Predicting Metamorphic Relations Based on Path Features

Hanyue Zhang 1,a, Lei Liu 2,b, Peng Zhang 2,c

1 Software College Jilin University ChangChun, China
2 College of Computer Science and Technology Jilin University ChangChun, China

a hanyuez18@mails.jlu.edu.cn, b liulei@jlu.edu.cn, c zhangpengccst@jlu.edu.cn

Abstract. Metamorphic test has been proposed to effectively solve the oracle problem, but the most of the existing metamorphic relations are difficult to reuse, which leads to a large cost of in the metamorphic test. In order to improve the efficiency of the metamorphic test and solve the problem of low reuse rate of metamorphic relations, based on the common set of metamorphic relations in scientific computing programs and execution path, we propose a novel string feature with a new extraction method. Then, this feature can be used to train support vector machine to decide the metamorphic relations for test. At last, we construct a various of experiments, and the experiments results show that our method can effectively predict the satisfaction of the input features for the metamorphic relations.

Keywords: Metamorphic test; metamorphic relations; feature extraction.

1. Introduction
During test period, the output of scientific computing programs are often very complicated, and testers cannot directly judge the correctness of the test results, so the oracle problems widely exist in the test. The metamorphic test proposed by Chen et al [1] in 1998, is an effective test method to solve oracle problems. The metamorphic test checks the program by constructing multiple sets of relations between input and output, that is, the metamorphic relations. The ability of the metamorphic relations to identify program errors directly affects the test performance. When conducting metamorphic testing, testers need to construct metamorphic relations from program functions. Therefore, metamorphic relations are difficult to reuse. Reconstructing the metamorphic relations for each program increases the time cost of testing, at the same time, constructing a metamorphic relation with a high error detection rate requires higher tester’s ability [2].

Chen et al. [3] believed that the specific attributes represented by the metamorphic relations are part of the software requirements, and proposed a method based on the category-choice framework to identify the metamorphic relations according to the software specification document. The framework recognizes the categories and choices in the program, identifying the metamorphic relations by checking the constraints between choices. This method relies on software specification documents, so it is not suitable for identifying the metamorphic relations of scientific computing programs. Zhang et al. [4] proposed a multi-label metamorphic relations prediction method based on an improved radial basis function (RBF) neural network. The algorithm uses affinity propagation algorithm and K-Means clustering algorithms to optimize the RBF neural network structure.
Upulee Kanewala et al. [5-7] proposed three methods training support vector machines to predict metamorphic relations. The first method directly input the labeled features to train the support vector machine. The features have node features and path features. The node features include the operation performed by the node and the in-degree and the out-degree of the node; The path feature is the sequence of nodes in the shortest path from the start node to each node to the end node. The second method calculates the graph kernel to compare the similarity of two graphs which representing the control flow and data dependency information of each pair of functions, and applies the calculated graph kernel to the support vector machine to create a predictive model. Upulee Kanewala et al. [8] used this method to predict the metamorphic relations of the matrix calculation programs. The third method uses a semi-supervised binary classification algorithm to predict the metamorphic relations, and applies the path features extracted by the control flow graphs to the support vector machine and the label propagation algorithm at the same time. Although the graph kernel and other methods can effectively predict the metamorphic relations, the process of calculating the similarity of the graph is very complicated and has obvious defects in time complexity. Among them, the randomness of the random walk kernel moving upstream in the graph will also lead to redundant calculations.

Murphy et al. [9] summarized 6 common metamorphic relations in scientific computing programs. In order to improve the testing efficiency of scientific computing programs and reduce the difficulty of testing, a method for predicting metamorphic relations based on path features is proposed to meet this demand. The implementation of our method is training support vector machine through path features extracted from execution path of the program, the classification standard of support vector machine is the program’s satisfaction with the 6 metamorphic relations. The string kernel function is used when training the support vector machine, and the predicted label is given by calculating the cosine similarity between strings.

2. Introduction to Metamorphic Relations

The metamorphic relations consist of two parts: the relations $r_1$ between the program inputs and the relations $r_2$ between the program outputs, denoted as $(r_1, r_2)$ [10]. When the inputs of program satisfy $r_1$ and the output of the program satisfy $r_2$, then the program satisfies this metamorphic relation. With the goal of alleviating the problem that the metamorphic relations are difficult to be reused, our method reuses the 6 common metamorphic relations found by Murphy et al., and the satisfaction of these six metamorphic relations by the scientific computing program is used as the training standard. After the training is completed, the support vector machine can be used to predict the satisfaction of these six metamorphic relations by other programs. The metamorphic relations used in our method is shown in Table I. The follow input of the program is transformed by the source input according to the input relation $r_1$ in Table I. If the corresponding outputs result of the program satisfy $r_2$, the program meets the metamorphic relation. For example, the additive metamorphic relation in Table I adds a positive constant to the source test case to generate a follow test case. If the output result of the follow test case is greater than or equal to the output result of the source test case, the program satisfies the additive metamorphic relation; For the permutative metamorphic relation in Table I, the source test case exchanges two random elements to generate the follow test case. If the output of the follow test case is equal to the output of the source test case, the program satisfies the permutative metamorphic relation.

| Metamorphic Relation | $r_1$ | $r_2$ |
|----------------------|-------|-------|
| Additive             | Add a positive constant | Increase or remain |
| Exclusive            | Remove an element        | Decrease or remain |
| Inclusive            | Add a new element         | Increase or remain |
| Invertive            | Take the inverse of each element | Decrease or remain |
| Multiplicative       | Multiply by a positive constant | Increase or remain |
| Permutative          | Randomly permute two elements | Remain |

Table 1. Metamorphic relations
3. Metamorphic relations prediction method

On the basis of Section II, the method we proposed will be introduced below. The metamorphic relations prediction method includes feature extraction and support vector machine training. The overview of feature extraction method is shown in Fig. 1. First, we generate the control flow graph from the program’s source code. Next, we use numbers to represent the control flow graph as the form in (c), then we input (c) to the basic path generation program. (d) is the output of the program. Finally, we obtain the string features by replacing numbers in (d) with the abbreviation of the operation performed in each node. The node mapping rules are shown in Table II. We divide the obtained features into train set and test set, and input the train set into the support vector machine to train model. The output labels are the satisfaction of the metamorphic relations by the program, +1 represents satisfied, -1 means not satisfied.

| Node | Abbreviation |
|------|--------------|
| assi | A            |
| add  | P            |
| mul  | M            |
| if Lt| L            |
| if equal |       |
| if not |               |
| return | R            |

| Node | Abbreviation |
|------|--------------|
| goto | G            |
| sub  | S            |
| div  | D            |
| if Mt| M            |
| if not |               |
| -    | -            |

**Table 2.** Node mapping rules

### 3.1. Feature extraction

Since the quality of the features directly affect the classification performance of the support vector machine, the features we extracted should contain both the structural information and semantic information of the program. Considering the internal logic of the program, the basic path set determines all the independent execution paths contained in the program. Therefore, we use the execution path of the program as feature to preserve the logic structure and operation sequence of the program to the greatest extent. At the same time, we can remove the redundant information, and reduce the time and space complexity of our method. After converting the source programs into string features through our method shown in Fig. 1, each string feature represents an independent execution path of the program.
and the comparison of execution paths between programs is converted into a comparison of semantic similarity of text. The classic text classification method only retains the frequency of word but loses the order of word. It also cannot retain the structural information of the path feature. While, the string kernel function considers text as a sequence of symbols, and compares the number of identical subsequences in two strings when calculating the similarity of text. String kernel function can classify features under the premise of ensuring the semantic and structural integrity of path features, so we use string features training support vector machine.

The basic path generation is a dynamic analysis method, it describes the behavior of the program in the real execution state. Therefore the independent paths in the basic path set can represent the logic of the program when it is actually executed. While generating the basic path set, the control flow graph of the program is first required, and the cyclomatic complexity is determined according to the control flow graph. The cyclomatic complexity is also the number of independent paths included in the program, which determines the upper limit of the number of paths required when each executable statement is executed at least once. After using the non-recursive depth-first search strategy to traverse all the nodes in the control flow graph, a set of executable paths of the program is generated, that is, the basic path set.

The execution path used in our method is essentially the sequence of operations of the program. As the sequence of operations is a linear arrangement, a string can be used to represent the execution path of the program. By using the string kernel function, convert comparison of execution logic of two paths into comparison of semantic similarity of two strings. Without destroying the structure of features, string kernel function greatly reduces the difficulty of the calculation of similarity. The feature space of the string kernel function is composed of subsequences constituting the string, where the subsequences may not be continuous, and the dimension of the feature space is determined by the mapping function. The mapping function maps the string feature which representing a path to a high-dimensional feature space, and each string feature corresponds to a feature vector represented by the subsequence it contains. The kernel value of two string features is the inner product of two feature vectors. If there are more similar subsequences in the two strings, the larger their inner product is, the higher the similarity.

3.2. Support Vector Machine
Support vector machine is a widely used binary classifier, which can also be used to multi-classify data in combination with other algorithms. Support vector machines can not only process linear classification, but also use the kernel function to map the data into a high-dimensional feature space and find a hyperplane when processing nonlinear data. The classification can be realized according to the distribution of data on the hyperplane. Inspired by Upulee Kanewala et al., we used the string kernel function in the shogun toolkit to train support vector machines in a Python environment, and we classified path features based on the satisfaction of metamorphic relations. In our method, we trained 6 support vector machines, corresponding to the 6 metamorphic relations introduced in Section II. The shogun toolkit used in our method is an open source machine learning library based on C++. It provides multiple classification methods such as support vector machines, regression and neural networks, and integrates multiple language interfaces such as Python, Java, and c#.

4. Experimental analysis

4.1. Experimental data
Our experiment selected three third-party libraries written in Java, namely The Colt Project, Apache Commons Mathematics Library, and Apache Mahout. A total of 66 functions as our data set. Through the basic path generation program, these 66 functions are used to form a basic path set of 203 independent paths. Through the feature extracted method, we transform the basic path set into string features. Among them, 173 path features generated by 57 functions in The Colt Project and Apache Commons Mathematics library are used as the train set, and 30 path features generated by 9 functions in the Apache Mahout library are used as the test set.
4.2. Experimental results and analysis
After the train set is input to the support vector machine to train the models, we input the test set to the models as well. The accuracy of our models is shown in Fig. 2, and the result is to three decimal places. Six support vector machines were trained for the six metamorphic relations, among which the prediction accuracy of additive, exclusive and permutative is higher than 73.3%. The accuracy of multiplicative reaches 83.3%. However, the accuracy of inclusive and invertive has only 66.7% and 66.3%, under the same circumstances. Obviously, the prediction performance of the model corresponding to inclusive and invertive is not ideal. The possible reasons are as follows: First, the proportion of positive and negative examples in the train set are not balanced enough. For example, when the number of positive examples is much larger than negative examples, support vector machine tends to predict the result as a positive example; Secondly, the number of features used in test set is not sufficient, the result maybe not representative enough, consequently, the path features in the test set should to be supplemented in future experiments; Finally, the structure of path features used in our experiment should be improved. In subsequent experiments, we will try to remove the assignment and goto statements before the branch or operation statements.

Fig.2 Accuracy of SVMs.

5. Conclusion
We proposed a feature extraction method based on the program execution path, which is used to transform an basic path of program into a string feature. The basic path set of a program is transformed into a train set and a test set. Then input the train set and the test set to the support vector machine to train models. The accuracy of the models shows that our method can successfully predict the satisfaction of the six metamorphic relations by most programs.

In future experiments, improvements will be made through the following aspects: First, adjust the proportion of positive and negative examples in the train set. In this experiment, the ratio of positive examples is slightly larger than that of negative examples. More programs that show negative examples should be collected to train set, the classification capabilities of support vector machine will improve; Secondly, due to the small number of programs in our test set, more scientific computing programs in third-party libraries will be collected as test set. In addition, cross-validation and ROC curve will be added to evaluate the accuracy and the predictive ability of the model; Finally, we will adjust the structure of path features, and delete some assignment statements and goto statements without affecting the program execution logic.
Acknowledgments

I would like to extend my deep gratitude to all my teacher who taught me, esp. Prof. Liu Lei, and our group leader Zhang Peng. They guide me, influence me and help me in the process of writing this paper. I also want to thank my parents. Without their refined education and care, I could never grow up such a joyous environment.

References

[1] Chen TY, Cheung SC, Yiu SM. Metamorphic testing: a new approach for generating next test cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology: Hong Kong, 1998.

[2] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. A comprehensive survey of trends in oracles for software testing. Technical Report CS-13-01, University of Sheffield, 2013.

[3] Chen T Y, Poon P L, Xie X. METRIC: METamorphic Relation Identification based on the Category-choice framework[J]. Journal of Systems & Software, 2016, 116(jun.):177-190.

[4] Zhang P, Zhou X, Pelliccione P, et al. RBF-MLMR: A Multi-Label Metamorphic Relation Prediction Approach Using RBF Neural Network[J]. IEEE Access, 2017, 5:21791-21805.

[5] Kanewala U, Bieman J M. Using machine learning techniques to detect metamorphic relations for programs without test oracles[C]. international symposium on software reliability engineering, 2013: 1-10.

[6] Kanewala U, Bieman J M, Ben-Hur Asa. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels[J]. Software Testing Verification & Reliability, 2016, 26(3):245-269.

[7] Hardin B, Kanewala U. Using Semi-Supervised Learning for Predicting Metamorphic Relations[J]. arXiv: Software Engineering, 2018.

[8] Rahman K, Kanewala U. Predicting Metamorphic Relation for Matrix Calculation Programs[J]. arXiv: Software Engineering, 2018.

[9] Murphy C, Kaiser G E, Hu L, et al. Properties of Machine Learning Applications for Use in Metamorphic Testing[C]/ Twentieth International Conference on Software Engineering & Knowledge Engineering. DBLP, 2008.

[10] Guowei D, Baowen X U, Lin C, et al. Survey of Metamorphic Testing[J]. Journal of Frontiers of Computer ence & Technology, 2009.