ISSN No. 0976-5697

Volume 8, No. 3, March – April 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Dynamic Test Data Generation using Negative Selection Algorithm and Equivalence Class Partitioning

Wasiur Rhmann
Babasaheb Bhimrao Ambedkar University,
Lucknow, India

Dr. Gufran Ahmad Ansari
Assistant Professor, Department of Information Technology,
College of Computer, Qassim University, Al-Qassim,
Kingdom of Saudi Arabia(KSA)

Abstract: Test data generation is challenging in software testing. Huge amount of generated test data is used to expose faults in the software. Test data generated inefficiently leads extra efforts. In path testing different independent paths generated from Control Flow Graph (CFG) are forced to traverse by test data. In the present paper, we proposed a novel technique to dynamically reduce test data based on Equivalence partitioning of output domain. Paths which are functionally meaningful are considered for test data generation. Experiments are performed on benchmark programs.

Keywords: Test data, Negative Selection Algorithm, Activity Diagram, Software Testing

I. INTRODUCTION

All Software testing is an important activity of the software development process. It improves the quality of the software and enhances the reliability. Testing is performed with the intent to show presence of faults. Test data is generated to check whether software is satisfying requirements. Due to large input domain of the software. Generation of all possible input data will be infeasible. Different coverage criteria are proposed to determine the adequacy of test data. These coverage criteria are based on program. Statement coverage criteria are least criteria required for unit testing [1]. Branch coverage exercise both true and false conditions of branch of control flow graph of a program. Branch coverage is stronger criteria than statement coverage. There are many other criteria in literature and path coverage is strongest criteria. Path coverage can not possible to achieve in the presence of loop. Path testing is strongest criteria for generation of test data. There may be infinite number of paths due to presence of loops. So test data for independent paths is usually considered for path testing. In path testing some paths may be infeasible i.e. no test data is possible which can traverse these paths. Presence of infeasible paths is problematic for test data generation. So in the present work only functionally meaningful paths are considered.

The rest of the paper is organized as follows: section 2 is related work and section 3 is Negative Selection algorithm section 4 described the proposed approach and section 5 is empirical study of proposed approach and finally section 6 concluded the work.

II. . RELATED WORK

Yao et al. [2] established a constrained multi objective model for test data generation. Authors considered better spatial distribution by considering the statement coverage. Genetic algorithm is used to solve the proposed model. Experimental results showed that proposed model has better fault finding ability. Zhang and Gong [3] presented a method based on multiple paths for detection of faults. A multi objective optimization problem with constrained is formulated based on multiple paths and faults. Weighted genetic algorithm is used to solve the proposed model. Yao and Gong[4] established a test model for test data generation for multiple paths coverage. Proposed model is solved using multi population genetic algorithm using individual sharing. Authors measured the performance of proposed approach theoretically and experimentally. Shimin and Zhangang [5] proposed a genetic algorithm based test data generation technique. Proposed approach reduced the redundancy is generated test data. Gonga et al. [6] proposed a technique of multiple test data generation. Evolutionary algorithm is used for test data generation based on grouping. Paths similarities are used to group similar paths and each group is used to form as sub optimization problem. These sub optimization algorithms are solved for test data generation.

III. NEGATIVE SELECTION ALGORITHM

Artificial immune system is an active research which is inspired from Biological immune system [7] [8]. Different types of AIS methods are used in change detection, fault detection and network intrusion detection [9] [10]. NSA, Clonal selection and immune network model are some AIS methods used extensively [11].

Human immune system protect from several infectious diseases and helps in proper functioning of the body. Negative selection algorithm is an Artificial Immune System (AIS) method inspired from Natural Immune System. Complicated computations are solved using immune based techniques. Negative selection algorithm works as classification algorithm. It classifies the data into self and non-self. In NSA a number of detectors are generated from search space. Then new generated data is classified as self or non-self-based detectors. Generation and maturation of T-cells in thymus is termed as Negative selection algorithm. T-cells that match self are deleted before they release to body.

There are two stages in NSA

1. Generation stage(Training stage)

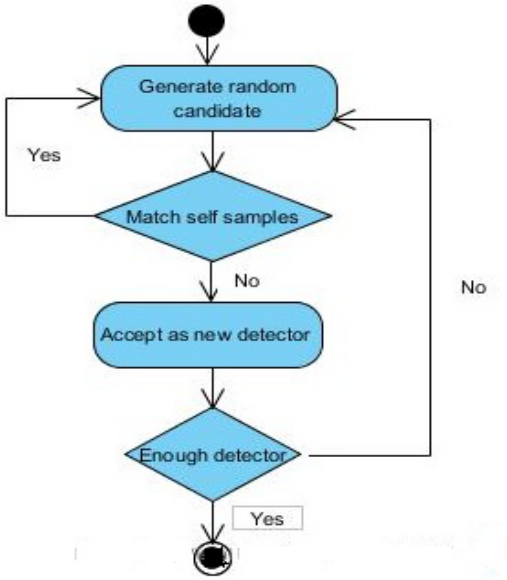

Figure 1 Generation of detectors

In the first stage of Negative selection algorithm number of detectors are generated randomly. If a generated data is matched with already available detectors then generated detector is discarded else it is added in detector set till the desired number of detectors are achieved.

2. Detection stage(Testing stage)

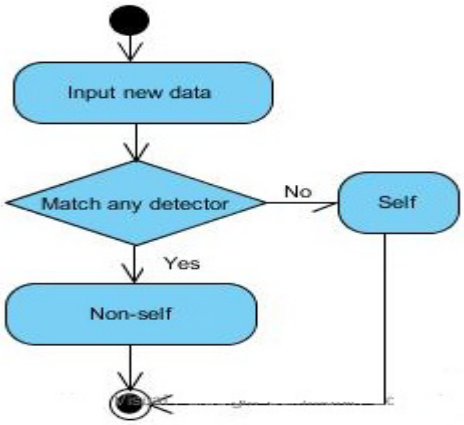

Figure 2 Detection stage

In detection stage detector set are used to classify the input sample as self and non-self. If new data is matched with any detectors is classified as a non-self.

IV. PROPOSED METHODOLOGY

Overview proposed technique of test data generation is given in Fig. 3. There are two main steps in test data generation process in first step Test data is generated randomly and redundant data is removed. Generated test data works like detectors of NSA algorithm. Test data is generated in binary form for each variable of the program. When enough test data are available from first step then, in the second step new test data is generated and newly generated test data is matched with the available test data from first step if newly generated data is not matched then hamming distance from all available test data is calculated. If hamming distance is greater than a constant value determined by trial and errors then software under test (SUT) is executed with newly generated test data. After execution if output matches from available equivalence classes then newly generated test data is added in the set of test data and that equivalence class is removed from Output file of equivalence classes.

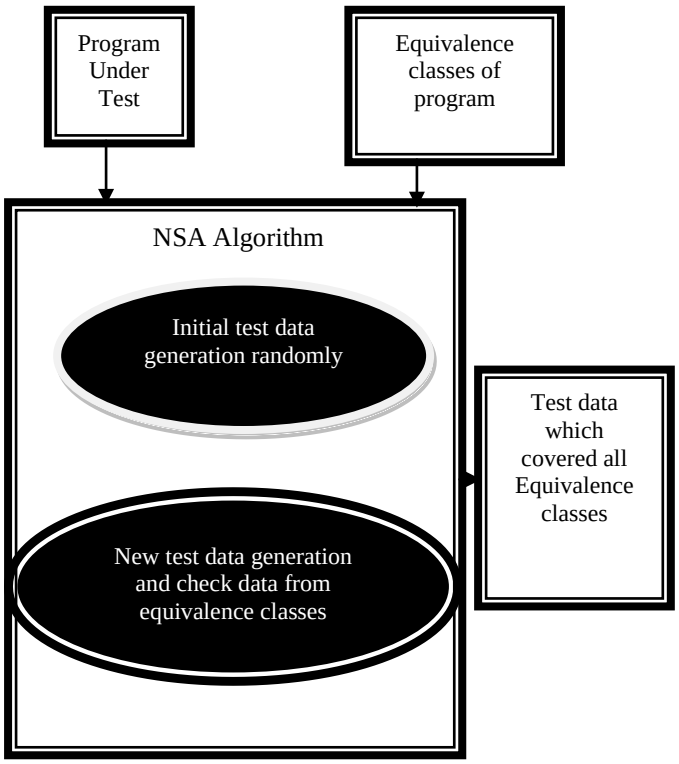

Figure 3 Overview of proposed approach

Input:
1- The program under test *P* and its input variable list *X*= (*x*1, *x*2, …, *x*n)
Where x is data from search space of program;
2-Output Equivalence Classes of program P;
3- The parameters of the algorithm, Number of detectors *max*, c;
Output:
1- Set of test data *D=(d1, d2, …, dn)* which satisfied the functionally meaningful path coverage and output equivalence classes;
Begin
Step 1: Initial test data is generated randomly;
Step 2: If initial test data reach to coverage then go to end
Step 3: Generate a new test data *x* from search space of program;
Step 4: Calculate the similarity of *x* with every test data *di* in *D*, which is the sum of hamming distance and could be calculated as given below;

H=Sum of hamming distance of test data x with already available test data d_i;
Step 5: Check the distance H

Step 6: if H<c then Remove the new test data *x*;
Where c is constant value determined by trial and errors;
Step 7: else if *x* from output Equivalence class then add s to D and Remove that Equivalence class from output file of Equivalence class;
Step 8: end if
Step 9: Repeat steps 3 to 8 until detector number >*max* **or** *D* reach to full coverage of functionally meaningful paths *U*;
End

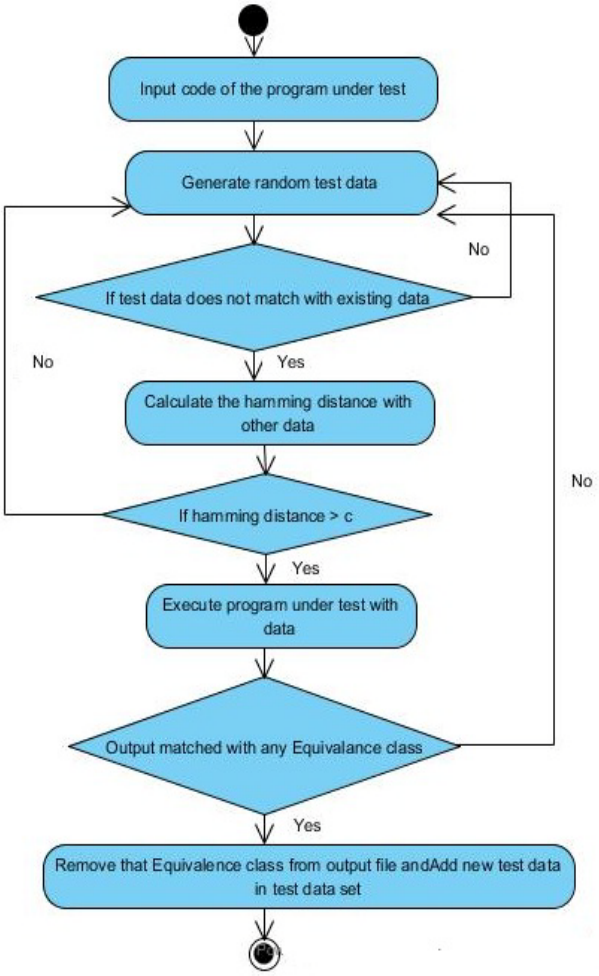

Figure 4 Activity diagram of proposed algorithm

V. EMPIRICAL VALIDATION

Fig. 5 shows the sample java program for finding roots of quadratic equation and Table 1 shows output equivalence classes of the program.

```
import java.io.*;
import static java.lang.System.out;
importjava.util.Scanner;
public class JavaApplication121
{
public static void main(String[] args) throws
FileNotFoundException
    {

double root1, root2, d;

      Scanner scanner = new Scanner(new File("F:/input.txt"));
String [] b = new String[1000]; int [] a = new int[1000];
inti = 0; int count=0;

while(scanner.hasNext()){
b[i] = scanner.nextLine(); a[i] = Integer.parseInt(b[i],2);
i++;
}
      d = a[1] * a[1] - 4 * a[0] * a[2];
if(d > 0)
      {
System.out.println("Roots are real and unequal");
         root1 = ( - a[1] + Math.sqrt(d))/(2*a[0]);
         root2 = (-a[1] - Math.sqrt(d))/(2*a[0]);
System.out.println("First root is:"+root1);
System.out.println("Second root is:"+root2);
      }
else if(d == 0)
      {
System.out.println("Roots are real and equal");
         root1 = (-a[1]+Math.sqrt(d))/(2*a[0]);
System.out.println("Root:"+root1);
      }
else
      {
System.out.println("Roots are imaginary");
      }
   }
}
```

Figure 5 Java program to find roots of quadratic equations

There are 3 equivalence classes of output domain of the program.

Table 1. Equivalence classes of program in Fig. 5

Name of Equivalence Class	Condition
Roots are real and unequal	d>0
Roots are real and equal	d=0
Roots are imaginary	d<0

Table 2. and Table 3. Shows the test data for quadratic equation program which is generated from generation step of NSA algorithm**.**

Table 2. Test data1

Test data1		
a[0]	a[1]	a[2]
1110	0111	1011
11100111101		

Table 3. Test data2

Test data2		
a[0]	a[1]	a[2]
1110	0101	1001
111001011001		

Table 4. shows the hamming distance calculated from test data1 and test data2

Table 4. Hamming distance calculation

Test data1	Test data2	Hamming distance
11100111101	11100111101	2

For experimental purpose java programming is used and proposed algorithm is implemented in Net beans using java. Test data is generated for program given in Table 5.

Table 5. Test data generated for sample programs

Program	Number of generations	Test data
Quadratic equations	2	53
Triangle classification	5	115

VI. CONCLUSIONS

In the present work a NSA based dynamic test data generation algorithm is proposed. Generated test data covered output domain with less number of data. Different search based techniques are used by researchers while proposed technique discarded the redundant data. Data redundancy is also removed by equivalence class partitioning of the output domain. Consideration of functionally meaningful paths saves efforts in test data generation.

VII. REFERENCES

[1] B. Beizer, Software Testing Techniques, Second Edition, Dreamtech Press, 2011.

[2] X. Yao, D. Gong, G. Zhang, "Constrained multi-objective test data generation based on set evolution", IET Software, Vol. 9, no. 4, pp. 103-108.

[3] Y. Zhang and D. Gong,"Generating test data for both coverage and faults detection using genetic algorithms: multi-path case", Front. Comput. Sci., Vol. 8, No. 5, pp. 726-740.

[4] X. Yao and D. Gong, "Genetic Algorithm-Based Test Data Generation for Multiple Paths via Individual Sharing", Computational Intelligence and Neuroscience, 2014, Article ID 591294, pp. 1-12.

[5] L. Shimin and W. Zhangang, " Genetic Algorithm and its Application in the path-oriented test data automatic generation", Procedia Engineering, 2011, Vol. 15, pp. 1186-1190.

[6] D. Gonga, W. Zhang, X. Yao, "Evolutionary generation of test data for many paths coverage based on grouping",The Journal of Systems and Software, 2011, Vol. 84, pp. 2222– 2233.

[7] J. Kim, P. J. Bentley, U. Aickelin, "Immune system approaches to intrusion detection-a review", Nat. Compt., 2007, Vol. 6, No. 4, pp. 413-466.

[8] D. Dasgupta, "Advances in artificial immune systems", IEEE Comput. Intel Mag., 2006, Vol. 1, No. 4, pp. 40-49.

[9] T. Li, "An Immunity based network security risk estimation", Sci Chin Ser F, 2005, Vol. 48, No. 5, pp. 798-816.

[10] J. Timmis, P. Andrews, N. Owens, "An interdisciplinary perspective on artificial immune systems", Evol. Intel, 2008, Vol. 1, pp. 5-26.

[11] Li T, Computer immunology, Beijing Publishing House of Electronics Industry, 2004.