PROGRAM SYSTEMS: THEORY AND APPLICATIONS ISSN 2079-3316
Review Article OPTIMIZATION METHODS AND CONTROL THEORY

UDC 004.415.53
10.25209/2079-3316-2024-15-2-37-86

A systematic review of methods for deriving metamorphic relations

Sofia Fedorovna **Iakusheva**[1], Anton Sergeevich **Khritankov**[2]

[1, 2] Moscow Institute of Physics and Technology, Moscow, Russia
[2] Higher School of Economics, Moscow, Russia

[1] ✉ *yakusheva.sf@phystech.edu*

Abstract. Metamorphic testing is one of the most effective methods of testing programs with the test oracle problem. This problem declares that it is impossible to know whether the test answer is correct for one reason or another. Metamorphic testing uses metamorphic relations to check the program correctness. Metamorphic relation is a function of several test inputs and corresponding outputs of the program. Developing metamorphic relations can be a non-trivial task.

This systematic review is dedicated to identifying general derivation techniques for metamorphic relation as well as techniques pertinent to particular domains. As a result, we propose a classification of techniques into six main types and compile a comparative table of input data transformations for testing tasks in different domains. Findings of this review will help researchers to apply metamorphic testing in practice.

Key words and phrases: metamorphic testing, metamorphic relation, software testing, test oracle problem

2020 *Mathematics Subject Classification*: 97P99; 97U99

For citation: Sofia F. Iakusheva, Anton S. Khritankov. *A systematic review of methods for deriving metamorphic relations*. Program Systems: Theory and Applications, 2024, **15**:2(61), pp. 37–86. https://psta.psiras.ru/read/psta2024_2_37-86.pdf

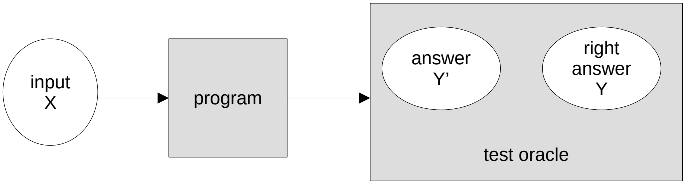

Figure 1. An example of checking the test. The role of the oracle is to compare the answer with the right one.

Introduction

Software quality control is a crucial part of the development process. At this stage, one verifies that the software meets all the set requirements, which means it will behave in an expected and previously known manner. Software deviations from the stated requirements regularly become the causes of aircraft crashes, self-driving car accidents, failures of space missions, and data security breaches [1]. Non-compliance with the requirements is a good reason for modifying the developed system or decommissioning the one already in use.

Software testing is one of the methods for quality monitoring during the software development process. Software testing identifies incorrect program behavior, violations of functional and non-functional requirements, and usage scenarios not provided in the documentation. A test oracle [2] is a function that determines the correctness of a program's response. The simplest test oracle compares the program response with a previously known answer (see Figure 1).

A more complex example could be an oracle that checks the correctness of the Hamiltonian path in a graph: it is enough to check the inclusion of all vertices of the graph in the path with the connecting edges; this test does not require an answer known a priori. In poorly formalized cases, if the requirements are inaccurate or a high level of expertise is necessary, then a user, an expert, or another program can be taken as a test oracle.

When testing knowledge-intensive, non-deterministic, distributed software systems, the so-called test oracle problem became a significant obstacle. The test oracle problem [3] is the difficulty in devising test oracles, especially automatic ones. The problem is relevant in situations that require exhaustive search to find an exact solution. Such situations appear in bioinformatics and combinatorics, in machine learning tasks due to the costly process of test data collection and labeling or tasks of creating artistic objects due to the need for human evaluation of conformance to the poorly stated requirements. However, there are various methods to approach this problem. Property-based testing, for instance, checks the fulfillment of a given relationship between the inputs and outputs of a program. Metamorphic testing is a variation of this method.

1. Metamorphic testing

1.1. Definitions and examples

Metamorphic testing [4] is used in many areas and has several development directions [5, 6]. The idea of the method is to check a metamorphic relation between stimuli and responses instead of the correctness of each specific program response. A metamorphic relation is a function calculated on several program inputs and outputs:

$$R(x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n)) \longrightarrow \{0, 1\}, \quad (1)$$

where $n \geqslant 2$ denotes the number of runs in the test case, x_i — input data for the i-th run in the test case, $f(x_i)$ — the i-th output. The metamorphic testing technique consists of the following steps. First, we set the procedure for obtaining input data which can involve using a data generator. Next, we run the program on generated inputs in the test case and check the if the relation holds. If there are input data for which R is zero, then there are errors or non-compliances with the requirements. Thus, instead of directly checking the answers, the method uses the so-called derived test oracle [6]. The construction of such a relation for a number of problems can naturally follow from the properties of the problem solved by the program and can be simpler for a program developer or researcher.

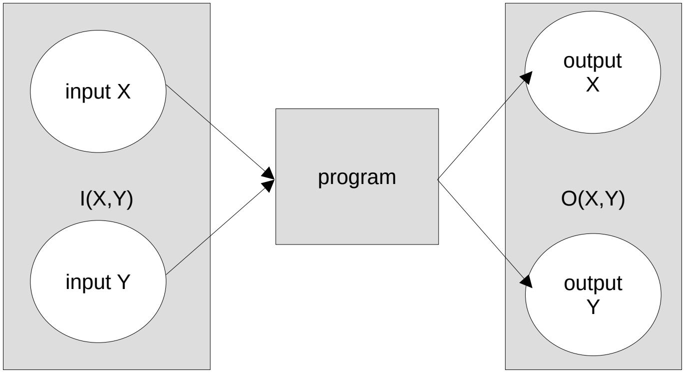

FIGURE 2. A simple example of metamorphic testing. The metamorphic relation is a function $R(x, y, X, Y) = I(x, y) \cap O(X, Y)$. Presence of dependence $I(x, y)$ between test inputs should cause the presence of the dependence $O(X, Y)$ between outputs.

A common metamorphic relation has the following statement. If two elements of the input data sequence satisfy some relation I, then the two corresponding program responses must satisfy the relation O (see Figure 2):

$$(2)\quad R(x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n)) = \\ = I(x_1, x_2, \ldots, x_n) \cap O(f(x_1), f(x_2), \ldots, f(x_n)),$$

where I is the transformation of the input data, O is the expected relationship between the outputs, $n \geqslant 2$ is the total number of test runs in the test case, x_i is the input data for the i-th run in the test case, and $f(x_i)$ is the i-th program output (see Figure 2). We will consider relations under the conditions that test runs are independent and that the initial state of the program in each run is known.

As an example, let us describe several metamorphic relations for checking the correct execution of queries to a relational database containing a table $T = a|b|c|...$, where a, b, c are the table columns.

- Let A and B be two search conditions. Then the answer of a query with condition $A \cap B$ is a subset of the answer of a query with condition A (similar to B).
- Let A be a search condition. The answers to the query with condition A and the query with condition $\neg A$ do not intersect.

- Let A be a search condition. The number of answers to a query with condition A when sorting in ascending order of column a and when sorting in descending order of column a is the same.

1.2. Terminology

Metamorphic testing is rarely discussed in Russian scientific literature, so it does not have a common terminology in Russian. Some researchers use the term *метаморфное тестирование* which is the literal translation of the English term *metamorphic testing*.

Authors use terms *тестирование инвариантами* (*invariant testing*) for metamorphic testing and *тестовый инвариант* (*test invariant*) for metamorphic relation in Russian text. Such terms are much easier to understand and to remember due to the simple close relation with the definition. In mathematical literature, invariant means a value or a property that is constant in the situation under observation. Test invariant usually means a function that preserves its value if the parameters change as expected. In physics, an invariant has a similar meaning, for example, in the phrase *time translation invariance*. The formula 1.1 defines such a function, so by meaning *metamorphic relation* we assume a *test invariant*. Moreover, a *test invariant* can be derived as a corollary from a mathematical model of the problem solved by the program. So the used terms *тестовый инвариант* and *тестирование инвариантами* are suitable translations from English.

2. Research goals and methods

At the current stage of the development of metamorphic testing, one of the unsolved problems is developing a methodology for devising metamorphic relations. A widely applicable method is still missing for programs that solve different applied problems in many areas. This leads to a effort-intensive compilation of metamorphic relations practically from scratch in each specific case.

In this study, we set the following tasks to overcome difficulties in devising metamorphic relations.

(1) Identify common and reusable techniques (patterns) for devising metamorphic relations in different problem areas. The results are given in Section 4.

(2) Determine specific traits of metamorphic testing for machine learning and data analysis programs that may allow for constructing more effective relations. More details are in Section 5.

(3) Identify indirect methods for obtaining the relations, as well as ways to combine relations with other methods in order to simplify the use of metamorphic testing in non-standard cases. The results are collected in Section 7.

(4) Identify omissions and opportunities for further development of methods for devising metamorphic relations. The results are given in Section 3 and Section 6.

We apply a systematic review method [7] to achieve the goals of the study. B. Kitchenham proposed this method for conducting meta-research (secondary study) in software engineering. This method can improve the reproducibility of the results and the validity of the conclusions.

To select publications, the following criteria were used: novelty, publication in the period from 2018 to 2023, accessibility to the reader, relevance to the research topic, detailed and clear description of the test invariants used, the presence of practical results in the work, and the possibility of generalizing the ideas used and techniques. The quality of the proposed invariants was not considered a search criterion, since the total number of studies on the topic is not enough to apply statistical methods for determining the average effectiveness of a particular invariant.

In order to outline the basic results of invariant testing, the review also includes seminal works in each area, some of which were published before 2018. To select them, the following criteria were used: relevance to the topic, citation, depth and versatility of the proposed ideas and methods, and practical value. The search followed the same rules, but no publication date restriction was applied. During the search, lists of sources from the most famous English-language review articles [4, 6] on this topic were additionally used. Also, seminal works have been replaced by sources that cite and reuse test invariants from these earlier works without significant modifications.

We searched for publications in Google Scholar in English using the keywords “metamorphic testing”, “metamorphic relation”, “neural network”, “classification”, “maps”, “graphs”, “natural language processing” and combinations of these words. We identify some additional publications by reviewing the reference lists from these publications. The presence of the word “metamorphic” in the source text is necessary since there are no alternative names for this method in the English literature. The search excluded patents and citations, review articles on related fields and various testing methods, and articles that only mentioned usage of metamorphic testing. The search was conducted from June 14 to September 13, 2023.

We found more than two thousand publications using the keywords “metamorphic testing” for the period from 2019 to 2023. This fact indicates the relevance and interest of researchers in this topic. We selected and studied about 120 papers and included 66 of them in this study.

We also searched for Russian-language publications in eLibrary. We found only one work on the topic, a short paper [8], and one article from a related field of verification of flowcharts [9].

We reviewed a description of the subject area, the solved problem, and the relations proposed in each publication. Then, we grouped this information by subject areas and identified the most popular ideas and techniques in each area. We also selected a brief description of the method from the reviewed articles that describe techniques for devising the relations.

Finally, we generalize the selected approaches. At this stage, we form a classification of methods for developing metamorphic methods based on input data transformations. Using this classification, we describe patterns for developing relations that can be applied in many areas.

3. Classification of metamorphic relations

An analysis of the selected metamorphic relations showed that they could be rewritten in the same form (2). We formulated a classification of metamorphic relations based on the types of input data transformations

Table 1. Classification by transformation of input data

Transformation	Meaning
LT linear transform	linear transformation (including increasing or decreasing, multiplication by a constant)
SM symmetry	reflectional and rotational symmetries, symmetrical relations and others
PR permutative	permutation of elements of initial data
IE inclusive / exclusive	inclusion/exclusion of elements from a set
BC blur / change	replacement or modification, adding random noise
UD unite / divide	merging or splitting input data

I (see table 1). These transformations allow us to obtain the input data $I(x_1, x_2, \ldots, x_n)$ for metamorphic relations.

The classification we describe does not include a few rare metamorphic relations tied to a specific problem so that their generalization is currently of no interest. A few examples of such relations are described in section 4.2.

4. Patterns for devising metamorphic relations

In this section, when we describe the methods used in the literature for compiling metamorphic relations, we will immediately indicate in parentheses, which of the previously identified classes it belongs to. For example, multiplying numerical parameters by a non-zero constant can be classified as linear transformations (LT).

4.1. Common techniques

Popular techniques for testing models and algorithms are changing parameters, rearranging equivalent parameters, and adding noise.

Often, researchers use a parameter change (LT) to clearly predict the change in the result (whether it will improve, worsen, change by a known amount, or not change at all). Examples include changing the number of computing devices [10], changing the parameters of the epidemiological model [11, 12], parameters and hyperparameters of neural network models [13].

Permutations of equivalent parameters or inputs (PR) are a popular method. Frequently, one can expect the same output of the program execution. For example, permutation of users of cloud services [10], operations performed under multi-threading conditions [14], and input data in stochastic optimization [15] should not affect the program output. The order in which one adds requirements when booking a hotel using a bot [16] should not affect the search results.

Adding noise (BC) is a widely used technique: errors [17, 18], noise [19–21], distortions [22], use of language features [23]. Usually, researchers expect that the added noise will not change the output, or at least will not improve it. If one replaces the original data (BC), the result usually be expected to change. For example, we can cite the already mentioned relation from section 1 with the conditions A and $\neg A$. With such a change in the input, the output should completely change and not intersect with the previous one.

Adding or removing parts of the source data (UD) is less common. For example, adding new users of cloud services [10], adding and removing language units from the text [24], parts of the input data [18, 25].

The technique of using symmetrical inputs is fruitful in the areas of imagery and geoinformatics. For example, reflection of a chessboard [26] when testing chess playing algorithms, reflection of images relative to the axes [27], time reversal of video sequences [27], rotations and reflections of card images [28], reflections of detected data clusters over a line [29].

4.2. Using the properties of the mathematical model and the subject area

One can use the properties of a mathematical model of the problem or a program to obtain metamorphic relations. For example, Guderlei et al. [30] compare the sampling results to the given statistical distribution. Researchers test deep neural network compilers [31] by adding source code that leads an equivalent program but changes the resulting computational graph (IE). Both the properties of the chessboard and the ranking of the pieces are used to test chess algorithms [26], for example, reflections of the board (SM), replacement of the pieces with ones of a different color (PR), or stronger or weaker ones (BC). In article [32], the authors propose to apply matrix transformations (LT) and modulo operations (LT) to construct metamorphic relations for testing hashing algorithms that use matrix transformations and the RSA algorithm. Removing particular bits in the message (IE) and splitting the message into parts (UD) are applied to test another hashing algorithm [33]. In these cases, the hash value should change after transformations.

Sometimes, domain features are useful for constructing metamorphic relations. Testing a comparative genetic analysis system [34] uses the ability to separate mutations into non-overlapping classes. The work on testing driver assistant systems [35] uses some considerations about labeled maps; for example, highways and buildings do not have common areas, and forbidden parts of some paths do not intersect other paths. The work on testing an application with a microservice architecture [36] uses the features of the payment system. For example, the sum of creditor and debtor balances should remain the same after moving data from one microservice to another.

5. Methods for devising metamorphic relations in different areas

5.1. Search algorithms

When testing search algorithms, one checks if the program output changes in the expected way after adding or removing search conditions (IE). New elements should not appear in the output after adding a new condition [36, 37], and old ones should not disappear after removing a condition. A paper on testing the hotel booking bot uses changing the order of adding search terms (PR) [16]. Another paper [27] uses symmetry: search results sorted in descending order can be obtained from the results sorted in ascending order after reversing (SM). The product price should not depend on the search query (BC) [27], and the search results should be the same for different languages (BC) [27] and for users around the world (BC). Such a relation would test if outputs differ due to the inconsistencies among different database replicas in a distributed system.

5.2. Machine learning and data analysis

Due to the complexity of testing machine learning systems and data analysis algorithms, researchers propose many interesting techniques and transformations of input data. One of the common techniques is a repartition of the training dataset (UD). For example, adding or excluding an element from the data sample [29, 39, 40], adding or excluding entire classes [39, 40], duplicating elements of the training set [39], changing labels of data elements (BC)[40], addition and exclusion of features (IE) [29, 39, 40]. The exclusion of non-important features should not significantly change the classification quality, and the exclusion of significant ones should not improve. Permutations (PR) of features [39, 40], class labels [39, 40], elements of the training set [29], as a rule, should not significantly change the final quality of the classifier. A paper on testing prediction tools for protein properties [41] uses transformations of proteins that are guaranteed to change their function (BC), and the model should also change the prediction.

Linear transformations (LT) are less popular in this area. One can use the application of an affine transformation to sample features [40], rotations, and transformations of the coordinate system in space [29] (such transformations should not significantly affect the program output). Xie et al. [29] propose several more techniques for testing clusterization algorithms: compressing selected clusters to their centers (LT), adding elements inside the convex hull of clusters (UD), and adding outliers (UD). The idea of symmetry (SM) results as a technique of reflecting clusters over a line [29].

5.3. Deep learning and neural networks

One can use the methods proposed for testing machine learning models to test deep neural networks. For example, linear feature transformation (LT) [13, 19, 42]), permutation of class labels (PR) [13], repartition of data into train and test samples (UD) [13], data duplication (UD) [13], increasing or decreasing the learning algorithm parameters (LT) (e.g. learning rate, the α parameter in the nonlinear activation function ReLU, the number of training epochs, the number of neurons in layers [13]).

Another set of techniques is applied for testing image processing algorithms. Checking for quality of facial recognition uses changing the hair color, eyebrows, and sex, and adding glasses, mustache, and beard (BC) [43]. Testing of self-driving cars involves changing the weather (BC) [44, 45] or replacing the background (BC) [46] on the pictures. Xu et al. [47] propose the idea of using inequalities for the detection logits on the image and on its parts (UD). Other techniques use symmetries (SM): reflection [27], video reversal [27], as well as perspective transformations (LT) [48] and brightness change (LT) [48]. A new idea is to add watermarks (BC) and masks (UD) [48]. Park et al. [49] add new objects to the images (UD) to test the model.

One can check robustness of the model with noise reduction (BC) [19–21] and data distortion (BC) [22]. Another technique is to use the permutations of elements of the training sample (PR) [42], training and target features (PR) [42], image channels [48].

5.4. Natural language processing

A connection between changes in the text and changes in its meaning or other parameters under consideration is unclear. Therefore, input data transformations that do not change the system output are the most popular for testing natural language processing systems. Thus, techniques for input data transformation use the spelling and grammar features of languages, for example, English and Chinese.

A common technique is to add a substitution that should not change the program output (BC). Replacement of a morphological or semantic entity property, for example, gender [50], is used to recognize discrimination. Robustness check uses the replacement of a word with its synonym, translation into another language or an unrelated word [16, 23, 51], replacement of symbols with similar ones (visually or phonetically) or with '*' [23]. Other transformations are voice change (BC), replacement of certain words, for example, «before» with «after» (BC), change in the order of sentence members (PR) [51], noise with typos (BC) [23], word skipping (IE) [16]. Wang et al. [23] propose several more ideas for systems to analyze the Chinese language: splitting words and characters into parts (UD), merging letter combinations (UD), rearranging letters in a word (PR), abbreviating to an acronym (UD), inserting special offers-indicators (IE) for recognition. Such transformations can be fruitful for assessing the robustness of the model.

In automatic translation, the replacement of a noun for another helps to determine translation consistency (BC) [52]. One can also use symmetries (SM): the source text and its back translation from another language [27, 53]; direct translation from language A to language B and translation from language A to language B via a third language C [54].

Researchers successfully apply metamorphic testing to named entity recognition systems. An article [24] on recognizing entities in the text uses transformations that preserve existing entities: paragraph permutations (PR), permutations of a list of added random words (PR), and sentence merging. Adding and deleting random words, sentences, and paragraphs (UD) should not result in the disappearance of previously recognized entities. Sun et al. [55] use swapping entities (SM) and replacing entities (BC) to test a system that searches for relationships between entities. These transformations do not affect the existence of the relationship and use its symmetry.

5.5. Graph models. Geoinformatics

Graph models are used for various tasks like searching for routes [28] and modeling gene interactions [17]. One can structurally modify the graph model by adding or removing vertices and edges. For example, Chen et al. [17] consider a tool for modeling gene regulatory networks. They use transformations that obviously increase or decrease indicators, for example, changes in the edge or vertex parameter (LT), adding or removing vertices and edges with different properties (IE). The work [56] on testing a model that simulates the spread of news in a social network also uses changes in parameters with predictable results: changes in weights, types, and parameters of neighboring vertices, and others.

Important mathematical properties of the graph model for geographical data are the symmetry of the reachability and the triangle inequality. Based on them, several authros propose to swap the beginning and the end of the route (SM) [27, 28] and apply the triangle inequality to the lengths of routes A-C-B and A-B (UD) [28] and to their costs (UD) [38]. The triangle inequality also applies when adding and removing obstacles (IE) [28], adding conditions (IE) [38]. Some relations use robustness assumptions (BC): a small change in the starting points does not dramatically change the program output [38], and the same request after a short period of time results in the same output [27].

In cartography and route planning software, researchers additionally use transformations that change only the presentation of the map image. For example, changing the coordinate system [28] (LT), rotational symmetries, and reflecting the map over a line [28] (SM). As a result, such transformations do not change the output. When the scale of the image is changed, they change the path length proportionally.

Iqbal et al. [35] use domain-specific features of labeled maps. They check that each path has at least one start and one finish, every roundabout has an entrance and an exit, and the forbidden parts of some paths do not intersect other paths.

5.6. Bioinformatics (texts)

Metamorphic testing is applied to testing aligners. An aligner is a program that determines the coordinates of small sections of nucleic acids read by a sequencer (so-called reads) on the reference genome. If an aligner calculates the read coordinates successfully, this read is called mapped; otherwise, it is called unmapped.

The use of inverted reads [17, 18] is an example of reflections. Researchers also propose changing the order of reads in the input (PR) [18], permutation of letters in the alphabet (PR) [17], adding [18, 25] and deleting [18] reads (UD), using only mapped or unmapped reads (UD) [18, 25], reads expansion (IE) [18], adding and removing sets of reads (UD) [17], changing the maximum allowed number of mismatches [17]. Also, one can add [17] and remove [17, 18] errors in reads (BC), replace errors with errors of a different type (BC) [17]. These operations may require a comparison of reads with the corresponding regions of the reference genome.

5.7. Compilers

A compiler is a program that converts code written in a programming language into a set of machine codes. The main idea used to test compilers is to create a program equivalent to the original one. Donaldson et al. [57] add dead code and identity functions (IE). Le et al. [58] use program reduction (BC) through sophisticated code coverage analysis.

Nowadays, some specialized compilers transform a high-level deep neural network model into optimized executable code. Xiao et al. [31] add complex constructions to test such a compiler. These constructions create a duplicate of the initial program but change the resulting computation graph (IE).

6. Analysis of the application of methods for relation construction by areas

We compile a summary table 2 based on the input data transformations patterns. The table cells contain objects, parameters, and properties. The column indicates the method of input transformation, and the row indicates the area. One can note that these transformations often change the minimal indivisible units from the area under consideration, such as a vertex or an edge of a graph, a sample element, a language unit, or their representation in computer memory.

Table 2. Transformations of input data for metamorphic relations

Area	LT	SM	PR	IE	BC	UD
Search algorithms		output order	order of conditions	a condition	language, send position	
Machine learning	a scale, pa-rameters	symmet-rical object	equivalent properties	properties	a label	a sample
Deep neural networks	properties, hyperpa-rameters	symmet-rical object	properties, classes	new objects	an object, a back-ground	a sample, an image
Geoinformatics	coordinate system	coordinate system, relations	a start and a finish of the path	obstacles, conditions	a point	paths
Graph models	an edge	a start and a finish of the path		vertices		paths
Natural language processing		languages, entities	language items	words, indicators	language and items	language items
Bioinformatics		reads, an alphabet	reads, an alphabet	reads, mutations, genes	reads, errors	reads
Compilers				syntax constructions	text of the program	

We were not able to find relations of all six identified types for all the considered areas, which may serve as a direction for further research. In addition, discovered relations have not yet used transformations of all possible units of the areas under consideration. For example, one can add noise to the capacities and other characteristics of vertices and edges in a graph (BC) and rearrange the edges themselves (PR) when storing the graph in the form of adjacency lists. In natural language processing, one can try to use the vector representation of words to apply linear transformations (LT). In object detection, collage images can help to compare results (UD). For search queries, one can change the filter parameter (LT) and compare the number of responses for different parameter ranges (UD). For example, the total number of responses equals the sum of the number of responses for the two halves of this range.

Thus, the resulting table shows the following algorithm for compiling relations. Researchers consider the minimal units in the subject area and apply transformations from the six specified groups. One can obtain a metamorphic relation if it is possible to say unambiguously what change in the output the transformation should lead to.

7. Other methods for metamorphic relations

In addition to obtaining relations directly from the problem statement or model properties, it is possible to make compositions of already existing metamorphic relations, combine them with the methods of mathematical statistics for application to stochastic systems, and combine them with other methods. These techniques also help to obtain new, sometimes more effective, relations, so let us consider them in more detail.

Application of statistical methods in metamorphic testing. Random errors may occur during the testing of stochastic systems with metamorphic testing, so one can use statistical approaches to verify the correctness of relations. One of the first works on statistics in metamorphic testing [30] considers testing the properties of a specific statistical distribution implementation as an example. More recent works use statistical methods to determine the fulfillment of relations (ANOVA [54], Spearman's rank correlation [59], a criterion for testing the presence of statistically significant differences [60, 61]). The work on testing multi-armed bandits [74] proposes another generalization: the relation is defined as a composition of the procedure for obtaining a sample and a statistical criterion.

Search for relations. Relations are not always easily derived from the problem statement or model of the solution algorithm. In addition, this process is currently poorly automated. Therefore, a separate direction in metamorphic testing considers searching for approaches for metamorphic relations [62, 63]. Zhang et al. [64] propose an approach for searching metamorphic relations as polynomial functions, and Sun et al. [65] search the relations dynamically by categories of inputs and outputs. The work [66] proposes the SVM algorithm application on program graphs to predict relations. Blasi et al. [67] propose to search for descriptions in natural language that can be converted into metamorphic relations.

Special types and compositions of relations. One of the early works in this field [68] proposes iteratively selecting input data for test cases. The paper [69] develops this idea and applies it to testing the linear model. The output of the first test run is used to get the next one. The work [39] proposes the idea of multidimensional metamorphic relations. The work on spectrum-based fault localization [70] introduces the concept of a metamorphic slice.

An early work on relation composition [71] examines non-trivial composition functions and shows that the defect discovery effectiveness of the composition usually exceeds the total effectiveness of individual relations. Authors of [72] conduct a theoretical study of the possibilities of composition and discuss some examples. The article on checking geographic systems [73] uses relations compositions of the form z(y(x())) with 2-4 functions, and uses the simple relations intended to check various properties: test requirements, program properties, algorithm properties, etc. The work on testing a bioinformatic pipeline [34] proposes a method for compiling compositions of metamorphic relations for systems using logical functions.

Combinations with other methods. Metamorphic testing can be combined with fuzzing. hlZhou et al. [21] slightly distort the LIDAR data, which makes it possible to evaluate the system's robustness. An interesting technique is the use of metamorphic testing and adaptive random testing [75]: when generating a subsequent test, the distance to the three previous ones is measured. In addition, metamorphic testing is used in spectrum-based fault localization [70] and in combination with genetic algorithms [76].

Conclusion

This systematic review examines methods and patterns for constructing metamorphic relations in various fields of knowledge. We analyze the available literature, identify the most frequently applied techniques, and examine popular application areas of metamorphic testing in detail.

As a result of the study,

- We have identified six common types of input data transformations that are used in the formulation of metamorphic relations: linear transformations, symmetries, permutations, adding or removing elements, replacing or adding noise, and merging or splitting into parts.
- We have analyzed and compared relations for problems from different fields of knowledge;
- We have considered indirect methods for obtaining relations, such as creating compositions of existing ones, using statistical approaches, and combining them with other methods;
- We have obtained a summary table of the variable units in each area and identify opportunities for compiling new relations.

The results of this analysis can contribute to the development of a general theory of building metamorphic relations, the popularization of metamorphic testing, and the creation of new test automation technologies and applications. We hope our work will help researchers apply metamorphic testing more easily and quickly.

References

[1] W. E. Wong, V. Debroy, A. Surampudi, H. Kim, M. F. Siok. "Recent catastrophic accidents: Investigating how software was responsible", 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement (09-11 June 2010, Singapore), 2010, pp. 14–22. doi ↑62

[2] W. Howden. "Theoretical and empirical studies of program testing", *IEEE Transactions on Software Engineering*, **SE-4**:4 (1978), pp. 293–298. doi ↑62

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo. "The oracle problem in software testing: A survey", *IEEE Transactions on Software Engineering*, **41**:5 (2015), pp. 507–525. doi ↑63

[4] T. Y. Chen, S. C. Cheung, S. M. Yiu. *Metamorphic testing: a new approach for generating next test cases*, 2020, 11 pp. arXiv 2002.12543 ↑63, 66

[5] T. Y. Chen, T. Tse. "New visions on metamorphic testing after a quarter of a century of inception", *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1487–1490. ↑63

[6] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, Z. Q. Zhou. "Metamorphic testing: A review of challenges and opportunities", *ACM Computing Surveys*, **51**:1 (2018), pp. 1–27. ↑63, 66

[7] B. Kitchenham. *Procedures for performing systematic reviews*, Keele University Technical Report TR/SE-0401, Keele University, Keele, UK, 2004, 33 pp. ↑66

[8] R. R. Fal'kovskij. "Metamorphic testing of image enhancement programs", *XIX Mezhdunarodnaya telekommunikacionnaya konferenciya molodyx uchenyx i studentov "MOLODEZh' I NAUKA"*, Tezisy dokladov. V. 3, NIYaU MIFI, M., 2015, ISBN 978-5-7262-2223-3, pp. 176–177 (in Russian). ↑67

[9] A. M. Mironov. "Verification of programs by the method of invariants", *Intellektual'nye sistemy. Teoriya i prilozheniya*, **21**:4 (2017), pp. 31–49 (in Russian). ↑67

[10] A. Núñez, P. C. Cañizares, M. Núñez, R. M Hierons. "TEA-Cloud: A formal framework for testing cloud computing systems", *IEEE Transactions on Reliability*, **70**:1 (2020), pp. 261–284. ↑68, 69

[11] L. L. Pullum, O. Ozmen. "Early results from metamorphic testing of epidemiological models", *2012 ASE/IEEE International Conference on BioMedical Computing (BioMedCom)* (14-16 December 2012, Washington, DC, USA), IEEE, 2012, pp. 62–67. ↑69

[12] A. Ramanathan, C. A. Steed, L. L. Pullum. "Verification of compartmental epidemiological models using metamorphic testing, model checking and visual analytics", *2012 ASE/IEEE International Conference on BioMedical Computing (BioMedCom)* (14-16 December 2012, Washington, DC, USA), IEEE, 2012, pp. 68–73. ↑69

[13] J. D. Ellis, R. Iqbal, K. Yoshimatsu. "Verification of the neural network training process for spectrum-based chemical substructure prediction using metamorphic testing", *Journal of Computational Science*, **55** (2021), id. 101456. ↑69, 71

[14] C.-A. Sun, H. Dai, N. Geng, H. Liu, T. Y. Chen, P. Wu, Y. Cai, J. Wang. "An interleaving guided metamorphic testing approach for concurrent programs", *ACM Transactions on Software Engineering and Methodology*, **33**:1 (2023), id. 8, 21 pp. ↑69

[15] S. Yoo. "Metamorphic testing of stochastic optimisation", *2010 Third International Conference on Software Testing, Verification, and Validation Workshops* (06-10 April 2010, Paris, France), IEEE, 2010, pp. 192–201. ↑69

[16] J. Bozic, F. Wotawa. "Testing chatbots using metamorphic relations", *Testing Software and Systems*, 31st IFIP WG 6.1 International Conference, ICTSS 2019 (15-17 October 2019, Paris, France), Lecture Notes in Computer Science, vol. **11812**, eds. Gaston C., Kosmatov N., Le Gall P., Springer, Cham, 2019, ISBN 978-3-030-31279-4, pp. 41–55. ↑69, 70, 72

[17] T. Y. Chen, J. W. Ho, H. Liu, X. Xie. "An innovative approach for testing bioinformatics programs using metamorphic testing", *BMC bioinformatics*, **10** (2009), id. 24, 12 pp. doi ↑69, 73, 74
[18] E. Giannoulatou, S.-H. Park, D. T. Humphreys, J. W. Ho. "Verification and validation of bioinformatics software without a gold standard: a case study of BWA and Bowtie", *BMC bioinformatics*, **15** (2014), id. S15, 8 pp. doi ↑69, 74
[19] Y. Tian, K. Pei, S. Jana, B. Ray. "DeepTest: Automated testing of deep-neural-network-driven autonomous cars", *Proceedings of the 40th International Conference on Software Engineering* (27 May 2018-3 June 2018, Gothenburg, Sweden), ACM, New York, 2018, ISBN 978-1-4503-5638-1, pp. 303–314. doi ↑69, 71, 72
[20] C. Wu, L. Sun, Z. Q. Zhou. "The impact of a dot: Case studies of a noise metamorphic relation pattern", *2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET)* (26 May 2019, Montreal, QC, Canada), IEEE, 2019, pp. 17–23. doi ↑69, 72
[21] Z. Q. Zhou, L. Sun. "Metamorphic testing of driverless cars", *Communications of the ACM*, **62**:3 (2019), pp. 61–67. doi ↑69, 72, 77
[22] S. Nakajima, T. Y. Chen. "Generating biased dataset for metamorphic testing of machine learning programs", *Testing Software and Systems*, 31st IFIP WG 6.1 International Conference, ICTSS 2019 (15-17 October 2019, Paris, France), Springer, 2019, pp. 56–64. doi ↑69, 72
[23] W. Wang, J.-t. Huang, W. Wu, J. Zhang, Y. Huang, S. Li, P. He, M. R. Lyu. "Mttm: Metamorphic testing for textual content moderation software", *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, IEEE, 2023, pp. 2387–2399. doi ↑69, 72
[24] M. Srinivasan, M. P. Shahri, I. Kahanda, U. Kanewala. "Quality assurance of bioinformatics software: a case study of testing a biomedical text processing tool using metamorphic testing", *Proceedings of the 3rd International Workshop on Metamorphic Testing* (27 May 2018-03 June 2018, Gothenburg, Sweden), ACM, New York, ISBN 978-1-4503-5729-6, pp. 26–33. doi ↑69, 72
[25] M. Troup, A. Yang, A. H. Kamali, E. Giannoulatou, T. Y. Chen, J. W. Ho. "A cloud-based framework for applying metamorphic testing to a bioinformatics pipeline", *Proceedings of the 1st International Workshop on Metamorphic Testing* (14-22 May 2016, Austin, Texas), ACM, New York, 2016, ISBN 978-1-4503-4163-9, pp. 33–36. doi ↑69, 74
[26] M. Méndez, M. Benito-Parejo, A. Ibias, M. Núñez. "Metamorphic testing of chess engines", *Information and Software Technology*, **162** (2023), id. 107263. doi ↑69
[27] Z. Q. Zhou, L. Sun, T. Y. Chen, D. Towey. "Metamorphic relations for enhancing system understanding and use", *IEEE Transactions on Software Engineering*, **46**:10 (2018), pp. 1120–1154. doi ↑69, 70, 72, 73
[28] J. Zhang, Z. Zheng, B. Yin, K. Qiu, Y. Liu. "Testing graph searching based path planning algorithms by metamorphic testing", *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)* (01-03 December 2019, Kyoto, Japan), IEEE, 2019, id. 158, 9 pp. doi ↑69, 73

[29] X. Xie, Z. Zhang, T. Y. Chen, Y. Liu, P.-L. Poon, B. Xu. "METTLE: A METamorphic testing approach to assessing and validating unsupervised machine learning systems", *IEEE Transactions on Reliability*, **69**:4 (2020), pp. 1293–1322. ↑69, 71

[30] R. Guderlei, J. Mayer. "Statistical metamorphic testing testing programs with random output by means of statistical hypothesis tests and metamorphic testing", *Seventh International Conference on Quality Software (QSIC 2007)* (11-12 October 2007, Portland, OR, USA), IEEE, 2007, pp. 404–409. ↑69, 76

[31] D. Xiao, Z. Liu, Y. Yuan, Q. Pang, S. Wang. "Metamorphic testing of deep learning compilers", *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, **6**:1 (2022), id. 15, 28 pp. ↑69, 74

[32] C.-A. Sun, Z. Wang, G. Wang. "A property-based testing framework for encryption programs", *Frontiers of Computer Science*, **8** (2014), pp. 478–489. ↑70

[33] N. Mouha, M. S. Raunak, D. R. Kuhn, R. Kacker. "Finding bugs in cryptographic hash function implementations", *IEEE Transactions on Reliability*, **67**:3 (2018), pp. 870–884. ↑70

[34] S. Iakusheva, A. Khritankov. "Composite metamorphic relations for integration testing", *Proceedings of the 2022 8th International Conference on Computer Technology Applications* (12-14 May 2022, Vienna, Austria), ACM, New York, 2022, ISBN 978-1-4503-9622-6, pp. 98–105. ↑70, 77

[35] M. Iqbal. "Metamorphic testing of advanced driver-assistance systems: Implementing Euro NCAP standards on OpenStreetMap", *2023 IEEE/ACM 8th International Workshop on Metamorphic Testing (MET)* (14 May 2023, Melbourne, Australia), IEEE, 2023, pp. 1–8. ↑70, 73

[36] G. Luo, X. Zheng, H. Liu, R. Xu, D. Nagumothu, R. Janapareddi, E. Zhuang, X. Liu. "Verification of microservices using metamorphic testing", *Algorithms and Architectures for Parallel Processing.* V. I, 19th International Conference, ICA3PP 2019 (9-11 December 2019, Melbourne, VIC, Australia), Springer, 2020, pp. 138–152. ↑70

[37] S. Segura, J. A. Parejo, J. Troya, A. Ruiz-Cortés. "Metamorphic testing of RESTful web APIs", *Proceedings of the 40th International Conference on Software Engineering* (27 May 2018–3 June 2018, Gothenburg, Sweden), ACM, New York, 2018, ISBN 978-1-4503-5638-1, pp. 882. ↑70

[38] J. Brown, Z. Q. Zhou, Y.-W. Chow. *Metamorphic testing of navigation software: A pilot study with Google Maps*, University of Wollongong Research Online, 2018, 12 pp. ↑73

[39] M. Jia, X. Wang, Y. Xu, Z. Cui, R. Xie. "Testing machine learning classifiers based on compositional metamorphic relations", *International Journal of Performability Engineering*, **16**:1 (2020), pp. 67–77. ↑71, 77

[40] P. Saha, U. Kanewala. "Fault detection effectiveness of metamorphic relations developed for testing supervised classifiers", *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)* (04-09 April 2019, Newark, CA, USA), IEEE, 2019, pp. 157–164. ↑71

[41] M. P. Shahri, M. Srinivasan, G. Reynolds, D. Bimczok, I. Kahanda, U. Kanewala. "Metamorphic testing for quality assurance of protein function prediction tools", *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)* (04-09 April 2019, Newark, CA, USA), IEEE, 2019, pp. 140–148. doi ↑71

[42] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. J. C. Bose, N. Dubash, S. Podder. "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing", *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis* (16-21 July 2018, Amsterdam, Netherlands), ACM, New York, 2018, ISBN 978-1-4503-5699-2, pp. 118–128. doi ↑71, 72

[43] H. Zhu, D. Liu, I. Bayley, R. Harrison, F. Cuzzolin. *Datamorphic testing: A methodology for testing ai applications*, 2019, 39 pp. arXiv 1912.04900 ↑71

[44] M. Zhang, Y. Zhang, L. Zhang, C. Liu, S. Khurshid. "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems", *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (3-7 September 2018, Montpellier, France), ACM, New York, 2018, ISBN 978-1-4503-5937-5, pp. 132–142. doi ↑71

[45] M. Raif, E.-M. Ouafiq, Rharras A. El, A. Chehri, R. Saadane. "Metamorphic testing for edge real-time face recognition and intrusion detection solution", *2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall)* (26-29 September 2022, London, United Kingdom), IEEE, 2022, pp. 1–5. ↑71

[46] Z. Zhang, P. Wang, H. Guo, Z. Wang, Y. Zhou, Z. Huang. "DeepBackground: Metamorphic testing for Deep-Learning-driven image recognition systems accompanied by Background-Relevance", *Information and Software Technology*, **140** (2021), id. 106701. doi ↑71

[47] L. Xu, D. Towey, A. P. French, S. Benford, Z. Q. Zhou, T. Y. Chen. "Enhancing supervised classifications with metamorphic relations", *Proceedings of the 3rd International Workshop on Metamorphic Testing* (27 May 2018-03 June 2018, Gothenburg, Sweden), 2018, pp. 46–53. URL ↑71

[48] R. Yan, S. Wang, Y. Yan, H. Gao, J. Yan. "Stability evaluation for text localization systems via metamorphic testing", *Journal of Systems and Software*, **181** (2021), id. 111040. doi ↑72

[49] H. Park, T. Waseem, W. Q. Teo, Y. H. Low, M. K. Lim, C. Y. Chong. "Robustness evaluation of stacked generative adversarial networks using metamorphic testing", *2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET)* (02 June 2021, Madrid, Spain), IEEE, 2021, pp. 1–8. doi ↑72

[50] P. Ma, S. Wang, J. Liu. "Metamorphic testing and certified mitigation of fairness violations in NLP models", IJCAI'20: Twenty-Ninth International Joint Conference on Artificial Intelligence (7-15 January 2021, Yokohama, Japan), 2021, pp. 458–465, id. 64. doi URL ↑72

[51] S. Chen, S. Jin, X. Xie. "Validation on machine reading comprehension software without annotated labels: a property-based method", *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (23-28 August 2021, Athens, Greece), ACM, New York, 2021, ISBN 978-1-4503-8562-6, pp. 590–602. doi ↑72

[52] L. Sun, Z. Q. Zhou. "Metamorphic testing for machine translations: MT4MT", *2018 25th Australasian Software Engineering Conference (ASWEC)* (26-30 November 2018, Adelaide, SA, Australia), IEEE, 2018, pp. 96–100. doi ↑72

[53] W. Gao, J. He, V.-T. Pham. "Metamorphic testing of machine translation models using back translation", *2023 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)* (15 May 2023, Melbourne, Australia), IEEE, 2023, pp. 1–8. doi ↑72

[54] D. Pesu, Z. Q. Zhou, J. Zhen, D. Towey. "A Monte Carlo method for metamorphic testing of machine translation services", *Proceedings of the 3rd International Workshop on Metamorphic Testing* (27 May 2018, Gothenburg, Sweden), ACM, New York, 2018, ISBN 978-1-4503-5729-6, pp. 38–45. doi ↑72, 76

[55] Y. Sun, Z. Ding, H. Huang, S. Zou, M. Jiang. "Metamorphic testing of relation extraction models", *Algorithms*, **16**:2 (2023), pp. 102. doi ↑73

[56] M. S. Raunak, M. M. Olsen. "Metamorphic testing on the continuum of verification and validation of simulation models", *2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET)* (02 June 2021, Madrid, Spain), IEEE, 2021, pp. 47–52. doi ↑73

[57] A. F. Donaldson, A. Lascu. "Metamorphic testing for (graphics) compilers", *Proceedings of the 1st International Workshop on Metamorphic Testing* (14-22 May 2016, Austin, Texas), ACM, New York, 2016, ISBN 978-1-4503-4163-9, pp. 44–47. doi ↑74

[58] V. Le, M. Afshari, Z. Su. "Compiler validation via equivalence modulo inputs", *ACM Sigplan Notices*, **49**:6 (2014), pp. 216–226. doi ↑74

[59] Z. Q. Zhou, T. Tse, M. Witheridge. "Metamorphic robustness testing: Exposing hidden defects in citation statistics and journal impact factors", *IEEE Transactions on Software Engineering*, **47**:6 (2019), pp. 1164–1183. doi ↑76

[60] J. Ahlgren, M. Berezin, K. Bojarczuk, E. Dulskyte, I. Dvortsova, J. George, N. Gucevska, M. Harman, M. Lomeli, E. Meijer, S. Sapora. "Testing web enabled simulation at scale using metamorphic testing", *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (25-28 May 2021, Madrid, Spain), IEEE, 2021, ISBN 978-1-6654-3869-8, pp. 140–149. doi ↑76

[61] F. u. Rehman, C. Izurieta. "Statistical metamorphic testing of neural network based intrusion detection systems", *2021 IEEE International Conference on Cyber Security and Resilience (CSR)* (26-28 July 2021, Rhodes, Greece), IEEE, 2021, pp. 20–26. doi ↑76

[62] F. Tambon, G. Antoniol, F. Khomh. *HOMRS: High order metamorphic relations selector for deep neural networks*, 2021, 33 pp. arXiv 2107.04863 ↑77

[63] P. Zhang, X. Zhou, P. Pelliccione, H. Leung. "RBF-MLMR: A multi-label metamorphic relation prediction approach using RBF neural network", *IEEE Access*, **5** (2017), pp. 21791–21805. doi ↑77

[64] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, H. Mei. "Search-based inference of polynomial metamorphic relations", *ASE '14: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (15-19 September 2014, Vasteras, Sweden), ACM, New York, 2014, ISBN 978-1-4503-3013-8, pp. 701–712. doi ↑77

[65] C.-A. Sun, A. Fu, P.-L. Poon, X. Xie, H. Liu, T. Y. Chen. "Metric$^+$+: A metamorphic relation identification technique based on input plus output domains", *IEEE Transactions on Software Engineering*, **47**:9 (2019), pp. 1764–1785. doi ↑77

[66] U. Kanewala, J. M. Bieman, A. Ben-Hur. "Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels", *Software: Testing, Verification and Reliability*, **26**:3 (2016), pp. 245–269. doi ↑77

[67] A. Blasi, A. Gorla, M. D. Ernst, Pezzè M., A. Carzaniga. "MeMo: Automatically identifying metamorphic relations in Javadoc comments for test automation", *Journal of Systems and Software*, **181** (2021), id. 111041. doi ↑77

[68] P. Wu. "Iterative metamorphic testing", *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*. V. 2 (26-28 July 2005, Edinburgh, UK), IEEE, 2005, ISBN 0-7695-2413-3, pp. 19–24. doi ↑77

[69] Y. Yang, Z. Li, H. Wang, C. Xu, X. Ma. "Towards effective metamorphic testing by algorithm stability for linear classification programs", *Journal of Systems and Software*, **180** (2021), id. 111012. doi ↑77

[70] X. Xie, W. E. Wong, T. Y. Chen, B. Xu. "Metamorphic slice: An application in spectrum-based fault localization", *Information and Software Technology*, **55**:5 (2013), pp. 866–879. doi ↑77

[71] H. Liu, X. Liu, T. Y. Chen. "A new method for constructing metamorphic relations", *2012 12th International Conference on Quality Software* (27-29 August 2012, Xi'an, China), IEEE, 2012, pp. 59–68. doi ↑77

[72] K. Qiu, Z. Zheng, T. Y. Chen, P.-L. Poon. "Theoretical and empirical analyses of the effectiveness of metamorphic relation composition", *IEEE Transactions on software engineering*, **48**:3 (2020), pp. 1001–1017. doi ↑77

[73] Z.-W. Hui, S. Huang, C. Chua, T. Y. Chen. "Semiautomated metamorphic testing approach for geographic information systems: An empirical study", *IEEE Transactions on Reliability*, **69**:2 (2019), pp. 657–673. doi ↑77

[74] S. Iakusheva, A. Khritankov. "Metamorphic testing for recommender systems", *Analysis of Images, Social Networks and Texts*, AIST 2023, Lecture Notes in Computer Science, vol. **14486**, eds. Ignatov D.I. et al., Springer, Cham, 2024, ISBN 978-3-031-54533-7. doi ↑76

[75] Z.-w. Hui, X. Wang, S. Huang, S. Yang. "MT-ART: A test case generation method based on adaptive random testing and metamorphic relation", *IEEE Transactions on Reliability*, **70**:4 (2021), pp. 1397–1421. doi ↑77

[76] D. Sobania, M. Briesch, P. Röchner, F. Rothlauf. "MTGP: Combining metamorphic testing and genetic programming", Genetic Programming. EuroGP 2023, Lecture Notes in Computer Science, vol. **13986**, eds. Pappa G., Giacobini M., Vasicek Z., Springer, Cham, 2023, ISBN 978-3-031-29572-0, pp. 324–338. doi ↑77

Received	22.11.2023;
approved after reviewing	30.03.2024;
accepted for publication	31.03.2024;
published online	14.05.2024.

Recommended by *Andrey V. Klimov*

Information about the authors:

Sofia Fedorovna Iakusheva

Research interests: development and verification of programs, metamorphic testing, machine learning

iD 0009-0000-1423-1123

e-mail: *yakusheva.sf@phystech.edu*

Anton Sergeevich Khritankov

Research interests: software engineering, machine learning, trustworthy artificial intelligence

iD 0000-0003-2889-9436

e-mail: *akhritankov@hse.ru*

Authors contribution: *Sofia F. Iakusheva* — 70% (software, investigation, resources, data curation, writing (review & editing), visualization, supervision); *Anton S. Khritankov* — 30% (methodology, software, formal analysis, visualization, project administration, funding acquisition).

The authors declare no conflicts of interest.