International Journal of Advance Research in Science and Engineering
Volume No.06, Special Issue No.(01), Nov 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

A Research on Generation of Metamorphic Relations And Automatic Test Cases Generation Using Nature-Inspired Optimization Algorithms For Integer Bug Detection

Simarjeet Kaur[1], Dr. Deepak Kumar[2]

[1] *Assistant Professor, Guru Kashi University,Talwandi Sabo (Bathinda)*

[2] *Assistant Professor, Guru Kashi University,Talwandi Sabo (Bathinda)*

ABSTRACT

Metamorphic testing is a new and innovative technique, which is used to determine if a test execution uncovers a fault. This is a more practical method than the test oracle method. Oracle is the big issue in testing that only compares the generated output to the predicted output It used as a component for figuring out results either successful or not. By contrast, metamorphic testing applies a modification to a test input and utilizes metamorphic relations. It then observes how the program output changes into a different one as a result metamorphic testing changes the way that software is tested the faults. This paper illustrate about the overview about metamorphic testing, various benefits of metamorphic testing technique with contrast to another techniques of software testing and detection of integer bugs with metamorphic testing by the generations of new relations and overcome various challenges regarding metamorphic testing.

Keywords*: Metamorphic testing, Oracle problem, Metamorphic relation, AFSA*

I. INTRODUCTION

In order to find fault within a test execution, often, a test oracle is used. The methodology utilized involves comparing the expected output against the observed results. The test oracle method is not practical in cases where the relationship between the program's input and output is complex. In such cases, the more fitting methodology involves metamorphic relationships. This methodology relies on transforming the input of a program and observing how the output of that same program morphs as a result. This article will discuss metamorphic testing, include it's origins, applications, and the challenges faced by this method [1].Integer variables are expressed by fixed bit- wide vector. Integer overflow takes place when the value attained by instruction operation is more than the value of storage capacity. Such errors can be attributed to the integer bug, which is one of the main reasons behind software calculation error. This can be problematic and even fatal in applications such as rocket launching. At times, software security vulnerabilities are produced when an integer is disposed to be an unexpected value by a program and this unexpected value is then used for the array indexes or loop variable. Due to expense. integer overflow in commercial software has not yet been detected on a large scale. Metamorphic testing was first introduced in a technical report by Chen et al. [3] that was published in 1998.It must be mentioned though that earlier reports describe the use of identity relations to check program outputs [4], [5] as well as fault tolerance [6].In total, the paper found 12 different application fields. Web

services and applications (16%) followed by computer graphics (12%),simulation and modelling (12%) and embedded systems (10%) are the most renowned domains.IT also identified applications to other areas (21%)like financial software, optimization programs or encryption programs.[2].

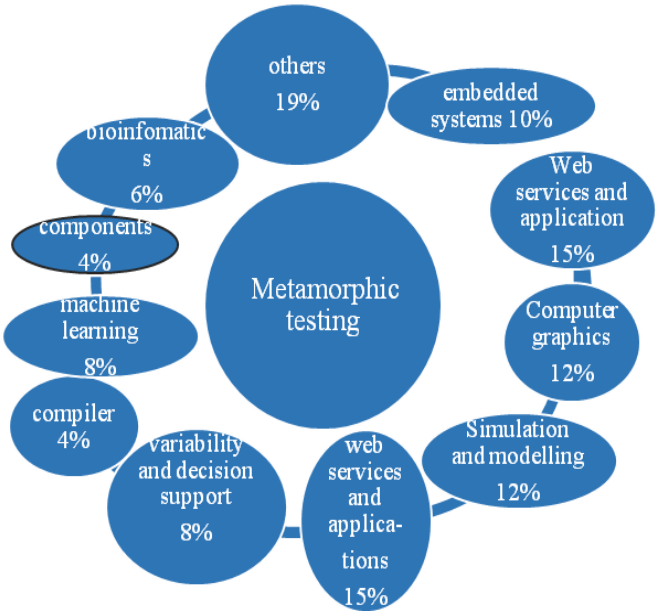


Fig 1. 1 Metamorphic testing application domains [2]

1.1. PROBLEM STATEMENT

It is impossible to test software with all the conceivable inputs. Successful test cases do not reveal error, so these test cases are not considered and discarded by testers. But these successful test cases do carry useful information which remains buried and used. One more limitation of Software testing is the oracle problem. The test oracle has always been restricting the development of software testing. In some cases the test oracle does not exists or if it exists, it is too complex to be used.

1.2. Objectives

The main goal of this research paper is:

- To study the various approaches of Metamorphic Testing.
- To generate various metamorphic relations.
- To propose and implement Metamorphic Testing with Optimizing Algorithm.
- To calculate the Mutation Detection Ratio and compare the results.
- To Overcome the challenges of Metamorphic Testing.

II. LITERATURE REVIEW

We've read several academic papers on software testing in the past, and as a general rule I really don't like them. However, this one's really worth reading. In the article the authors suggest the use of a new software testing technique called metamorphic testing to finding integer bugs in the software code. I've generated the universal metamorphic relations and it's some of the most difficult to formulate MR I've ever done.After some

discussion about why traditional techniques aren't enough, the authors describe their new method of metamorphic testing (MT): Instead of using the traditional test oracle, MT uses some problem domain specific properties, namely metamorphic relations (MRs), to verify the testing outputs. The end users, together with the testers or program developers, first need to identify some properties of the software under test. Then, MRs can be derived according to these properties. The article then reviews a couple of studies and talks about the limitations of the technique.W. K. Chan et al[7] describes testing the correctness of services assures the functional quality of service oriented applications. A service oriented application may bind dynamically to its supportive services. For the same service interface, the supportive services may behave differently.E.J. Weyuker [8]frequently invoked assumption in program testing is that there is an oracle (i.e. the tester or an external mechanism can accurately decide whether or not the output produced by a program is correct).T song Yueh Chen:[9]Summaries what testing techniques have been successfully integrated with metamorphic testing.Barr et al[10]describes Testing involves examining the behaviour of a system in order to discover potential faults.

III. RESEARCH METHODOLOGY

Here the flow of our work in described in steps.

Step 1.we test code by giving the function with mutant and without mutant and test that gave the same result. This shows that fault did not detected. We found that the bugs could not be detected. The fault is not detected by the testing it give same results with and without mutant. There is a fault in area1() but it give same value for both the functions. This problem will be solved by applying the metamorphic testing. For this detection we generate the universal metamorphic relations. In our research we take the mathematical formulas of triangle.

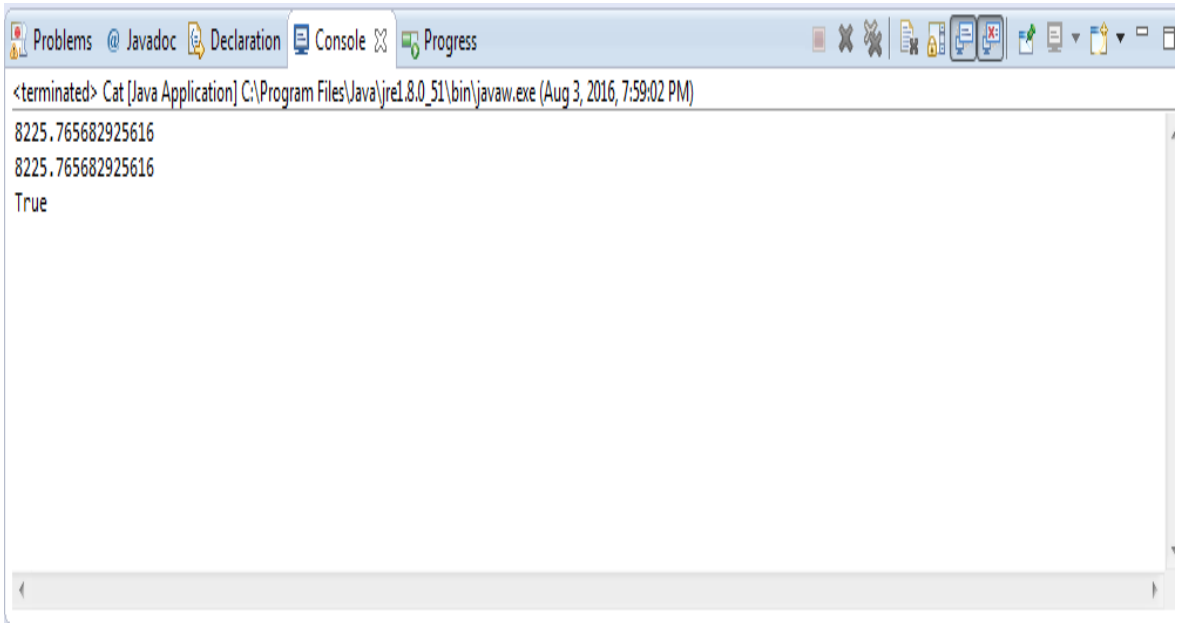


Fig 3.1 Output with mutant

International Journal of Advance Research in Science and Engineering
Volume No.06, Special Issue No.(01), Nov 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

Step 2.Then we generate 7 new relations.by which we can test our code. The flow of work is represented in flow chart form.

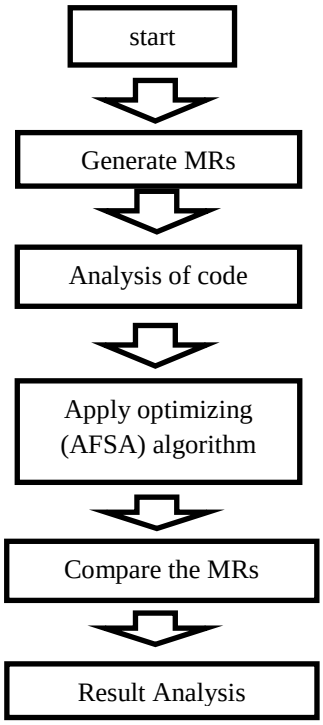


Fig 3.2 Flow chart of work

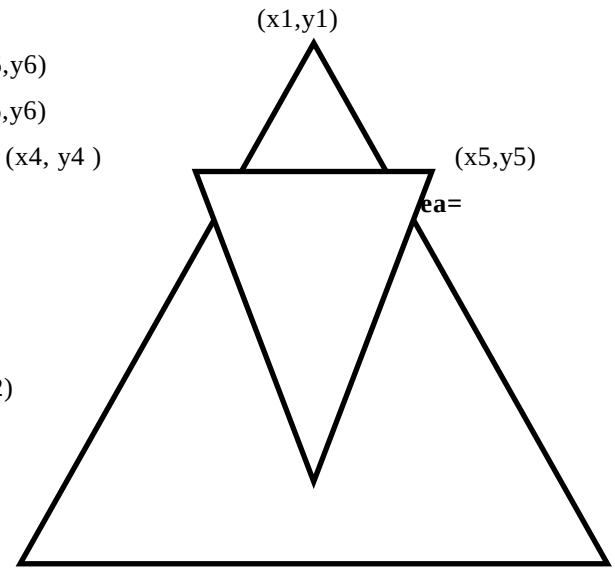
Step 3.

Metamorphic Relation 1:

a1=(x1,y1)(x4,y4),(x5,y5),a2=(x4,y4)(x2,y2)(x6,y6)

a3=(x5,y5)(x6,y6)(x3,y3),a4=(x4,y4)(x5,y5),(x6,y6)

Total Area= a1+a2+a3+a4 MR 1



A general form triangle has six main characteristics three linear (side lengths a, b, c) and three angular (α, β, γ). The classical plane trigonometry problem is to specify three of the six characteristics and determine the other three.This is the our first relation.here universal formula of area is sum of all triangles area is equals to the sun

of outer triangle.so we put this relation in our coding and check if the total area is equals to the sum of inner areas then there is no fault if it gives value false then there is fault in the programing.

Step 4.First Input file given: This is the file of our first relation .Further this data given for optimization.

Table 3.1 Input data file

I1	1100.0000	2500.0000	1600.0000	5640.0000	7600.0000	2500.0000	FALSE
I2	2980.0000	2500.0000	1160.0000	5210.0000	6200.0000	2500.0000	FALSE
I3	1100.0000	2500.0000	1600.0000	5600.0000	7660.0000	2500.0000	TRUE
I4	1110.0000	2500.0000	1600.0000	5600.0000	7660.0000	2500.0000	FALSE
I5	9800.0000	2500.0000	1160.0000	5210.0000	6200.0000	2500.0000	FALSE
I6	1110.0000	2500.0000	1160.0000	5600.0000	7660.0000	2500.0000	FALSE
I7	9800.0000	2500.0000	1160.0000	5100.0000	6200.0000	2500.0000	FALSE
I8	1110.0000	2500.0000	1600.0000	5600.0000	7600.0000	2500.0000	FALSE
I9	9800.0000	2500.0000	1160.0000	5610.0000	6820.0000	2500.0000	FALSE
I10	1100.0000	2500.0000	1600.0000	5600.0000	7660.0000	2500.0000	FALSE
I11	9800.0000	2500.0000	1600.0000	5600.0000	6820.0000	2500.0000	TRUE
I12	1210.0000	2500.0000	1600.0000	5600.0000	7660.0000	2500.0000	FALSE
I13	9800.0000	2500.0000	1600.0000	5600.0000	6800.0000	2500.0000	TRUE
I14	2100.0000	2500.0000	1600.0000	5600.0000	7600.0000	2500.0000	FALSE
I15	2100.0000	2600.0000	8000.0000	6700.0000	9000.0000	5000.0000	TRUE
I16	5500.0000	2500.0000	1160.0000	5600.0000	7600.0000	2500.0000	FALSE
I17	9800.0000	2500.0000	1600.0000	5600.0000	6000.0000	2500.0000	FALSE
I18	1500.0000	2500.0000	1600.0000	5600.0000	7600.0000	2500.0000	FALSE
I19	7500.0000	2500.0000	1600.0000	5600.0000	7600.0000	2500.0000	FALSE
I20	7500.0000	2500.0000	1600.0000	5600.0000	6000.0000	2500.0000	FALSE

Input with MR1

It shows the result with each input values when it gives false value it means we detect the bug because we give the mutant in the input and vice versa. This is the example of only one MR.Then by applying AFSA we generate multiple test cases.

Step 5.Calculation of mutation detection ratio: It is the ratio of mutant detected and total mutant present.

Metamorphic Relation 1: 16/20

Here in MR1 there are 16 mutants detected out of 20 inputs with mutants given.

Step 6.Comparison Of Metamorphic Relations: Here mutation detection ratio of different metamorphic relations are generated now it will compare the ratio and find the best metamorphic relations that detects more faults.

Table 3.2 Comparison of MRs

Metamorphic Relations	MR 1	MR 2	MR 3	MR 4	MR 5	MR 6	MR 7
MDR	0.8	0.85	1	0.9	1	1	1

Step 7. Overcome Two Major Challenges Of Metamorphic Testing: It shows the comparison table of both challenges

❖**Overcome challenge no. 1: Prioritization and minimization of metamorphic relations**:For most programs a variety of metamorphic relations with different fault–detection capabilities can be derived. In certain cases using all the metamorphic relations may be too expensive and a subset of them must be selected. It is therefore important to know how to prioritize the most effective metamorphic relations. To this end, several authors have proposed using code coverage or test case similarity with promising results. However, the applicability of those approaches as domain–independent prioritization criteria is still to be explored. Prioritization and minimization of metamorphic relations by analysis the comparison table of MDR we concluded:

Table 3.3 Priority of MRs

Metamorphic relation Priority
MR 3
MR 5
MR 6
MR 7
MR 4
MR 2
MR 1

❖**Overcome challenge no.2:Generation of likely metamorphic relations-**The generation of metamorphic relations is probably the most challenging problem to be addressed. Although some promising results have been reported, those are mainly restricted to the scope of numerical programs. The generation of metamorphic relations in other domains as well as the use of different techniques for rule inference are topics where contributions are expected. Generation of likely metamorphic relations by analysis the comparison table of MDR we concluded:

Table 3.4 Likely MRs

Likely generated metamorphic relations
MR 3
MR 5
MR 6
MR 7

IV. CONCLUSION AND FUTURE SCOPE

Conclusion:This paper demonstrates the use of metamorphic testing as a complement to special value testing using randomly generated values to test the software. Metamorphic Testing is very helpful in detecting faults when general testing fails to detect the bugs.Along with Metamorphic Testing, Artificial Fish Swarm Algorithm is deployed to increase the efficiency of Metamorphic Testing.The challenges of Metamorphic Testing like prioritization and likely generation overcome by us.

Future Scope: In this thesis, we have generated the metamorphic relations under Metamorphic Testing for testing the software manually which requires a lot of resources as well as time. So a mechanism can be devised in future to generate automated metamorphic relations to test the software more effectively and efficiently.

REFERENCES

[1]Segura, Sergio, G. Fraser, A. B. Sanchez, A. R. Cortes, *A Survey on Metamorphic Testing* ,IEEE,*42(9)*, 2016, 805 – 824.

[2]Segura, Sergio, G Fraser, A. B. Sanchez, and A.R.-Cortes *Metamorphic Testing: A Literature Review*, Applied Software Engineering Research Group,University of Seville Spain,*1.1*, ,july 2015.

[3] T. Y. Chen, S. C. Cheung, and S. M. Yiu,*Metamorphic testing:A new approach for generating next test cases*, Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology, Tech. Rep., 1998.

[4] W. Cody, *Software Manual for the Elementary Functions*,Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1980.

[5] M. Blum, M. Luby, and R. Rubinfeld,*Self-testing/correcting with applications to numerical problems*,Journal of Computer and System Sciences,*47(3)*,1993,549 – 595.

[6] P. E. Ammann and J. C. Knight,*Data diversity: An approach to software fault tolerance*,IEEE Transactions on Computers,*37(4)*,1988,418–425.

[7]W.K.Chan,S,C.Cheung,Karl,R.P.H.Leung,*A Metamorphic Testing approach for Online Testing of service oriented applications*,International Journal of Web Services Research,*4(2)*,2009,61-81.

[8]E.J. Weyuker,*On Testing Non-Testable Program*,Oxford Journals Science & Mathematics Computer Journal *25(4)*,1982,465-470.

[9]T song Yueh Chen,*Metamorphic Testing: A Simple Approach to Alleviate the Oracle Problem*,International Journal of Modern Engineering Research (IJMER),*3(2)*,2013,990-99.

[10]Barr,E.T Harman,M.Mc Minn,P.Shabhaz,M.ShinYoo,*The Oracle Problem in Software Testing:A Survey*,Software engineering,IEEE,*41(5)*,2015,507-525.