A New Method for Constructing Metamorphic Relations

Huai Liu, Xuan Liu, and Tsong Yueh Chen
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn 3122 VIC, Australia
{*hliu, xuanliu, tychen*}*@swin.edu.au*

Abstract

A fundamental problem for software testing is the oracle problem, which means that in many practical situations, it is extremely expensive, if not impossible, to verify the test result given any possible program input. Metamorphic testing is an approach to alleviating the oracle problem. The key part of metamorphic testing is a set of necessary properties of the software under test, namely metamorphic relations. Metamorphic relations not only help generate test cases, but also provide a mechanism to partially verify the test results without the need of oracle. In most previous studies, metamorphic relations were identified manually by testers in an ad hoc way. There is no systematic methodology that helps us identify metamorphic relations. In this paper, we propose a simple method, namely, the composition of metamorphic relations, for systematically constructing new metamorphic relations based on the already identified metamorphic relations. We conduct a case study and show that new metamorphic relations can be easily constructed by compositing some existing metamorphic relations. It is also observed that the new metamorphic relations are very likely to deliver a higher cost-effectiveness of metamorphic testing than the original metamorphic relations.

Key words*: software testing, metamorphic testing, metamorphic relation, composition of metamorphic relations.*

1. Introduction

Currently, software is playing a critical role in all areas of the whole world, and has a profound impact on this competitive society. However, failures universally exist in software products, and have caused massive disasters [17]. Software quality assurance has been acknowledged as a crucial activity in current software industry. Software testing, a major approach to guaranteeing the software quality, aims at revealing software failures as many as possible and as early as possible.

Software testing is normally accomplished by selecting some program inputs as *test cases*, executing the selected test cases, and verifying the test results [16]. Many testing techniques have been proposed to guide the selection of test cases with the purpose of improving the effectiveness in detecting software failures. Most of these test case selection techniques have implicitly assumed that there exists a systematic mechanism (termed as *oracle*) that helps testers verify the test result given any possible program input. Nevertheless, in many practical situations, there does not exist an oracle, or it is very expensive to apply the oracle [9]. Such a problem, termed as *oracle problem*, is a fundamental challenge of software testing. When the oracle problem exists, the applicability and effectiveness of many test case selection techniques will be greatly restrained, as it is very difficult, if not impossible, to check whether a failure has been detected or not.

Metamorphic testing [4] is a simple yet effective approach to the oracle problem. In metamorphic testing, some necessary properties of the software under test are identified from the software specification. These properties are presented in the form of some relations, namely *metamorphic relations*. Besides providing a test result verification mechanism when there is no oracle, metamorphic relations can also be used in the test case selection process. The metamorphic testing technique has been applied into the testing of various programs from different application domains, such as bioinformatics [5], machine learning [21], telecommunications [7], etc. In addition, metamorphic testing has also been used to alleviate the oracle problem for various software engineering techniques, such as fault-based testing [9], self-testing COTS components [3] symbolic execution [8], program slicing [22], etc.

It is obvious that metamorphic relations are the core part of the metamorphic testing, as they are not only used in test case generation, but also provide a mechanism for test result verification. Although it has been demonstrated that it is not very difficult to come up with some metamorphic relations for a program [13], metamorphic relations were often identified manually by testers in an ad hoc way [5, 7, 19, 21]. To date, no formal methodology has been proposed for systematically identifying metamorphic relations. Some researchers [6, 14] proposed some guidelines on how to select "good" metamorphic relations that intuitively have high

failure-detection capabilities out of already identified metamorphic relations. Testers were recommended to consider these selection guidelines when manually identifying metamorphic relations. However, these studies did not provide any systematic methodology for generating metamorphic relations.

As a matter of fact, the identification of metamorphic relations involves much human intelligence for analyzing specifications, finding necessary properties of the system under test, etc. Therefore, it is extremely difficult, if not impossible, to fully automate the identification process. In this paper, we are not going to propose a methodology for systematically identify metamorphic relations from scratch. Instead, we attempt to construct new metamorphic relations based on some already identified metamorphic relations. Our method is named as *composition of metamorphic relations*. Intuitively speaking, when several different metamorphic relations are composited into one single metamorphic relation, the resultant new metamorphic relation should embed all properties of the original metamorphic relations, and thus should not have a lower failure-detection capabilities than any individual original metamorphic relation. By compositing existing metamorphic relations, we may be able to use fewer metamorphic relations to reveal most failures that are detected when all original metamorphic relations are used, and hence improve the cost-effectiveness of metamorphic testing.

In this paper, we investigate the composition of metamorphic relations and the impact it may have on the cost-effectiveness of metamorphic testing. The rest of the paper is organized as follows. Section 2 introduces some preliminary information on metamorphic testing. Section 3 presents the basic concepts of the composition of metamorphic relations. In Section 4, we report a case study on our method, and investigate the cost-effectiveness of the metamorphic relations constructed by our method. The threat to validity of our study is discussed in Section 5. Section 6 compares our study with the related work. Section 7 concludes the paper.

2. Metamorphic Testing

Metamorphic testing is normally conducted according to the following steps.

1. Identify metamorphic relations from the specification of the software under test.

2. Generate the *source test case* using some traditional test case selection methods, and execute them.

3. Construct the *follow-up test case* from the source test cases based on metamorphic relations, and execute them.

4. Compare the results of source and follow-up test cases against metamorphic relations.

In Step 1 (metamorphic relation identification), domain-specific knowledge is required to fully understand the specification, so it is highly recommended that testers discuss with software users or developers to make sure that the identified metamorphic relations are correct and necessary properties for the software under test. Normally, a metamorphic relation is composed of two major parts. One part, namely *input relation*, refers to the relation between the inputs of source and follow-up test cases; while the other part, namely *output relation*, reflects the relation that the outputs of source and follow-up test cases are expected to hold. Theoretically, any test case selection technique can be used in the source test case generation (Step 2). Previous studies have used special case [20] and random testing [19] techniques to generate source test cases. In our case study, we use random testing [16] in Step 2, as it can generate a large number of test cases at low cost and with little human bias. In Step 3 (follow-up test case construction), the input relation of each metamorphic relation is used to construct the follow-up test case from the source test case. In this paper, we define a group of source and follow-up test cases as *a metamorphic test group*. It should be noted that the number of source or follow-up test cases in a metamorphic test group is not necessarily restricted to one. One or more source test cases may be associated with one or more follow-up test cases. In this paper, for ease of illustration, we assume that a metamorphic test group involves one source test case and one follow-up test case, unless otherwise specified. In Step 4 (test results verification), when the results of a metamorphic test group violate the output relation of a metamorphic relation, a failure is said to be detected.

The following simple example illustrates the basic process of metamorphic testing. Suppose that P is a program calculating the shortest path between two nodes in an undirected graph. Two metamorphic relations can be identified for P as follows.

- MR_A: If the starting and ending nodes are swapped, the length of the shortest path should remain unchanged.

- MR_B: If the graph is permutated, the length of the shortest path should remain unchanged.

The input relations of MR_A and MR_B are "the swapping of starting and ending nodes" and "the permutation of graph", respectively. MR_A and MR_B have the same output relation, that is, "the length of the shortest path should remain unchanged".

Suppose that the source test case is (G,a,b), where G is an undirected graph, a and b are the starting and ending nodes of the shortest path. Based on the first metamorphic relation (MR_A), we can generate a follow-up test case (G,b,a), where a and b are swapped. After executing both test cases, we check whether the relation $|P(G,a,b)| = |P(G,b,a)|$ (that is, the shortest path from a to b should have the same length as that from b to a) is satisfied or violated. If $|P(G,a,b)| \neq |P(G,b,a)|$, a failure is detected. Similarly, according to the second metamorphic relation (MR_B), the

follow-up test case can be constructed as (G', a', b'), where G' is the permutation of G, a' and b' are the permutated nodes of a and b, respectively. After the execution of the metamorphic test group ((G, b, a) and (G', a', b')), we check whether the relation $|P(G, a, b)| = |P(G', a', b')|$ is satisfied or violated. If violated, a failure is said to be detected.

As consistently shown in previous studies [14, 19, 20, 22], metamorphic testing has many advantages. First and foremost, metamorphic testing provides a test result verification mechanism when oracle does not exist. The test results are verified against metamorphic relations instead of oracle. In addition, most metamorphic relations are simple in concepts, so it is easy to automatically verify test results by using some simple scripts.

3. Composition of Metamorphic Relations

The following presents several definitions related to the composition of metamorphic relations. In these definitions, we suppose that given a source test case T, a follow-up test case $F_x(T)$ can be constructed according to a metamorphic relation MR_x.

Definition 1. *Let MR_x and MR_y be two metamorphic relations. MR_y is compositable to MR_x iff for any source test case T for MR_x, its corresponding follow-up test case $F_x(T)$ can always be used as the source test case for MR_y.*

Note that MR_y being compositable to MR_x does not necessarily imply MR_x being compositable to MR_y.

Definition 2. *Let MR_x and MR_y be two metamorphic relations such that MR_y is compositable to MR_x. A composite metamorphic relation MR_{xy} is said to be the composition of MR_x and MR_y iff for any source test case T for MR_{xy}, its corresponding follow-up test case $F_{xy}(T) = F_y(F_x(T))$.*

Note that even if both MR_{xy} and MR_{yx} exist, they are not necessarily equivalent to each other. In other words, the composition is sensitive to the order of the metamorphic relations to be composited.

Definition 3. *Let MR_1, MR_2, $\cdots$, and MR_n ($n \geq 2$) be n metamorphic relations, where MR_i is compositable to MR_{i-1} ($i = 2, \cdots, n$). A composite metamorphic relation $MR_{12\cdots n}$ is said to be the composition of MR_1, MR_2, $\cdots$, and MR_n iff for any source test case T for $MR_{12\cdots n}$, its corresponding follow-up test case $F_{12\cdots n}(T) = F_n(F_{n-1}(\cdots(F_1(T))\cdots))$.*

Let us consider the example given in Section 2. We can define a new composite metamorphic relation MR_{AB} through the composition of MR_A and MR_B.

- MR_{AB}. If the starting and ending nodes are swapped and then the graph is permutated, the length of the shortest path should remain unchanged.

For a source test case $T = (G, a, b)$, we can have a follow-up test case $F_{AB}(T) = (G', b', a')$ according to the new metamorphic relation MR_{AB}. After executing the source and follow-up test cases, we verify the test results by checking whether the relation $|P(G, a, b)| = |P(G', b', a')|$ is satisfied or violated.

Intuitively speaking, the new metamorphic relation defined by the composition of metamorphic relations will embed all properties associated with the original metamorphic relations. For example, the above MR_{AB} can help check the properties related to "permutation" as well as "node swapping". Intuitively, the more properties a metamorphic relation can reflect, the more failures it may be able to detect, provided that these properties do not cancel each other partially or completely.

In addition, the composition of metamorphic relations can reduce the number of test cases generated and executed in metamorphic testing. For example, if we use MR_A and MR_B for testing the program P, we generate and execute at least three test cases (one common source test case plus two follow-up test cases) in each run of testing. In iterative metamorphic testing [20], there are also at least three test cases (one initial source test case for MR_A, the follow-up test case of MR_A used as the source test case for MR_B, and the follow-up test case for MR_B, given that the order of usage is MR_A followed by MR_B) in each run of testing. However, if the composite metamorphic relation MR_{AB} is used instead of original MR_A and MR_B, only two test cases (one source and one final follow-up for MR_{AB} only) are executed for each run of testing.

4. A Case Study

4.1. Subject program and its metamorphic relations

Currently, many bioinformatics programs have the oracle problem as they often involve sophisticated computations and large-scale complex datasets. Some studies [2, 5] have been conducted for tackling the oracle problem in the testing of bioinformatics programs. Chen et al. [5] have used metamorphic testing technique to reveal a real-life bug in an open-source bioinformatics program. In this study, we select a bioinformatics program as the subject of the experiments. The subject program, namely `dnapars`, is a phylogenetic program, which is used to "infer evolutionary relationships among taxa using aligned sequences of characters, typically DNA or amino acids" [18]. The major input to `dnapars` is a $u \times v$ matrix, which presents the DNA sequences with u taxa and v nucleotides. `dnapars` constructs and outputs the *phylogenetic tree* based on the input DNA sequences. `dnapars` can also calculate the evolutionary steps for the constructed tree, termed as *total length*.

Sadi et al. [18] have identified seven metamorphic relations for testing `dnapars`. All these metamorphic relations will be used in our study, as listed in the following. In these metamorphic relations, the inputs for source and follow-up test cases are represented by X and X', respectively. T and

T' denote the output trees of X and X', respectively. The total lengths of T and T' are t and t', respectively.

- MR_1. X' is constructed by inserting a number of uninformative sites into X. Then, $T = T'$ and $t = t'$.
- MR_2. X' is constructed by changing every alphabet in every sequence of X according to the same transformation scheme. Then, $T = T'$ and $t = t'$.
- MR_3. X' is constructed by swapping two sites in X. Then, $T = T'$ and $t = t'$.
- MR_4. X' is constructed by removing some uninformative sites from X. Then, $T = T'$ and $t = t'$.
- MR_5. X' is constructed by inserting a number of hypervariable sites into X. Then, $T = T'$.
- MR_6. X' is constructed by concatenating each sequence with itself. Then, $T = T'$ and $2t = t'$.
- MR_7. X' is constructed by adding a duplicate sequence into X. Then, T and T' only differ in the subtree of the duplicate taxon having the same DNA sequence and $t = t'$.

4.2. Composite metamorphic relations

We name the composite metamorphic relations that are constructed by the composition of k metamorphic relations as *k-composite metamorphic relations*. We also name the original metamorphic relations used in the composition as *component metamorphic relations*. In our study, one component metamorphic relation will be used at most once in constructing a composite metamorphic relation. In other words, a composite metamorphic relation is composed of distinct component metamorphic relations. For the subject program, seven component metamorphic relations (MR_1, MR_2, $\cdots$, MR_7) have been identified, so we can have 2-, 3-, $\cdots$, 7-composite metamorphic relations. The following gives some examples of the 2-composite metamorphic relations.

- MR_{12}. X' is constructed by inserting a number of uninformative sites into X and then changing every alphabet in every sequence according to the same transformation scheme. Then, $T = T'$ and $t = t'$.
- MR_{63}. X' is constructed by concatenating each sequence with itself and then swapping two sites in X. Then, $T = T'$ and $2t = t'$.
- MR_{24}. X' is constructed by changing every alphabet in every sequence according to the same transformation scheme and removing some uninformative sites from X. Then, $T = T'$ and $t = t'$.
- MR_{75}. X' is constructed by adding a duplicate sequence into X and then inserting a number of hypervariable sites into X. Then, T and T' only differ in the subtree of the duplicate taxon having the same DNA sequence.

After an investigation, we found that MR_7 is not compositable to MR_5 (but MR_5 is compositable to MR_7). In other words, there is no composite metamorphic relation as $MR_{\ldots 5 \ldots 7}$ (but there are composite metamorphic relations as $MR_{\ldots 7 \ldots 5}$). Except for this case, all metamorphic relations are compositable to one another. Although there are only seven component metamorphic relations, a huge number of composite metamorphic relations can be constructed. The numbers of 2-, 3-, 4-, 5-, 6-, and 7-composite metamorphic relations are 41, 195, 720, 1,920, 4,680, 2,520, respectively. In total, 7,076 composite metamorphic relations can be constructed based on the existing seven component metamorphic relations for the subject program. Since the composition of metamorphic relations is very simple in concept, it can be easily automated. In other words, based on a few already identified metamorphic relations, it is very likely to automatically construct a huge amount of composite metamorphic relations at low cost.

4.3. Evaluation of cost-effectiveness

As discussed above, a composite metamorphic relation should embed all properties associated with its corresponding component metamorphic relations. Through the composition of metamorphic relations, we may be able to use the composite metamorphic relation to detect most failures that are revealed by its component relations. As a result, the cost-effectiveness of metamorphic testing may be enhanced. We conducted a series of experiments to demonstrate such enhancement on the cost-effectiveness.

Mutation analysis technique [10] has been used in previous study [18] for evaluating the failure-detection effectiveness of metamorphic testing on `dnapars`. Faults were seeded into the original version of `dnapars` to generate some faulty versions, namely *mutants*. In this study, we will use 11 mutants of `dnapars`, as summarized in Table 1.

4.3.1. Failure-detection capabilities of metamorphic relations.

We generated 500 test inputs using the random testing techniques. These test inputs were used as the source test cases, based on which, follow-up test cases are constructed according to metamorphic relations. Therefore, totally 500 metamorphic test groups (each of which is composed of one source test case and its corresponding follow-up test case, as defined in Section 2) were executed for each metamorphic relation (component or composite). Table 2 reports the number of failures detected by each component metamorphic relation on each mutant. For example, the "94" in the rightmost bottom cell of the table means that out of all 500 metamorphic test group for MR_7, 94 groups can detect failures in the mutant $\mu 11$.

It should be noted that our testing results of the component metamorphic relations are slightly different from those in previous study [18]. The difference is due to the use of different random test inputs and different implementations

Table 1. Mutants of the subject program

mutant	line#	original statement	faulty statement
$\mu 1$	738	$ns = 1 << G$	$ns = 1 << C$
$\mu 2$	2807	if($i == j$)	if($i != j$)
$\mu 3$	565	if(ally[alias[i-1]-1] != alias[i-1])	if(ally[alias[i-1]-1] >= alias[i-1])
$\mu 4$	1115	for($i = a$; $i < b$; i++)	for($i = a$; $i <= b$; i++)
$\mu 5$	567	$j = i + 1$;	$j = i - 1$;
$\mu 6$	992	for(i = (long)A; $i <= $ (long)O; i++)	for(i = (long)A; $i >$ (long)O; i++)
$\mu 7$	575	itemp = alias[i-1];	itemp = alias[i+1];
$\mu 8$	1137	for(j = (long)A; $j <= $ (long)O; j++)	for(j = (long)A; $j >= $ (long)O; j++)
$\mu 9$	1077	else p -> numsteps[i] += weight[i];	else p -> numsteps[i] -= weight[i];
$\mu 10$	1182	for(j = (long)A; $j <= $ (long)O; j++)	for(j = (long)A; $j >$ (long)O; j++)
$\mu 11$	566	if($j <= i$)	if($j > i$)

Note: all failures were seeded into the file `seq.c`.

Table 2. Failure-detection capabilities of component metamorphic relations in terms of the number of violations out of 500 random metamorphic test groups

	$\mu 1$	$\mu 2$	$\mu 3$	$\mu 4$	$\mu 5$	$\mu 6$	$\mu 7$	$\mu 8$	$\mu 9$	$\mu 10$	$\mu 11$
MR_1	0	1	472	3	13	500	0	0	0	500	0
MR_2	494	0	496	0	414	0	67	0	0	0	75
MR_3	0	0	268	0	0	0	0	0	0	0	0
MR_4	203	204	465	204	106	500	203	204	201	500	184
MR_5	0	0	79	116	1	0	0	0	8	0	0
MR_6	0	0	465	2	374	0	66	0	0	0	79
MR_7	15	1	69	83	254	16	10	300	128	10	94

of metamorphic relations (such as different transformation schemes for alphabet in MR_2).

We have investigated the failure-detection capabilities of all composite metamorphic relations. Due to page limit, we only report the results of 2-composite metamorphic relations in Table 3. Each cell of Table 3 denotes the number of failures detected by 500 metamorphic test groups for a pair of 2-composite metamorphic relation and mutant.

The experimental results are analyzed based on the following two research questions.

4.3.2. One composite metamorphic relation vs. several component metamorphic relations.

Research Question 1. *Suppose that k component metamorphic relation are composited into one composite metamorphic relation. Will the composite metamorphic relation have higher or at least similar failure-detection capability than each of its component metamorphic relations?*

It is obvious that the new composite metamorphic relation will involve fewer test executions than using all k component metamorphic relations in each run of testing. Therefore, even if the new metamorphic relation detects a similar number of failures as the component metamorphic relations, the cost-effectiveness is improved.

Binomial tests [15] were conducted to compare the effectiveness of the 2-composite metamorphic relations and the component metamorphic relations. The null hypothesis (H_0) is that for a 2-composite metamorphic relations MR_{pq}, $m_{pq} < \max\{m_p, m_q\}$, where m_{pq}, m_p, and m_q are the numbers of failures detected by MR_{pq}, MR_p, and MR_q, respectively. The significance level is set as 0.05. The results of the comparisons based on mutants are given in Table 4. In Table 4, N_{MR} refers to the total number of 2-composite metamorphic relations used to test a mutant, and N_s refers to the number of 2-composite metamorphic relations MR_{pq} whose corresponding $m_{pq} < \max\{m_p, m_q\}$.

In general, it is statistically significant that 2-composite metamorphic relations have at least similar failure-detection capabilities as the component metamorphic relations. Even for the exceptional cases (that is, for mutants $\mu 3$ and $\mu 5$), the difference between the performances of composite metamorphic relations and the best component metamorphic relation is not very large, as shown in Table 3.

In order to further examine the impact of individual component metamorphic relations on the effectiveness of composite metamorphic relations, we also conducted the binomial test based on each component metamorphic relation. The results are given in Table 5. In Table 5, for each column corresponding to MR_i, N_c refers to the total number of 2-composite metamorphic relations (MR_{ij} or MR_{ji}) that are composed of MR_i and another metamorphic relation MR_j, N_μ refers to the number of mutants (in this study,

Table 3. Failure-detection capabilities of all 2-composite metamorphic relations in terms of the number of violations out of 500 random metamorphic test groups

	$\mu 1$	$\mu 2$	$\mu 3$	$\mu 4$	$\mu 5$	$\mu 6$	$\mu 7$	$\mu 8$	$\mu 9$	$\mu 10$	$\mu 11$
MR_{12}	494	1	496	6	414	500	67	0	0	500	75
MR_{13}	0	1	468	5	13	500	0	0	0	500	0
MR_{14}	203	204	484	206	94	500	203	204	204	500	184
MR_{15}	0	1	74	120	0	0	0	0	8	0	0
MR_{16}	0	1	479	4	374	500	66	0	0	500	79
MR_{17}	15	2	98	86	255	16	10	300	128	10	94
MR_{21}	494	1	498	4	414	500	67	0	0	500	75
MR_{23}	494	0	497	1	414	0	67	0	0	0	75
MR_{24}	495	204	492	204	425	500	243	204	204	500	248
MR_{25}	31	0	96	117	72	0	0	0	8	0	60
MR_{26}	494	0	494	2	430	0	71	0	0	0	79
MR_{27}	45	1	130	83	289	16	10	300	128	10	123
MR_{31}	0	1	465	4	13	500	0	0	0	500	0
MR_{32}	494	0	497	1	414	0	67	0	0	0	75
MR_{34}	203	204	479	204	106	500	203	204	204	500	184
MR_{35}	0	0	73	117	1	0	0	0	8	0	0
MR_{36}	1	1	482	3	375	1	67	1	1	1	80
MR_{37}	15	1	88	83	254	16	10	300	128	10	94
MR_{41}	193	194	469	195	92	500	194	194	194	500	173
MR_{42}	495	204	492	203	425	500	242	202	202	500	246
MR_{43}	203	204	478	205	101	500	203	204	204	500	184
MR_{45}	2	0	73	128	5	0	0	0	10	0	3
MR_{46}	203	204	477	206	383	500	245	204	204	500	261
MR_{47}	17	1	100	119	257	16	10	301	129	10	95
MR_{51}	0	1	77	120	0	0	0	0	8	0	0
MR_{52}	31	0	98	114	72	0	0	0	8	0	60
MR_{53}	0	0	77	115	1	0	0	0	8	0	0
MR_{54}	2	0	73	128	5	0	0	0	10	0	3
MR_{56}	1	1	79	117	118	1	1	1	9	1	66
MR_{61}	0	1	482	3	374	500	66	0	0	500	79
MR_{62}	494	0	494	3	430	0	71	0	0	0	79
MR_{63}	0	0	483	2	374	0	66	0	0	0	79
MR_{64}	207	207	484	207	383	500	248	207	207	500	264
MR_{65}	0	0	84	117	117	0	0	0	8	0	65
MR_{67}	15	1	131	83	371	16	10	300	128	10	94
MR_{71}	15	2	98	85	255	16	10	300	128	10	94
MR_{72}	45	1	130	83	289	16	10	300	128	10	123
MR_{73}	15	1	88	83	254	16	10	300	128	10	94
MR_{74}	17	1	100	119	257	16	10	301	129	10	95
MR_{75}	5	0	104	83	255	15	10	300	128	10	94
MR_{76}	15	1	131	83	371	16	10	300	128	10	94

Table 4. Binomial test on the failure-detection capabilities of 2-composite metamorphic relations based on mutants

	N_s	N_{MR}	p-value	decision
$\mu 1$	9	41	0.0002	reject H_0
$\mu 2$	6	41	2.44×10^{-6}	reject H_0
$\mu 3$	28	41	0.9942	accept H_0
$\mu 4$	9	41	0.0002	reject H_0
$\mu 5$	16	41	0.1055	accept H_0
$\mu 6$	9	41	0.0002	reject H_0
$\mu 7$	14	41	0.0298	reject H_0
$\mu 8$	5	41	3.92×10^{-7}	reject H_0
$\mu 9$	5	41	3.92×10^{-7}	reject H_0
$\mu 10$	8	41	5.61×10^{-5}	reject H_0
$\mu 11$	11	41	0.0022	reject H_0
Total	120	451	3.36×10^{-24}	reject H_0

Table 5. Binomial test on the failure-detection capabilities of 2-composite metamorphic relations based on each component metamorphic relation

	N_s	$N_c \times N_\mu$	p-value	decision
MR_1	26	132	5.84×10^{-13}	reject H_0
MR_2	26	132	5.84×10^{-13}	reject H_0
MR_3	12	132	7.11×10^{-24}	reject H_0
MR_4	55	132	0.0336	reject H_0
MR_5	56	121	0.2336	accept H_0
MR_6	20	132	5.18×10^{-17}	reject H_0
MR_7	45	121	0.0031	reject H_0

$N_\mu = 11$), and N_s refers to the number of 2-composite metamorphic relations (MR_{ij} or MR_{ji}) whose corresponding m_{ij} or $m_{ji} < \max\{m_i, m_j\}$.

Table 5 shows that except MR_5, all other metamorphic relations have positive impacts on the failure-detection capabilities of the composite metamorphic relations. We further investigated why MR_5 has a negative impact on the composite metamorphic relations under some situations. Different from all other metamorphic relations, MR_5 only involves the phylogenetic trees ($T = T'$). All other metamorphic relations additionally involve total lengths ($t = t'$ for MR_1 to MR_4 and MR_7; $2t = t'$ for MR_6). In other words, MR_5 has a "looser" output relation, which may also explain why MR_5 has a lower failure-detection capability than other component metamorphic relations. The following MR_{15} gives an example of the composition of MR_5 with another metamorphic relation.

- MR_{15}. X' is constructed by inserting a number of uninformative sites as well as a number of hypervariable sites into X. Then, $T = T'$.

In order for the resultant composite metamorphic relations to be valid, when MR_5 is composited with other metamorphic relations, the output relation between total lengths in other component metamorphic relations could not be used. In the above example, as MR_5 does not have a certain relation between t and t', the relation $t = t'$ from MR_1 should not appear in MR_{15}. Such an observation implies that when compositing metamorphic relations, their output relations should have similar "tightness". Once there is one metamorphic relation that involves a "loose" output relation, the failure-detection capabilities of the composite metamorphic relations may be deteriorated.

Due to the page limit, we will not present the detailed results of 3-, 4-, $\cdots$, 7-composite metamorphic relations. In brief, as the number of component metamorphic relations becomes higher, the failure-detection capabilities of composite metamorphic relations become worse. For example, only 52.7% of all 3-composite metamorphic relations MR_{uvw} can detect m_{uvw} failures such that $m_{uvw} \geq \max m_u, m_v, m_w$, where m_u, m_v, and m_w are the numbers of failures revealed by MR_u, MR_v, and MR_w, which composed MR_{uvw}. In the case of 7-composition, all 7-composite metamorphic relations cannot outperform the best component metamorphic relation. As discussed above, MR_5 has a negative impact on the failure-detection capabilities of composite metamtorphic relations. As the number of component metamorphic relations becomes higher, it is more likely to use MR_5 in the construction of the composite metamorphic relations (probabilities involving MR_5 in the composition are 26.8%, 38.5%, 50.0%, 62.5%, 84.6%, and 100% for 2-, 3-, 4-, 5, 6-, and 7-composite metamorphic relations, respectively). Therefore, it is not surprising that the failure-detection capabilities of composite metamorphic relations deteriorate with the increase of the number of used component metamorphic relations.

We also analyzed the composite metamorphic relations that do not involve MR_5. These metamorphic relations consistently show higher failure-detection capabilities than each of the component metamorphic relations. Such an observation, from another perspective, confirms our conjecture on MR_5, that is, MR_5 has a negative impact on the failure-detection capabilities of composite metamorphic relations.

Based on the above results, we can answer our first research question as follows.

Answer to Research Question 1. *The composition of k ($k > 1$) metamorphic relations can produce a composite metamorphic relation that normally has higher (or at least similar) failure-detection capability than each component metamorphic relation, as long as the output relations of the k component metamorphic relations have similar "tightness". Since the composite metamorphic relation always involves fewer test executions, it normally can deliver a higher cost-effectiveness in testing than aggregatively using the k component metamorphic relations whose output relations have similar "tightness". On the other hand, if one component metamorphic relation involves a "loose" output relation, it is not guaranteed that the composite metamorphic relation will still have a high failure-detection capability.*

4.4. A set of composite metamorphic relations vs. a set of component metamorphic relations

Research Question 2. *Suppose that there are totally h metamorphic relations (MR_1, MR_2, $\cdots$, MR_h) that are originally identified for one program. l composite metamorphic relations (MR'_1, MR'_2, $\cdots$, MR'_l) are constructed as follows. Each MR'_i ($i = 1,2,\cdots,l$) is the composition of $N_i > 1$ component metamorphic relations. Each component metamorphic relation MR_j ($j = 1,2,\cdots,h$) is used **once and only once** to construct the set of l composite metamorphic relations. Will the new l composite metamorphic relations have higher cost-effectiveness than the h component metamorphic relations?*

In this study, we name the set of l composite metamorphic relations as the *complete set of composite metamorphic relations*, because all h component metamorphic relations have been considered in the set. For the subject program `dnapars`, we have $h = 7$ component metamorphic relations. A complete set of composite metamorphic relations can be constructed according to one of the following four schemes.

- $l = 3$: two 2-composite metamorphic relations and one 3-composite metamorphic relation.
- $l = 2$: one 3-composite metamorphic relation and one 4-composite metamorphic relation.
- $l = 2$: one 2-composite metamorphic relation and one 5-composite metamorphic relation.
- $l = 1$: only one 7-composite metamorphic relation.

We constructed 100 complete sets of composite metamorphic relations in the following way.

- For one and only one set, $l = 1$. We found that all 7-composite metamorphic relations have very similar failure-detection capabilities. However, the failure-detection capabilities of 2-, 3-, 4-, and 5-composite metamorphic relations are significantly different to one another. The experimental scenarios should be as diversified as possible such that the conclusion could be made in a general sense. Therefore, we only used one complete set with $l = 1$.
- We randomly selected the first three schemes for constructing the other 99 complete sets.

In our study, 500 metamorphic test groups were generated and executed for each metamorphic relation. For the 7 component metamorphic relations, totally $7 \times 500 = 3,500$ metamorphic test groups were used in testing 11 mutants. Based on the data shown in Table 2, we can calculate that the *average failure-detection ratio* of these 3,500 metamorphic test groups on the 11 mutants is 0.2194. Here, the failure-detection ratio refers to the ratio between the number of detected failures and the number of metamorphic test groups.

We used the $l \times 500$ metamorphic test groups generated based on each of 100 complete sets of composite metamorphic relations to test 11 mutants. The average failure-detection ratio of each set was compared to that of the 7 component metamorphic relations. Binomial test was conducted on such comparisons. The null hypothesis (H_0) is that the average failure-detection ratio of a complete set of composite metamorphic relations is not larger than that of the component metamorphic relations. The significance level is 0.05. We observed that among all 100 complete sets of composite metamorphic relations, 68 sets have the average failure-detection ratios larger than 0.2194. The p-value of the binomial test is 0.0002. Therefore, the null hypothesis is rejected. In other words, it is statistically significant that l composite metamorphic relations have higher cost-effectiveness than h component metamorphic relations. Based on the above results, we can answer our second research question as follows.

Answer to Research Question 2. *Suppose that h metamorphic relations are originally identified for the program under test. A complete set of $l < h$ composite metamorphic relations can be constructed through the compositions of the h component metamorphic relations, where each component metamorphic relation is used once and only once for constructing a composite metamorphic relation. The cost-effectiveness of l composite metamorphic relations is normally higher than that of h component metamorphic relations.*

5. Threats to Validity

The threats to validity of our study are discussed as follows.

The threat to internal validity is mainly related to the implementation of metamorphic testing based on the metamorphic relations. The programming work was fulfilled independently by one author of the paper. All the source code was carefully reviewed by another author. The involved programs were also cross-checked with those in a previous study [18]. We are confident that the metamorphic testing has been correctly implemented in our experiments.

The major threat to external validity is about the subject program and its associated metamorphic relations. In this pilot study, we selected the subject program that has been used in a previous study [18]. Though the subject is a typical program with oracle problem, we cannot say that our method will work for any other type of programs. The original/component metamorphic relations used in our study are extracted from a previous study [18], but the identification of these metamorphic relations were manually conducted in an ad hoc way, and thus is somewhat subjective.

The main concern about the threat to construct validity is the measurement. In our study, we measured the failure-detection capabilities of metamorphic relations on some mutants of the subject program. The mutation analysis technique has been acknowledged as the major method for fairly

evaluating the effectiveness of a testing method [1]. In addition, we evaluated the cost-effectiveness of metamorphic testing using the ratio between the number of revealed failures and the number of executed test cases. Such a metric is straightforward for showing the testing cost-effectiveness.

There is little threat to conclusion validity to our study, because a large number of test cases have been used for the implementation of each metamorphic relation. Our experiments resulted in a huge amount of data, which could help us reach a statistically reliable conclusion. A formal statistical technique has also been conducted to verify the statistical significance of the experimental results.

6. Related Work

Some studies have been conducted on how to select "good" metamorphic relations. Chen et al. [6] compared various metamorphic relations identified for programs of shortest path and critical path, and attempted to distinguish metamorphic relations that are effective in detecting software failures. Their study showed that theoretically understanding the application domain is not sufficient for identifying good metamorphic relations. It was further suggested that testers should understand the algorithm structure before identifying metamorphic relations. It was also observed that there is a common characteristic of good metamorphic relations, that is, the execution behaviors of source and follow-up test cases should be quite different from each other. Mayer and Guderlei [14] examined some metamorphic relations for several Java programs of determinant computation. They observed that metamorphic relations that have rich semantic properties normally have high failure-detection capabilities. It was also suggested that testers should avoid the metamorphic relations that involve similar computations as the implemented algorithm. All the studies on selecting "good" metamorphic relations only provided some guidelines that help testers identify metamorphic relations that intuitively have high failure-detection capabilities. No systematic methodology has been proposed for constructing metamorphic relations, and testers still need to identify metamorphic relations in an ad hoc way. In this paper, we proposed a simple yet effective method of defining new metamorphic relations through the composition of existing metamorphic relations. To our best knowledge, this study is the first attempt to systematically construct metamorphic relations (although not from scratch).

Gotlieb and Botella [12], in their automated metamorphic testing framework, proposed some "general" forms of metamorphic relations. Based on these general forms, some concrete metamorphic relations could be constructed for a given program. For example, for the programs `GetMid` (which selects the median of three integers) and `TriType` (which outputs the type of a triangle given three integers as the side lengths of the triangle), one general form of metamorphic relation was proposed based on the permutation of the inputs. Both programs can have the metamorphic relation that the permutation of inputs should not change the output. However, there is neither a systematic method for identifying the "general" forms of metamorphic relations, nor a systematic approach for deciding whether and how a general form can be used to construct a metamorphic relation for a concrete program. In other words, this framework still requires lots of human effort in the construction of metamorphic relations. Although our method also requires human intelligence in identifying original metamorphic relations, the composition of metamorphic relations is very straightforward and thus can be automatically conducted at low cost.

Iterative metamorphic testing [11, 20] and our method have a commonality that both make use of multiple component metamorphic relations aggregatively. However, iterative metamorphic testing does not involve the concept of new composite metamorphic relations. Basically, they generate test cases relation by relation successively. Once a follow-up test case is generated based on one metamorphic relation, this test case will be used as the source test case for another metamorphic relation. In our work, multiple component metamorphic relations are first composited into a new composite metamorphic relation, which is then used to generate test cases as well as verify test results. Our method involves fewer test executions than iterative metamorphic testing. In addition, there exist some situations where the follow-up test cases for one metamorphic relation cannot be used as the source test cases for another metamorphic relation. Such cases were not fully discussed in the studies of iterative metamorphic testing [11, 20]. In this paper, we presented some definitions for the composition of metamorphic relations. These definitions can also be used in iterative metamorphic testing for guiding the aggregative usages of metamorphic relations.

7. Conclusion and Future Work

Metamorphic testing is an approach to the oracle problem in software testing. Metamorphic testing makes use of metamorphic relations for verifying test results as well as generating test cases. To date, there is no systematic method for identifying metamorphic relations, and testers often identified metamorphic relations in an ad hoc way. In this paper, we proposed a simple yet effective method for the construction of metamorphic relations, namely the composition of metamorphic relations. Although our method still needs some metamorphic relations as its inputs, it can systematically construct much more new metamorphic relations from the given metamorphic relations. An experimental study also showed that the composition of metamorphic relations can improve the cost-effectiveness of metamorphic testing.

As a pilot study, we only investigated the composition of metamorphic relations on one bioinformatics program. It is important to further study this topic on various subject programs to avoid any potential threat to validity. For ease of

illustration, this pilot study only involved metamorphic relations of simple forms. There exist metamorphic relations of more complicated forms in the literature. Some metamorphic relations involve multiple source test cases and/or multiple follow-up test cases. For some metamorphic relations, the outputs of the source test cases should also be considered in the construction of follow-up test cases. It will be much more challenging to composite such metamorphic relations. Another future work on the composition of metamorphic relations is to investigate the basic reason why some metamorphic relations have negative impacts on the failure-detection capabilities of the composite metamorphic relations. It is necessary to formally define the notion of "tightness" of the output relation in a metamorphic relation. It is also crucial to propose some principles or guidelines for judging whether a metamorphic relation has a "loose" output relation and thus should be isolated from the composition such that the constructed composite metamorphic relations will definitely bring higher cost-effectiveness than original metamorphic relations. It is also worthwhile to integrate the research on the selection of "good" metamorphic relations and our study on the composition of metamorphic relations. On one hand, the composition of "good" metamorphic relations may further improve the cost-effectiveness of metamorphic testing. On the other hand, the criteria for evaluating "good" metamorphic relations can also be used to guide the composition of metamorphic relations.

Acknowledgment

This research project is supported by an Australian Research Council Discovery Grant.

References

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proceedings of the 27th International Conference on Software Engineering (ICSE05)*, pages 402–411, 2005.

[2] F. T. Bergmann and H. M. Sauro. Comparing simulation results of SBML capable simulators. *Bioinformatics*, 24(17):1963–1965, 2008.

[3] S. Beydeda. Self-metamorphic-testing components. In *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC06)*, pages 265–272, 2006.

[4] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: A new approach for generating next test cases. Technical Report HKUSTCS98-01, Department of Computer Science, Hong Kong University of Science and Technology, 1998.

[5] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, 10:24, 2009.

[6] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou. Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC04)*, pages 569–583, 2004.

[7] T. Y. Chen, F.-C. Kuo, H. Liu, and S. Wang. Conformance testing of network simulators based on metamorphic testing technique. In *Proceedings of the 29th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE09)*, pages 243–248, 2009.

[8] T. Y. Chen, T. H. Tse, and Z. Zhou. Semi-proving: An integrated method for program proving, testing, and debugging. *IEEE Transactions on Software Engineering*, 37(1):109–125, 2011.

[9] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. Fault-based testing without the need of oracles. *Information and Software Technology*, 45:1–9, 2003.

[10] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):31–41, 1978.

[11] G. Dong, C. Nie, B. Xu, and L. Wang. An effective iterative metamorphic testing algorithm based on program path analysis. In *Proceedings of the 7th International Conference on Quality Software (QSIC07)*, pages 292–297, 2007.

[12] A. Gotlieb and B. Botella. Automated metamorphic testing. In *Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC03)*, pages 34–40, 2003.

[13] P. Hu, Z. Zhang, W. K. Chan, and T. H. Tse. An empirical comparison between direct and indirect test result checking approaches. In *Proceedings of the 3rd International Workshop on Software Quality Assurance (SOQUA06)*, pages 6–13, 2006.

[14] 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 (COMPSAC06)*, pages 475–484, 2006.

[15] H. Motulsky. *Intuitive Biostatistics*. Oxford University Press, 1995.

[16] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, second edition, 2004.

[17] National Institute of Standards and Technology. The economic impacts of inadequate infrastructure for software testing. `http://www.nist.gov/director/prog-ofc/report02-3.pdf`, Gaithersburg, Maryland, USA, 2002.

[18] M. S. Sadi, F.-C. Kuo, J. W. K. Ho, M. A. Charleston, and T. Y. Chen. Verification of phylogenetic inference programs using metamorphic testing. *Journal of Bioinformatics and Computational Biology*, 9(6):729–747, 2011.

[19] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen. Metamorphic testing for web services: Framework and a case study. In *Proceedings of the 9th International Conference on Web Services (ICWS10)*, pages 283–290, 2011.

[20] P. Wu. Iterative metamorphic testing. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC05)*, pages 19–24, 2005.

[21] X. Xie, J. W. K. Ho, C. Murphy, G. E. Kaiser, B. Xu, and T. Y. Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544–558, 2011.

[22] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu. Spectrum-based fault localization: Testing oracles are no longer mandatory. In *Proceedings of the 11th International Conference On Quality Software (QSIC11)*, pages 1–10, 2011.