybernetics [13].
- *An MRs-centric adaptive metamorphic testing technique (M-AMT)* which adds a feedback mechanism to control the execution process of MT. First, M-AMT 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 results of test execution. Second, a partition is selected according to updated test profile, and an MR is randomly selected

from the set of MRs whose source test cases belong to selected partition.

- *A supporting tool of adaptive metamorphic testing called APT2MT* has been developed to further improve the automation of the proposed AMT, which has features such as termination condition setting, partition setting, test execution, and test report generation.
- *An empirical study* that has been conducted to evaluate the fault detection efficiency of the proposed AMT , where three Java programs are selected as subjects to compare the performance of traditional MT and AMT in terms of fault detection efficiency and time overhead in test case selection.

The rest of the paper is organized as follows. Section II introduces underlying concepts related to MT, adaptive partition testing, and mutation analysis. Section III gives the motivation of this study. Section IV presents a framework of MT for Web services. Section V reports a case study where MT is employed to test a Web service implementing the electronic payment. Related work is discussed in Section VI.

II. Background

A. Metamorphic Testing

A test oracle is a mechanism used to verify the correctness of outputs of a program [1]. However, there are test oracle problem [2], [3] in testing, that is, there are not an oracle or the application of such an oracle is very expensive. In order to alleviate the test oracle, several techniques have been proposed such as N-version testing [14], metamorphic testing (MT) [4], assertions [15], machine learning [16], etc. Among of them, MT obtains metamorphic relations (MRs) according to the properties of software under test (SUT). MRs are used to generate follow-up test cases from source test cases, then the both source and follow-up test cases are executed, and their results are verified against the corresponding MRs. If an MR is violated, that is, a fault is detected.

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 and 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. Letting P denotes a program that implements the function $f(x, y)$. Suppose P is executed with a test case $(1, 2)$, giving an output of 2. With respect to the MR $f(x, y) = f(y, x)$, P should next be executed with another test case $(2, 1)$. Then the output of second execution is compared with that of the first test case: does $P(1, 2) = P(2, 1)$? If the equality does not hold, then we consider that P at least has one fault.

B. Mutation Analysis

Mutation analysis [17]–[20] is widely used to evaluate the adequacy of test suite and the effectiveness of test techniques. Mutation operators are used to seed various faults into the SUT, and generate a set of variants (i.e. mutants). If a test case causes a mutant to behave differently to the SUT (for example, by giving different results for the same test case), then we say that this test case “kill” the mutant, and thus detects the injected fault. The mutation score (MS) is used to measure how thoroughly a test suite“kill” the mutants, defining as blow:

$$MS(p, ts) = \frac{N_k}{N_m - N_e}, \tag{1}$$

where p is the source program being mutated, ts is the test suite under evaluation, N_k is the number of mutants killed, N_m is the number of all mutants, and N_e is the number of equivalent mutants whose behaviors are always the same as that of p.

In this study, we employ mutation analysis to evaluate the efficiency and effectiveness of proposed AMT.

III. Motivation

Since MT was first published, a considerable number of studies have been conducted on various aspects of MT. To improve the effectiveness of MT, most of studies have paid their attention to identify the better MRs, which are more likely to be violated. For the effectiveness of MRs, several factors such as the difference between the source and follow-up test cases [6], [7] and the the detecting-faults capacity of MRs compared to existing test oracles [21], have been investigated.

Since the follow-up test cases are generated based on source test cases and MRs, in addition to the effective MRs, source test cases also have a impact on the effectiveness of MT. However, 57% of existing studies employed RT to select test cases, and 34% of existing studies used existing test suites [12]. In this study, we investigate the use of feedback information to select next test case, and its impacts on the effectiveness of MT.

This study combines RT and PT, with the goal of benefiting from the advantages of both. During the testing, we make using of feedback information to update the testing profile, and a new source test case is selected from a partition that was randomly selected according to updated testing profile, and then an MR is randomly selected from the set of MRs whose source test cases belong to selected partition.

IV. Adaptive Metamorphic Testing

In this section, we describe a framework for improving the effectiveness of MT, develop one algorithm for AMT, namely MRs-centric adaptive metamorphic testing (M-AMT), and present a prototype that partially automates M-AMT.

A. Framework

To show the idea of controlling the execution precess of MT, we propose a AMT framework, as illustrated in Figure 1. In the figure, interactions between MT components and the PT components are depicted in the framework. We next discuss the individual framework components.

1 *MRs Identification.* This component is responsible for identifying MRs according to the specification of SUT.
2 *Partition Construction.* Partition testing (PT) refers to a class of testing techniques that break the input domain into a number of partitions [22]. Various approaches and principles for achieving convenient and effective partitions have been discussed in the literature [22]–[25]. The input domain of the SUT can be partitioned based on the SUT specifications. 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 instance, a partition should be given higher priority if more faults have been detected within it in the previous testing history.
3 *MR Selection.* This component randomly select a MR from a set of MRs.
4 *Follow-up Test Cases Generation.* If the test is executed for the first time, the source test case is randomly selected from the input domain of the SUT, otherwise it is selected from a partition s_i that is randomly selected according to the testing profile. Next,
5 *Test Case Execution.* This component executes SUT with source and follow-up test cases, and outputs their results.
6 *Testing Profile Adjustment.* Upon completion of each test, its pass or fail status is determined by verifying the results of source and follow-up test cases against the corresponding MRs.
7 *Partition Selection.* This component selects a partition according to the testing profile.

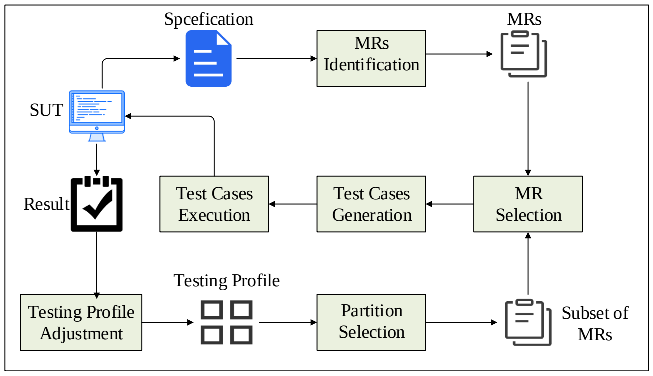

Fig. 1. DRT for web services framework

It has been pointed out that fault-detecting inputs tend to cluster into "continuous regions" [26], [27], that is, the test cases in some partition are more likely to detect faults than the test cases in other partitions. Following the above idea, AMT source test cases and MRs selection are in accordance with the testing profile that is dynamically updated during the test process. In our study, the strategies updating test profile, have been described in Section IV-B.

B. Two Strategies of Updating Testing Profile

1) Dynamic Random Testing: Dynamic Random Testing (DRT) [28] combines RT and PT, with the goal of benefitting from the advantages of both. Given a test suite TS classified into m partitions (denoted $s_1, s_2, \ldots, s_m$), and DRT is employed to update the values of the selection probabilities p_i. Assume that a source test case stc_i from s_i $(i = 1, 2, \ldots, m)$ is selected, and an metamorphic relation MR_h that is selected according to some strategies, is used to generate follow-up test case ftc_i from stc_i. If stc_i and ftc_i belong to s_i, and their results violate the MR_h, $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we then set

$$p'_j = \begin{cases} p_j - \dfrac{\epsilon}{m-1} & \text{if } p_j \geq \dfrac{\epsilon}{m-1} \\ 0 & \text{if } p_j < \dfrac{\epsilon}{m-1} \end{cases}, \tag{2}$$

where ϵ is a probability adjusting factor, and then

$$p'_i = 1 - \sum_{\substack{j=1 \\ j \neq i}}^{m} p'_j. \tag{3}$$

Alternatively, if their results hold the MR_h, we set

$$p'_i = \begin{cases} p_i - \epsilon & \text{if } p_i \geq \epsilon \\ 0 & \text{if } p_i < \epsilon \end{cases}, \tag{4}$$

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

$$p'_j = \begin{cases} p_j + \dfrac{\epsilon}{m-1} & \text{if } p_i \geq \epsilon \\ p_j + \dfrac{p'_i}{m-1} & \text{if } p_i < \epsilon \end{cases}. \tag{5}$$

The detailed DRT algorithm is given in Algorithm 1. In DRT, the source test case is selected from a partition that has been randomly selected according to the testing profile $\{\langle s_1, p_1 \rangle, \langle s_2, p_2 \rangle, \ldots, \langle s_m, p_m \rangle\}$, and an metamorphic relation is select according to some strategies (Lines 2 to 4 in Algorithm 1). During the testing, if the source and follow-up test cases are belonging to same partition, then the testing profile is updated by changing the p_i: If a fault is detected, then Formulas 2 and 3 are used to adjust the values of p_i (Line 8), otherwise Formulas 4 and 5 are used (Line 10). This process is repeated until a termination condition is satisfied (Line 1). Examples of termination conditions can be "when the testing resources are exhausted," "when a certain number of test cases have been executed," "when the first faults is detected," etc.

2) Metamorphic Dynamic Random Testing: When the source test case stc_i and follow-up test case ftc_i related to an metamorphic relation MR_h belong to same partition s_i, DRT is suitable to adjust the testing profile. However, during the testing process, there are some scenarios where the stc_i is not in the same partition as ftc_i. In such scenarios, DRT cannot be used to adjust the testing profile. To solve this problem, a new strategy, named Metamorphic Dynamic Random Testing (MDRT), is proposed.

Algorithm 1 DRT

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

C. M-AMT

D. Prototype

V. Empirical Study

A. Research Questions

B. Object Programs

VI. Related Work

A. Metamorphic Testing

B. Danymic Random Testing

Cai et al. introduced software cybernetics [13] to software testing, and proposed Adaptive Testing (AT) [29]–[31], which takes advantage of feedback information to control the execution process, has been shown that AT outperforms RT in terms of T-measure and increasing the number of faults, which means that AT has higher efficiency and effectiveness than RT.

Acknowledgment

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] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] K.-Y. Cai, "Optimal software testing and adaptive software testing in the context of software cybernetics," *Information and Software Technology*, vol. 44, no. 14, pp. 841–855, 2002.

[14] 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)*. IEEE, 1990, pp. 408–415.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

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

[21] 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.

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

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

[24] 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.

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

[26] 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.

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

1991.

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

[29] 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.

[30] 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.

[31] 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.