Predicting Metamorphic Relations for Matrix Calculation Programs

Karishma Rahman
Montana State University
Bozeman, Montana
karishma.rahman@student.montana.edu

Upulee Kanewala*
Montana State University
Bozeman, Montana
upulee.kanewala@montana.edu

ABSTRACT

Matrices often represent important information in scientific applications and are involved in performing complex calculations. But systematically testing these applications is hard due to the oracle problem. Metamorphic testing is an effective approach to test such applications because it uses metamorphic relations to determine whether test cases have passed or failed. Metamorphic relations are typically identified with the help of a domain expert and is a labor intensive task. In this work we use a graph kernel based machine learning approach to predict metamorphic relations for matrix calculation programs. Previously, this graph kernel based machine learning approach was used to successfully predict metamorphic relations for programs that perform numerical calculations. Results of this study show that this approach can be used to predict metamorphic relations for matrix calculation programs as well.

KEYWORDS

Metamorphic testing, metamorphic relation, control flow graph, support vector machine, random walk kernel

ACM Reference Format:
Karishma Rahman and Upulee Kanewala. 2018. Predicting Metamorphic Relations for Matrix Calculation Programs. In *MET'18: MET'18:IEEE/ACM International Workshop on Metamorphic Testing , May 27, 2018, Gothenburg, Sweden.* ACM, Gothenburg, Sweden, Article 4, 4 pages. https://doi.org/10.1145/3193977.3193983

1 INTRODUCTION

Matrix calculations are common in scientific applications. Often, matrices represent data, graphs or mathematical equations in the applications. [7]. They can be used to get quick and good approximation for complicated calculation in time-sensitive engineering applications [7]. Moreover, matrix multiplication is used in graphics, digital videos and solving linear equations of particular variables in different applications [7]. But testing these applications is hard due to the difficulties associated with defining suitable test oracles [14]. This is known as the oracle problem [14]. *Metamorphic Testing (MT)* can be used to alleviate the test oracle problem [15]. MT conducts testing by checking whether the programs behave according to a set of *metamorphic relations (MRs)* [4]. A metamorphic relation specifies how the output should change according to a change made to the input [4]. MT operates as follows [4, 15]:

(1) Identify a suitable set of metamorphic relations which should satisfy the program under test.
(2) Create a set of initial test cases.
(3) Apply the input transformations specified by the identified MRs in Step 1 and create follow-up test cases for each of the initial test case.
(4) Execute the initial and follow-up test case(s) and check if the output change satisfies the change predicted by the MR. When testing a program, a run-time violation of an MR can mean that the program under test contains fault(s).

In a previous work [10], a *graph kernel-based machine learning method* was introduced to predict MRs for programs with numerical inputs and outputs that are represented with simple one-dimensional data structures such as arrays or lists. So in this work, we use the above method to predict MRs for functions performing matrix calculations. Typically, matrices are represented with two-dimensional data structures, thus the source code of these programs are dissimilar to the ones used in previous work.

This approach starts by creating the *control flow graphs (CFGs)* of each program, and the random walk kernel is used to compute the similarity between the graphs. The computed kernel values are used by a *support vector machine (SVM)* to automatically predict MRs for previously unseen functions. In this study, three high level categories of metamorphic relations are identified for the matrix-based programs and are used for the predictions. We used 55 functions obtained from open source matrix calculation libraries for evaluation of the method. Our result shows that for matrix-based calculations, the random walk kernel can effectively predict the MRs.

2 APPROACH

This section discusses the details of the metamorphic relation approach used in this study.

2.1 Function Representation

The first step of this method is to convert a function into its CFG. This representation is specifically used since it allows the extraction of information about the sequence of operations performed in a control flow path that is directly related to the MRs satisfied by a given function.

*Corresponding author

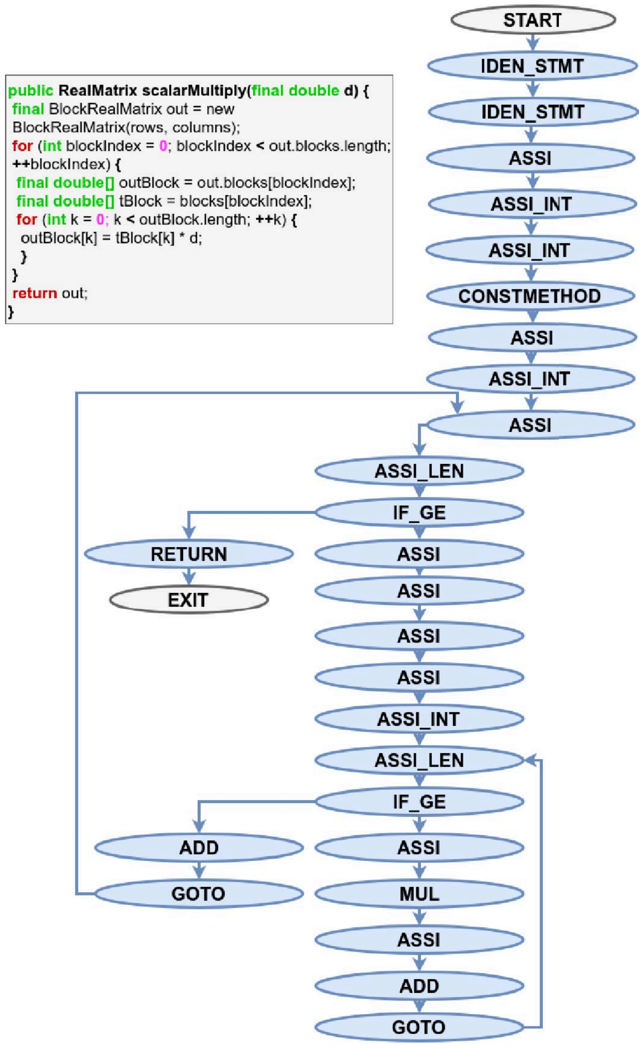
MET'18, May 27, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5729-6/18/05…$15.00
https://doi.org/10.1145/3193977.3193983

Table 1: The Metamorphic Relations used in the study

Metamorphic Relation Category	Change made to the input	Expected change in the output
Permutative	Permutation of all the elements Permutation of rows Permutation of columns	The matrix size will remain same
Additive	Scalar addition to matrix Addition of two or more matrices Addition to the subset of elements of the matrix	Element values will increase or remain same
Multiplicative	Scalar multiplication to matrix Multiplication to the subset of elements of the matrix	Element values will increase

A CFG is a directed graph $G_f = (V, E)$ of a function f. Here, x is a statement in f, represented by each node $v_x \in V$. The operation performed in each x are labeled $label(v_x)$. Supposedly if x and y are statements of f, after execution of x, y is executed. Then it can be said that e is an edge where $e = (v_x, v_y) \in E$. Control flow of f is represented by all the edges, and the starting and the exiting point are represented by nodes v_{start} and v_{exit} respectively [1].

We use the *Soot*[1] framework to create the CFGs. It generates CFGs in *Jimple*: a typed 3-address intermediate representation of the Java code, thus each CFG node represents an atomic operation [13]. We post-processed the generated CFGs by labeling all the nodes indicating the operation performed in each nodes. In addition we annotated all the method call nodes in the CFG with their return types. Figure 1 represents a function for calculating scalar multiplication of a matrix and its post-processed CFG representation.



```
public RealMatrix scalarMultiply(final double d) {
 final BlockRealMatrix out = new
 BlockRealMatrix(rows, columns);
 for (int blockIndex = 0; blockIndex < out.blocks.length;
++blockIndex) {
  final double[] outBlock = out.blocks[blockIndex];
  final double[] tBlock = blocks[blockIndex];
  for (int k = 0; k < outBlock.length; ++k) {
   outBlock[k] = tBlock[k] * d;
  }
 }
 return out;
}
```

Figure 1: Scalar multiply function and its post-processed CFG representation

2.2 Random Walk Kernel

After creating the CFG representation of the functions, the next step is to use a graph kernel to compute the similarity between the CFGs. In previous work [10], two graph kernels were used and among them, better performance was shown by the random walk kernel. Therefore we use the random walk kernel in this study. We briefly describe the idea of the random walk kernel in this section. More information about the random walk kernel and it's definition can be found in [10].

The random walk kernel computes the similarity score between two graphs by summing up the similarity scores of all the pairs of walks in the two graphs. The similarity score of a pair of walks is computed by multiplying the similarity scores of their corresponding step pairs. The similarity score of a pair of steps is computed by multiplying the similarity scores of node and edge pairs that make up the step. The similarity score of a node pair is determined by their node labels: if the two node labels are the same, then the pair is assigned a similarity score of one, else it is assigned a similarity score of zero. Also, if the two node labels represent operations with similar mathematical properties (but not identical), then the pair is assigned a similarity score of 0.5. Edge labels decide the value assigned for the similarity score of a pair of edges. Here, we only used one type of edge showing the flow of control between the operations. So the similarity score for a pair of edges is always one.

[1]https://www.sable.mcgill.ca/soot/

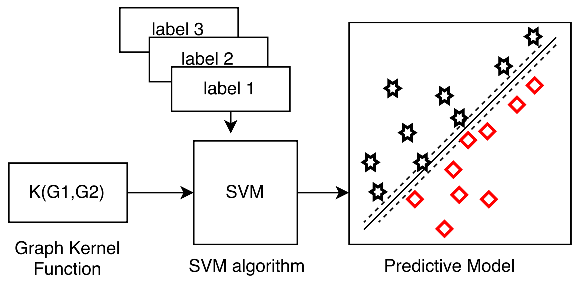


Figure 2: Overview of the predictive Model creation

Table 2: Number of positive and negative instances for each metamorphic relation

Metamorphic Relation	Positive instances	Negative instances
Permutative	14	41
Additive	37	18
Multiplicative	21	34

2.3 Predictive Model Creation

Support Vector Machine (SVM) is a supervised machine learning algorithm and it can be used for binary classification [5]. The computed random walk kernel values are supplied to a SVM with a binary label indicating whether a given function satisfies a given MR or not. The SVM uses the provided information to create a model that can predict if a new function would satisfy the considered MR or not. In this study, the SVM implementation from the scikit-learn[2] toolkit was used. Figure 2 shows the overview of the creation of the predictive model.

3 EXPERIMENTAL SETUP

This section describes the code corpus and MRs used in this study. The details of the evaluation procedure are also discussed here.

3.1 The Code Corpus

A total of 55 functions, all of which takes matrices as inputs and produces matrices as outputs, were used to measure the effectiveness of the method described in Section 2 for predicting MRs. These functions were collected from *Apache Commons Math Library*[3], which is an open source project. These functions execute a variety of calculations on matrices such as addition, multiplication, subtraction, and searching (e.g. getting column matrix, getting row matrix). There were several functions that performed the same functionality, but they were implemented differently. For example, Array2DRowRealMatrix class and OpenMapRealMatrix class both have multiplication functions for matrices, but they are implemented in different ways. In such cases, both the functions are used in the code corpus. All the functions used in this study can be found via the following URL: https://github.com/MSU-STLab/MRPrediction/tree/master/alldotfiles

[2]http://scikit-learn.org/stable/
[3]https://commons.apache.org/proper/commons-math

Table 3: Best C and λ parameter of train model for each metamorphic relation

Metamorphic Relation	Best λ	Best C
Permutative	0.9	0.1, 1, 10, 100, 1000
Additive	0.9	0.1, 1, 10, 100, 1000
Multiplicative	0.9	0.1, 1, 10, 100, 1000

3.2 Metamorphic Relations

We manually identified three categories of MRs - Additive, Permutative, and Multiplicative, that are generally applicable to matrix calculations. These three high-level categories are further divided based on whether the modification is made at the element, row, or column levels. The full categorization of the MRs is shown in Table 1. In this work we only focus on predicting the high level MR category; i.e. Permutative, Additive and Multiplicative.

3.3 Evaluation Procedure

We use *train, validation and test method* to evaluate the MR prediction effectiveness. Table 2 shows the number of positive and negative instances for each MR; positive indicates that a function satisfies the given MR and negative indicates that the function does not satisfy the given MR. For each MR, we divided the data into three subsets, where each fold contained approximately the same portion of positive and negative instances, as the original dataset. The three folds were named as Train data, Test data, and Validation data. The precomputed kernel values of the functions in Train data were used to create the prediction model. The Validation data was used to select parameters for the predictive model. Those are- (1) regularization parameter C of the SVM (2) path weighing factor λ in the random walk kernel where $0 \leq \lambda < 1$.

The parameter values selected using the validation set were then used to create the predictive model for predicting the MRs for the test data. We repeated the train, validation and test method ten times so that the functions in each fold is selected randomly each time to avoid any biases occur in fold divisions.

We used the Area Under the receiver operating characteristic Curve (AUC) [8] to measure the prediction effectiveness and takes values ranged in [0, 1] [9]. AUC measures the probability that a randomly chosen negative example will have a lower prediction score than a randomly chosen positive example. AUC does not depend on the discrimination threshold of the classifier and has been shown to be a better measure for comparing learning algorithms [8]. A perfect classifier would have a AUC value of 1 [9]. A classifier that makes random predictions has an AUC value of 0.5. [9].

4 RESULTS AND DISCUSSION

Table 3 lists the λ and C values that recorded the highest AUC values for each MR on the validation set. For the three MRs here, the value selected for the parameter C doesn't seem to have a big effect on the prediction accuracy. But for all the three MRs, the best value for λ is 0.9. When the λ value is higher, the random walk kernel gives a higher weight to longer paths according to it's definition [10]. Therefore this indicates that longer paths in the

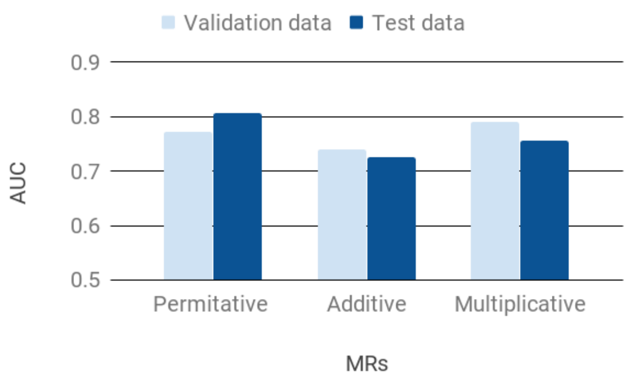


Figure 3: Prediction *AUC* score for *random graph kernel* for *validation dataset* and *test dataset*

CFGs are more important for predicting these MRs than the other paths.

Figure 3 shows the AUC scores for the validation data set and the test data set. On the test data, the highest AUC score (0.81) could be observed when predicting the Permutative MR. The other two MRs also reported AUC values higher than 0.7 indicating that our approach created effective predictive models for all the three MRs. Further, for all the three MRs, AUC values for the validation data set and the test data set is close. This indicates that there is a low chance of over-fitting in the predictive model.

5 RELATED WORK

Several previous studies have looked into automatically generating/predicting MRs. Kanewala *et. al* showed that, in previously unseen programs, MRs can be predicted using a machine learning method. Features were extracted from CFGs of the functions and they were then used to create a predictive model [9]. Later, they developed the graph kernel based approach used in this study to predict MRs for numerical programs [10].

Liu *et al.* introduced a new method called *Composition of Metamorphic Relation (CMR)*, where the generation of new metamorphic relations is done by combining existing metamorphic relations [11]. Dong *et. al* conducted a similar study, where *Compositional MR* was generated based on the speculative law of proposition logic [6].

Zhang *et al.* suggested a technique, where an algorithm searches for metamorphic relations in the form of linear or quadratic equations [16]. Su *et al.* also suggested a new method called *KABU*, which can be used to find more likely metamorphic relations by dynamically inferring the properties of the status of a method [12].

Chen *et al.* introduced an approach called *DESSERT*, where DividE-and-conquer methodology was used to identify the categorieS, choiceS, and choicE Relations for Test case generation [2]. Later, Chen *et al.* proposed a tool called *METRIC*, where metamorphic relations were identified with category-choice framework [3].

6 CONCLUSION & FUTURE WORK

The metamorphic testing technique is very useful to test programs that do not have a test oracle. The effectiveness of this technique highly depends on the set of MRs used for testing. But the identification process of MRs is mostly done manually.

This study is an extension of previous work, where the random walk kernel is used to predict MRs for functions that performs matrix calculation. Our results show that for these types of functions, random walk kernel can be effective in predicting MRs.

In the future, we plan to increase the number of functions used in this study. Further, new types of MRs, specifically for functions that perform matrix calculation, can also be considered. We also plan to extend the MR prediction scope beyond the function level.

ACKNOWLEDGMENTS

This work is supported by award number 1656877 from the National Science Foundation. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

REFERENCES

[1] F. E. Allen. 1970. Control Flow Analysis. In *Proceedings of a Symposium on Compiler Optimization*. ACM, New York, NY, USA, 1–19. https://doi.org/10.1145/800028.808479
[2] T. Y. Chen, P. L. Poon, S. F. Tang, and T. H. Tse. 2012. DESSERT: a DividE-and-conquer methodology for identifying categorieS, choiceS, and choicE Relations for Test case generation. *IEEE Transactions on Software Engineering* 38, 4 (July 2012), 794–809. https://doi.org/10.1109/TSE.2011.69
[3] T. Y. Chen, P. L. Poon, and X. Xie. 2016. METRIC: METamorphic Relation Identification based on the Category-choice framework. *The Journal of systems and software* 116 (2016), 177–190.
[4] T. Y. Chen, T. H. Tse, and Z. Zhou. 2003. Fault-based testing without the need of oracles. *Information and Software Technology* 45, 1 (2003), 1 – 9. https://doi.org/10.1016/S0950-5849(02)00129-5
[5] C. Cortes and V. Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (Sept. 1995), 273–297. https://doi.org/10.1023/A:1022627411411
[6] G. Dong, B. Xu, L. Chen, C. Nie, and L. Wang. 2008. Case studies on testing with compositional metamorphic relations. 24 (12 2008), 437–443.
[7] L. Hardesty. 2013. Explained: Matrices. (Dec 2013). http://news.mit.edu/2013/explained-matrices-1206
[8] J. Huang and C. X. Ling. 2005. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering* 17, 3 (March 2005), 299–310. https://doi.org/10.1109/TKDE.2005.50
[9] U. Kanewala and J. M. Bieman. 2013. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 1–10. https://doi.org/10.1109/ISSRE.2013.6698899
[10] U. Kanewala, J. M. Bieman, and A. Ben-Hur. 2016. Predicting Metamorphic Relations for Testing Scientific Software: A Machine Learning Approach Using Graph Kernels. *Softw. Test. Verif. Reliab.* 26, 3 (May 2016), 245–269. https://doi.org/10.1002/stvr.1594
[11] H. Liu, X. Liu, and T. Y. Chen. 2012. A New Method for Constructing Metamorphic Relations. In *2012 12th International Conference on Quality Software*. 59–68. https://doi.org/10.1109/QSIC.2012.10
[12] F. H. Su, J. Bell, C. Murphy, and G. Kaiser. 2015. Dynamic Inference of Likely Metamorphic Properties to Support Differential Testing. In *Proceedings of the 10th International Workshop on Automation of Software Test (AST '15)*. IEEE Press, Piscataway, NJ, USA, 55–59. http://dl.acm.org/citation.cfm?id=2819261.2819279
[13] R. Vallee-Rai and L. J. Hendren. 1998. Jimple: Simplifying Java Bytecode for Analyses and Transformations. (1998).
[14] E. Weyuker. 1982. On Testing Non-Testable Programs. 25 (11 1982).
[15] T Y. Chen, S C. Cheung, and S. M. Yiu. 1998. Metamorphic testing: a new approach for generating next test cases. (01 1998).
[16] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei. 2014. Search-based Inference of Polynomial Metamorphic Relations. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. ACM, New York, NY, USA, 701–712. https://doi.org/10.1145/2642937.2642994