A Machine Learning Approach for Developing Test Oracles for Testing Scientific Software

Junhua Ding
Department of Computer Science
East Carolina University
Greenville, NC, USA
dingj@ecu.edu

Dongmei Zhang
School of Computer Science
China University of Geosciences
Wuhan, China
jjilee@163.com

Abstract—**Absence of test oracles is the grand challenge for testing complex scientific software. Metamorphic testing is the novel technique for developing test oracles on metamorphic relations. Although it is easy to find metamorphic relations based on general guidelines and domain knowledge, the ones that can adequately test the software are difficult to be developed. This paper introduces a machine learning approach for iteratively developing metamorphic relations. The approach develops initial metamorphic relations and tests first, and then the relations and tests are refined through mining the initial test execution and evaluation results with machine learning algorithms. The approach and its effectiveness are illustrated through testing an open source discrete dipole approximation program.**

Keywords-metamorphic testing, metamorphic relation, test oracle, scientific software, machine learning.

I. INTRODUCTION

Scientific software is the software that includes computational components for supporting scientific investigation and decision making [8]. The examples of scientific software include simulation software of nuclear reactions, software for predicting and tracking hurricanes, and software for analyzing medical images. Testing scientific software faces many challenges. Many of them are "non-testable" due to the absence of test oracles [2]. Oracle problems are key to solve for adequately testing scientific software [8]. Metamorphic testing [2] as a novel software testing technique is a promising approach for solving oracle problems. It creates tests according to metamorphic relationship (MR) and verifies the predictable relations among the actual outputs of the related tests. It was first proposed by Chen [2] for addressing oracle problems, and it has been applied to several domains such as bioinformatics systems, machine learning systems, compilers, and scientific software [16]. However, the application of metamorphic testing to large scale of scientific software is rare because of the difficulty of the identification of MRs.

Existing testing approaches normally check the correctness of each individual execution, but not the relation among outputs of multiple executions. The success of metamorphic testing tells us that checking the relation among multiple test outputs is necessary for testing scientific software that is hard to find a test oracle. However, the quality of metamorphic testing is highly depended on MRs. Due to the grand challenge to develop strong MRs, many MRs used for testing a complex software system are relatively too weak to ensure the testing quality. It is important to define a set of criteria for evaluating the adequacy of MRs. Then the test execution and evaluation results shall be used for guiding the generation of adequate MRs [4]. The approach introduced in this paper includes a framework for the development of MRs and tests, and a strategy for refining the relations and tests though mining the test execution and evaluation results with machine learning algorithms. The test evaluation consists of test coverage evaluation and mutation testing. The mutation testing includes two steps: the first one is applied to the mutated metamorphic relations, and the second one is applied to the mutated program. The proposed approach is an enhancement of metamorphic testing in the development of test oracles for testing "non-testable" scientific software. A case study of testing an open source discrete dipole approximation (DDA) program called ADDA[19][21] is conducted to explain the approach and to demonstrate its effectiveness.

DOI reference number: 10.18293/SEKE2016-137

ADDA is a fairly complex scientific software system with many characteristics that cause the testing challenges that were described by Kanewala and Bieman [8]. The most challenge issue for testing ADDA is the absence of test oracles since a test input of ADDA may include thousands of parameters and it is impossible to know the correctness of the output for an arbitrary input. ADDA as open source software, its source code, manual, and other documents are available online. Anyone who is interested in the technique discussed in this paper has the opportunity to reproduce the experiment. The experience from testing ADDA can be easily shared in the software testing community. The iterative metamorphic testing for developing test oracles with machine learning for adequately testing ADDA can be easily extended for testing other scientific software.

The rest of this paper is organized as follows: Section 2 describes the general idea of the proposed approach. Section 3 discusses iteratively developing test oracles for testing ADDA. Section 4 describes related work, and section 5 concludes this paper.

II. ITERATIVE METAMORPHIC TESTING

In iterative metamorphic testing, MRs serve as test oracles, and test coverage evaluation and mutation testing are used for evaluating the test adequacy and iteratively producing adequate MRs for testing the Software Under Test (SUT).

A. *The Approach*

The iterative metamorphic testing consists of three major steps: development of initial MRs and tests, test evaluation, and refinement of MRs. (1) *Development of initial MRs and tests*. Based on domain knowledge of the SUT and general framework of metamorphic testing *[13]*, one develops a set of MRs. Based on general test generation strategies such as combinatorial technque, category-based or random approach to produce tests for each MR. (2) *Test evaluation*. As soon as the SUT passes all tests, tests created for mutations of MRs are used for checking the quality of MRs. A mutation test is a set of valid tests whose outputs violate an MR. The purpose of testing SUT with mutation tests is to ensure the relation can differentiate positive tests from negative ones. The effectiveness of the test is then evaluated with selected test coverage criteria and mutation testing. A mutant should be killed by an MR or weakly killed by a test. A mutant is weakly killed when the output of the mutated program is different to the predicted one based on refined MRs. (3). *Refinement of MRs*. It is the process for creating test oracles, which can be developed through refining current MRs or defining new MRs. Machine learning algorithms could be used for processing existing test execution data and test evaluation results to find patterns for severing MRs.

B. *A Running Example*

This section describes the proposed approach with a running example: testing a program that is used for calculating the contrast of an image, which is 101 pixels * 101 pixels in gray scale with 8-bit resolution like those shown in Fig. 1. We define the contrast based on the average intensities of a concentric ring area and a circle area in an image. The intensity of each pixel is the pixel value such as 125 or 24 of an 8-bit resolution image. The contrast C of an image is defined as follows:

$$C = \frac{\overline{R_c} - \overline{R_p}}{\overline{R_c} + \overline{R_p}} \qquad (1)$$

where $\overline{R}_c$ is the image intensity averaged over a circle area of 18 pixels diameter centered at the origin and $\overline{R}_p$ is the intensity averaged over a concentric ring area in the region with 30 and 66 pixels as the inner and outer diameters, respectively. There is no easy way to decide the correctness of the calculated contrast of an image, which is exactly the purpose of building the program. Iterative metamorphic testing can be used for testing the program.

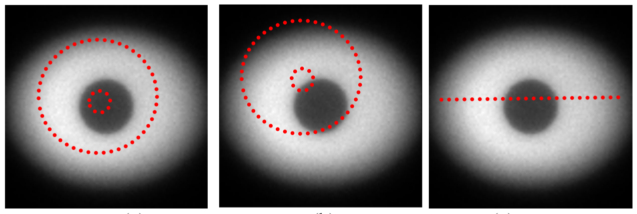

Figure 1: (a) The test outputs (the red dots in the ring) form a closed circle; (b) the test outputs (the red dots within the ring) form a half circle within the concentric ring, which violates circle relation MR1. (c) The tests cannot detect the error in the program.

1) *Devloping Initial MRs and Tests.*

Based on formula (1), it is easy to define an MR like *increase/decrease* as follows:

MR1 (Increase/Decrease)*: Given an image, increase the intensity of any pixel in the circle area will increase the contrast of the image, and increase the intensity of any pixel in the concentric ring area will decrease the contrast of the image.*

Although MR1 is useful for testing the program, it isn't good enough. For example, MR1 cannot verify whether a pixel is correctly calculated for the concentric ring or the circle area. If the concentric ring or the circle area is incorrectly implemented, it is difficult to detect the defect using the tests based on MR1 since only very few special tests will violate the MR. For example, if the program calculates the circle area using diameter 17 instead of 18, then only tests that are located on the boundary of diameter 18 may detect the error. If the origin position of the circle is set to position <50, 51> instead of <50, 50>, it is almost impossible to find a test to detect the error without checking the relation among outputs of multiple tests. Therefore, it is necessary to add new MRs for checking the implementation of the concentric ring area and the circle area in the program, the relations for checking the concentric ring area and the circle area are used for guiding the selection of tests and checking the testing results. We know the center of the image is at position at <50, 50>, and then the test oracles for deciding whether a pixel <*x, y*> is within the circle area or within the concentric ring area are defined as follows:

Test oracle 1 (a pixel is within a circle)*: pixel <x,y> is within the circle if* $\sqrt{(x-50)^2+(y-50)^2} < 18/2$.

Test oracle 2 (a pixel is within the concentric ring)*: pixel <x, y> is within the concentric ring area if* $(30/2)^2 < \sqrt{(x-50)^2+(y-50)^2} < 66/2$.

A test for checking the implementation of the areas is a pixel <*x, y*> in an image, and it is marked with a special color when the pixel is counted for an area during testing. Each MR1 test is a candidate for creating new tests based on new MRs for metamorphic testing. The relation among the outputs, which are colored pixels, can be visually verified. MR2 is defined as follows:

MR2 (circle): *Given a set of test inputs* $t_1 = \{<x_i, y_i>| \sqrt{(x_i-50)^2+(y_i-50)^2}=32\}$*, the outputs shall form a circle that is within the concentric ring area and has the same origin as the ring. Given a set of test inputs* $t_2 = \{<x_i, y_i>| \sqrt{(x_i-50)^2+(y_i-50)^2}=8\}$*, the outputs shall form a circle that is within the circle area and has the same origin as the circle.*

t_1 and t_2 form a circle in the circle area and a circle in the concentric ring area, respectively, and their outputs should form a colored circle in the circle area and one in the concentric ring area.

2) *Test Evaluation*

As soon as the program passes all tests, it is further tested with mutation tests. Testing with mutation tests is a type of negative testing to determine the response of the system with unexpected test inputs. The purpose of testing with mutation tests in this research is to check the quality of MRs and mitigate the problem that a relation is so weak that can be satisfied by any tests. A mutation test is a set of valid tests whose outputs don't satisfy an MR. A mutation test t'_1 of MR2 is defined as follows:

Mutation test t'_1: $\{<x_i, y_i> | ((x_i - 50)/40)^2 + ((y_i - 50)/20)^2 = 1\}$

The outputs of test set t'_1 don't satisfy relation MR2 since they form an oval instead of a circle, which shows relation *circle* is a carefully selected relation and only satisfied by a carefully selected test sets. Using the same idea, one can create a mutation test for MR1. The test includes pixels outside of the areas, and changing of the intensity of the pixels would not affect the contrast.

Test adequacy is evaluated through measuring selected test coverage criteria such as function coverage, condition coverage and mutations [7]. It is infeasible to kill all mutants, and it is even more challenge to kill mutants when testing "non-testable" programs. Many mutants such as one systematically shifts the real values cannot be killed by simple MRs, which is also a reason of checking MRs with mutation tests. If the correctness of an individual output can be checked, then regular mutation testing [7] can used for testing the program. Otherwise, we expect each mutant will be at least weakly killed. A good MR needs kill some mutants. If a mutant was not killed or weakly killed by any test, then new tests or new MRs should be developed until the mutant is killed or a conclusion of the infeasibility of killing the mutant is made. If all mutants are killed or weakly killed, it is necessary to check whether these mutants are killed uniformly by the MRs.

3) Refining MRs

When a pixel $<x, y>$ in test t_1 or t_2 is tested, the program marks the pixel with a different color. As soon as all pixels are tested, it is easy to observe whether the outputs of t_1 or t_2 form a circle within the expected area. For example, if the concentric ring is implemented shifted from the center origin <50, 50> to <55, 50>, then we can find some colored pixels are shifted from the center of the concentric ring area, and the colored pixels form an incomplete circle within the concentric ring area, as shown in Fig. 1 (b). If the test was created without considering the relation, such as those shown in Fig. 1 (c), the shifting error would not be detected since each individual output is correct. For the same reason, relation *circle* is also useful to detect the problem in test oracles. For example, if test oracle 2 incorrectly uses position <55, 50> as the origin of the image, the problem can be easily detected by either test inputs t_1 or t_2.

In this research, the initial relations are revised and refined during test process and in turn the updated MRs are used for producing more tests. Let's use MR *increase* as an example to explain the refinement process. We select one image and change the intensity of some pixels in the concentric ring or circle areas of the image to create a new image. In the beginning, no one knows the exact difference of the contrast between the original image and the modified one except relation MR1 of the contrasts between the two images. But as soon as the program calculates the exact numbers of pixels and their intensities in the concentric ring and circle areas, it is easy to calculate the exact difference between the contrasts of two images that have different intensity for exact one pixel so that MR *increase* is refined from a general relation to a precise one. For example, if a pixel's intensity in the circle area is increased from 64 to 128, and we know there are n pixels with total intensity I_c in the circle area based on previous execution, one can use new $\overline{R_c} = (I_c + 64)/n$ to calculate the exact contrast of the new image using formula 1. The MR *increase* is refined with exact values. The refined relations are used for creating more tests for testing the program, and each individual output can be precisely verified. In order to develop MRs for complex scientific software, machine learning of the test execution data and evaluation results should be used.

III. TESTING ADDA USING METAMORPHIC TESTING

In this section, we discuss testing ADDA using the iterative metamorphic testing, especially on refinement and development of MRs using machine learning approaches.

A. Testing ADDA

DDA is a method to simulate light scattering from particles through calculating scattering and absorption of electromagnetic waves by particles of arbitrary geometry [19]. The particle as a dielectric scatterer are divided into many small volumes called dipoles. The dipoles make up a small cubic lattices of spacing within a fraction of wavelength of the incident of light and they are each exposed to the incident field and the field due to all other dipoles. The interactions of dipoles are approximated based on equations of the electric field [19]. DDA has been widely used for light scattering simulations [19]. ADDA is an open source implementation of DDA written in C99 with routines in Fortran and C++ [21]. The general input parameters of ADDA define the optical and geometry properties of a scatterer/particle including the shape, size, diploes, refractive index of each dipole, orientation of the scatterer, definition of incident beam, and many others. ADDA produce different outputs for different applications such as Muller matrix at different scattering angles used for producing diffraction images. It is unknown the correctness of an ADDA simulation with an arbitrary heterogeneous scatterer with an arbitrary shape in advance. The authors of ADDA have conducted extensive testing of ADDA for special cases, but more general tests of ADDA is still necessary. Here we discussed how to test ADDA using iterative metamorphic testing, particularly on developing MR using a machine learning approach.

B. Development of Initial MRs and Tests

ADDA has been extensively tested with special cases [14][21], which serve as the initial tests for creating tests using MRs. Since an output of ADDA may include thousands items and its input may include lots of parameters, it is very difficult to find an MR directly on its inputs and outputs. We define MRs on the textual property of the the diffraction image generated from an ADDA output. The textual patterns of diffraction images can be visually observed or quantitatively characterized with Grey Level Co-occurrence Matrix (GLCM) features [6]. GLCM calculates computable textural features based on grey-tone spatial dependencies. It defines how often different combinations of gray level pixels occur in an image for a given distance d in a particular angle θ [6]. We define initial MRs based on the shape, size, orientation, and refractive index of a scatterer. Each MR checks the correlation between a parameter and the textual pattern of the ADDA calculated diffraction image of a scatterer.

MR3 (**Difference**): *When the size, shape, orientation, refractive index value of a scatterer is changed, its textual pattern of the diffraction image is changed. Only one parameter is changed at each time, and the change of the orientation doesn't affect the textual pattern of a sphere scatterer.*

Since the ADDA simulation results of sphere scatterers have been compared to the results calculated from Mie theory with many different configurations [14]. We assume the calculation of ADDA on sphere scatterers is correct. For testing each MR, we first compare the result to a sphere scatterer result. Fig. 2 shows the comparison of the textual patters of three scatterers in different shapes, and the result satisfies MR3.

We use combinatorial technique to create tests to cover more scenarios. For example, the four input parameters are the *scatterer size*, *shape*, *refractive index*, and *orientation*. Then select the possible values for each parameter guided by techniques such as category based technique, random, or boundary values. The base tests can be created using combinatorial techniques such as pairwise, and then the MRs are used for producing tests based on the base tests. For example, the possible values of size are {3μm, 5μm, …, 16μm} for the ADDA study, shapes are {*sphere*, *ellipsoid*, *bi-sphere*, *prism*, *egg*, *cylinder*, *capsule*, *box*, *coated*, *cell1*, *cell2*, …}, orientations are {<0, 0, 0>, <10, 90, 0>, <270, 0, 0> , …}, and refractive index values are {1.0, … 1.5}. Using pairwise, one can create many base tests, and then select the valid tests as the first tests to create MR tests for MR3. Fig. 3 shows a comparison of the textual patters of diffraction images of a cell with different refractive index values in nuclear. The result satisfies MR3.

Although MR3 can offer some preliminary testing for ADDA for heterogeneous scatterers including real cells, the relation is still too weak to adequately test ADDA. MR3 should be refined and probably new MRs should be created for testing the program.

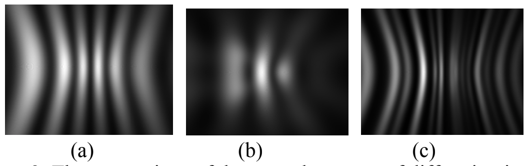

Figure 2. The comparison of the textual patterns of diffraction images of scatterers in different shapes (a) sphere, (b) ellipsoid, and (c) bi-sphere.

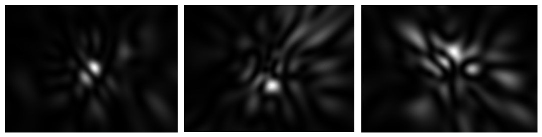

Figure 3. The comparison of the textual patterns of diffraction images of a cell with different refractive index values in nuclear.

C. Evaluation of MRs and Tests

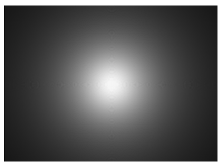

Figure 4. An error diffraction image.

The initial tests created using MR3 covered 100% statements and close to 100% of conditions. Partial mutation testing was conducted to check the effectiveness of the metamorphic testing. The mutation testing was applied only to one module of ADDA program. Several mutants were instrumented to the code manually and their results were manually inspected.

Mutation testing of "non-testable" programs like ADDA is difficult since many MRs are not sensitive to mutants. In this case, one can check the consistency between the outputs of the mutated program and the original one. A mutant is killed if it causes a violation of any MR, and a mutant is weakly killed if the output of a test of the mutated program is different to the original one. Absolute Value Insertion (ABS) and Relational Operator Replacement (ROR) were used for creating mutants. Among total all 20 mutants (10 ABS mutants, and 10 ROR mutants) we created for testing ADDA, 17 of them were killed by crashing of the program or exception handing. The other 3 mutants were killed by the MRs since the outputs didn't generate any textual pattern as shown in Fig. 4. We found mutants are easily killed by simple tests in scientific software probably due to the complexity of the software. A slight change of the program can cause a catastrophic error in the calculation. The mutant that instrumented to ADDA program produced the diffraction image shown in Fig. 4 is easily to be killed since the two different scatterers produced the diffraction images with the same textual patterns. The results show that MR3 can test ADDA in some degree.

However, it is very difficult to create a mutation test for MR3 that would produce two diffraction images with the same textual patterns by changing the input parameters. MR3 is so weak that many wrong results can satisfy it.

D. Refinement of MRs

According to the results discussed in Section III. B, C, it is easy to see the intuitive relation between the textual pattern of a diffraction image and its 3D morphology. We need refine the relation for understanding how the change of 3D morphology parameters including the shape, size, refractive index and orientation are precisely related to the textual pattern. We investigate the problem via an experimental study. First, we took many diffraction images of different types of cells using a diffraction image based flow cytometer called p-DIFC. These images are called measured diffraction images to compare the calculated diffraction images produced from ADDA. Second, process these measured images for GLCM, and check how the GLCM features are related to cell types. Select optimal GLCM features for cell classification, build a feature vector including labeled cell type for each image, and construct a feature vector matrix with the images that are belonged to the same type of cells for training a SVM. Third, use the trained SVM to classify diffraction images for cell types based GLCM feature values. If the cells can be successfully

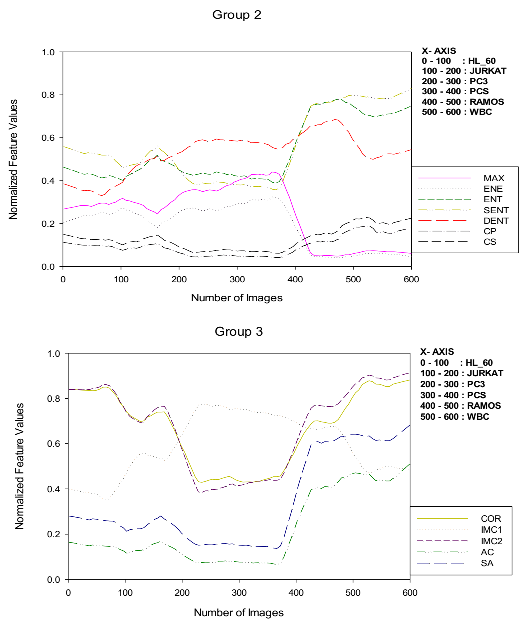

Figure 5. GLCM features of 6 types of diffraction images

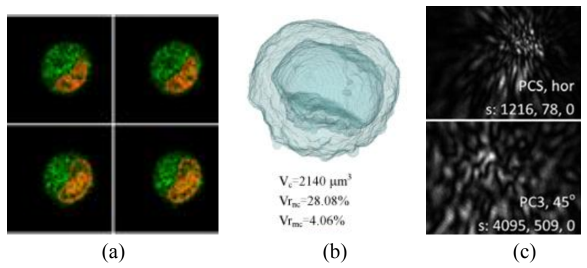

Figure 6. Confocal images (a), 3D structure (b) and diffraction images (c).

classified by the GLCM features, then we need test whether the calculated diffraction images can be also used for classifying the cell type. If it does, the cell classification based on diffraction images will serve as a test oracle for testing ADDA.

We took 100 diffraction images for each type of cells for total 6 different types of cells using p-DIFC. Fig. 5 is the experimental result of selected GLCM features among 600 cells. From the two diagrams, it is not difficult to see that the correlation between some GLCM features and the cell types. However, the cell classification cannot be completed just based on one feature, instead multiple features have to be used. In this study, feature selected was conducted, and 8 GLCM features were selected for the SVM based classification [17]. Fig. 5 should be able to serve as a test oracle for testing ADDA since we expect the ADDA calculated diffraction images of the same type of cells in Fig. 5 should also have the same feature patterns as those shown in Fig. 5. However, the real reflective index value of each cell organelle is unknown but a guessed one, the feature patterns between the measured images and calculated images should be similar but not identical, and the precise relation between them cannot be defined. Fig. 5 can serve as a reference for testing ADDA, but it is not enough.

The SVM based classifier is built on Weka with SVM library LIBSVM. Stratified 10-Fold Cross Validation (10FCV) was used for checking the accuracy of the SVM classification. In this study, specifically 90 images per each type of cells were used for training the SVM and remaining 10 images were used for testing the classifier. We experimented the classification using different sets of GLCM features and different distance and grey level for calculating GLCM. The best performance of the SVM classification is configured with 8 selected GLCM features, and the GLCM calculation was configured with distance as 2 and grey level as 64. The average accuracy of the classification of the 6 types of cells with total 600 diffraction images is 91.16% [17]. Based on the experimental result, we expected the calculated diffraction images should have the same property as the measured images, which can classify cell types based on the GLCM features. However, we cannot use the SVM classifier trained with the measured diffraction images for classifying the calculated images, which has to be trained using the calculated diffraction images.

In order to calculate a diffraction image of a cell using ADDA, we need to construct the 3D structure of the cell and then assign the refractive index values for each voxel of the cell. A stack of confocal image sections are taken using a confocal microscope, and then each image section is processed to segment the cell components such as nuclear and mitochondria. The 3D structure of the cell is built based on the processed image sections, and a refractive index value is assigned to each voxel in the structure. The 3D structure then is imported to ADDA for the simulation, and a diffraction image is generated from the simulation result. Since orientation is one of the input parameters to ADDA, but cell type doesn't have any relation to the orientation. Therefore, we use ADDA to calculate diffraction images in many different orientations for each cell. Fig. 6 shows 4 of total 48 confocal image sections of a cell, its 3D structure and calculated diffraction images in two different orientations.

We took confocal image for 30 cells each type of cells, and 4 types of cells were used in the experiment. The 3D structure of each cell was simulated for 25 different orientations in ADDA. Therefore, we calculated 750 diffraction images for each type of cells. The images are processed for GLCM features with distance 2 and grey level 64, and a feature vector matrix including the feature values and labeled cell type is built for the same type of images. Finally, the feature vector matrix is used for training the SVM and 10FCV is used for checking the accuracy of the classification. Our preliminary result has shown the cells can be classified with the accuracy as high as the measured diffraction images. However, we haven't completed the experiment for all 4 type cells. Based on the experimental result, a new MR is developed:

MR4 (**Cell classification**): *A calculated diffraction image of a cell can be correctly classified into a classification by a SVM classifier. If two calculated diffraction images produced from two different types of cells, an SVM classifier can classify each into the correct classification.*

The new MR built based SVM classifier can test the program using existing data. The data size used for this study is very limited, it is still not clear whether the approach can be extended to all different types of cells or even different scatterers. However, the approach should be very useful in general for developing test oracles for testing scientific software that is absent of test oracles.

IV. Related Work

One of the greatest challenges for testing scientific software is due to the oracle problem [8][1]. Metamorphic testing is the technique for addressing the oracle problem though developing them with MRs [2][16]. It has been applied to several domains such as bioinformatics systems, machine learning systems, and online service systems [10] [11][20]. Murphy et al. classified six types of MRs for testing machine-learning systems [15], which are useful for creating initial MRs. Xie *et al.* investigated some specific MRs that were extended from general MRs for testing machine learning applications [18]. Guderlei and Mayer proposed a statistical metamorphic testing in [5], where two or more output sequences were generated and compared according to the statistical MRs. The similar idea is used in this paper for developing test oracles based on cell classification. Metamorphic testing has been also used for testing scientific software. For example, Mayer and Guderlei developed a group of MRs for testing image process programs [13]. Chen, Fend and Tse have applied metamorphic testing for testing partial differential equations [3]. However, the program they tested were much smaller and simpler than ADDA. Knewala, Bieman and Ben-Hur recently reported a result on the development of MRs for scientific software using machine learning approach integrated with data flow and control flow information [9]. However, whether the result can be extended for testing large scientific software is unclear.

V. Summary

In this paper, we introduced an iterative metamorphic testing technique for testing scientific software, where MRs are iteratively developed based on analyzing test execution and evaluation results. We illustrated the approach and its effectiveness through testing ADDA, the widely used open source scientific software. A new MR was developed through analyzing experimental data using machine learning algorithm SVM. The approach could be useful for testing similar scientific software as well as other software systems that are absent of test oracles.

Acknowledgment

The authors would thank Dr. Xin-Hua Hu, Eric King at East Carolina University for assistances of the experiments. This research is supported in part by grant CNS-1262933 and CNS-1560037 from the National Science Foundation.

References

[1] E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, "The Oracle Problem in Software Testing: A Survey," IEEE Trans. on Software Engineering, , vol.41(5), pp.507-525, 2015.

[2] T. Y. Chen, S. C. Cheung, and S. Yiu, "Metamorphic testing: a new approach for generating next test cases", Tech. Rep. HKUST-CS98-01, Dept. of Computer Science, Hong Kong Univ. of Science and Technology, 1998.

[3] T. Y. Chen, J. Feng and T. H. Tse, "Metamorphic testing of programs on partial differential equations: a case study," COMPSAC 2002. pp. 327-333.

[4] J. Ding, T. Wu, J. Q. Lu, X. Hu, "Self-Checked Metamorphic Testing of an Image Processing Program," 4th Intl. Conf. on Security Software Integration and Reliability Improvement, Singapore, 2010.

[5] R. Guderlei, and J. Mayer, "Statistical metamorphic testing - testing programs with random output by means of statistical hypothesis tests and metamorphic testing", in Proc. of the 7th ICQS. pp. 404-409, 2007.

[6] R. M. Haralick, K. Shanmugan, and I. H. Dinstein, "Textural features for image classification", IEEE Trans. Syst., Man, Cybern., vol. SMC-3, pp.610 -621, 1973.

[7] Y. Jia; M. Harman, "An Analysis and Survey of the Development of Mutation Testing," IEEE TSE, vol.37, no.5, pp.649-678, 2011.

[8] U. Kanewala, J. M. Bieman, "Testing scientific software: A systematic literature review", Information and Software Technology, Vol. 56, Issue 10, Oct. 2014, pp. 1219-1232, 2014.

[9] U. Kanewala, J. M. Bieman, A. Ben-Hur, "Predicting Metamorphic Relations for Testing Scientific Software: A Machine Learning Approach Using Graph Kernels", Journal of Software Testing, Verification and Reliability, Nov. 16, 2015, DOI: 10.1002/stvr.1594.

[10] V. Le, M. Afshari, and Z. Su. "Compiler validation via equivalence modulo inputs". In Proceedings of the 35th ACM SIGPLAN Conference on PLDI '14. pp.216-226. 2014.

[11] M. Lindvall, D. Ganesan, R. Árdal, and R. E. Wiegand. "Metamorphic model-based testing applied on NASA DAT: an experience report". Proc. of the 37th ICSE, Vol. 2. pp. 129-138. 2015.

[12] H. Liu, F. Kuo, D. Towey, T.Y. Chen, "How Effectively Does Metamorphic Testing Alleviate the Oracle Problem?" IEEE Trans. on Software Engineering, vol.40, no.1, pp.4,22, Jan. 2014.

[13] J. Mayer, and R. Guderlei, "An empirical study on the selection of good metamorphic relations", In proc of 30th COMPSAC, pp. 475-484, 2006.

[14] M. Moran, "Correlating the morphological and light scattering properties of biological cells", PhD dissertation, department of physics, East Carolina University, 2013.

[15] C. Murphy, G. Kaiser, L. Hu, and L. Wu. "Properties of machine learning applications for use in metamorphic testing". In Proc. of the 20th SEKE, pp. 867–872, 2008.

[16] S. Segura; G. Fraser; A. Sanchez; A. Ruiz-Cortes, "A Survey on Metamorphic Testing," in IEEE Trans. on Software Engineering , vol.PP(no. 99), doi: 10.1109/TSE.2016.2532875. 2016.

[17] S. K. Thati, J. Ding, D. Zhang, and X. Hu, "Feature Selection and Analysis of Diffraction Images", the 4th IEEE Intl. Workshop on Information Assurance, Vancouver, Canada, August 3-5, 2015.

[18] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers". 9th Intl. QSIC '09, pp. 135 – 144, 2009.

[19] M.A. Yurkin and A.G. Hoekstra, "User manual for the discrete dipole approximation code ADDA 1.3b4", http://a-dda.googlecode.com/svn/trunk/doc/manual.pdf (2014). Last accessed on March 25, 2016.

[20] Z. Zhou, S. Xiang, T.Y. Chen, "Metamorphic Testing for Software Quality Assessment: A Study of Search Engines", IEEE Trans. on Software Engineering, doi:10.1109/TSE.2015.2478001, 2015.

[21] ADDA project, https://code.google.com/p/a-dda/, Last accessed on March 12, 2016.