Automatic Test Case Generation for Structural Testing Using Negative Selection Algorithm

Shayma Mustafa Mohi-Aldeen 1,2*, Radziah Mohamad 1, Safaai Deris 1

1 Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia
2 University of Mosul, Mosul, Iraq

Abstract

Software testing is an essential part of software development as it consumes a one half of the software development cost. The most critical step of software testing is generation of the test cases. Many approaches have been developed by researchers to automate it. Negative Selection Algorithm NSA has been used to generate test cases automatically to satisfy path coverage of software. The proposed algorithm has been applied to the most widely used benchmarking program which is triangle classifier, and the experimental results show the test cases are efficient in time of execution and effective in generation of test cases. The results are compared with random testing to assess the efficiency and the effectiveness of the proposed algorithm.

Keywords. Software Testing; Test Case Generation; Negative Selection Algorithm

1 Introduction

Software testing is an important activity of Software Development Life Cycle, it is simply a program execution aims to discover errors [1]. Software testing is costly and laborious. Software testing takes up to 50% of software development costs and it can be done either automatically or manually [2].

In structural testing, a test case is designed to cover a program meaning executing an entire program by searching for test cases that satisfy chosen testing criteria. There are three major types of coverage criteria: statement coverage (every statement in the program has been executed at least once), branch coverage (every logical branch in the program is executed with both outcomes at least once), and path cov-
verage (every distinct path in the program is executed at least once). Path coverage is the strongest form of structural coverage [3] and is the focus of this paper.

The most critical step of testing involves test cases generation. Test cases are a set of conditions or inputs given to an application to validate its functionality. Test data generation means the identification of a set of input data for a program which satisfy the specified criterion of testing. The process of generation of test cases should be accomplished with the implementation of test adequacy criterion that is determined when the testing process of a program is finished. Manual testing is a tedious process which consumes large amounts of time particularly because of the manual effort devoted to solve the problem. As a result, many papers have been proposed to automate the testing process in order to reduce the cost and to increase confidence in the results [4]. A good test case has a high detection rate of faults in software. The aim of some researchers is to optimize and improve test cases quality [5].

Path testing is a structural testing method that guarantees to execute every path through a program at least once. One of the main difficulties is automatic generation of suitable test data set that satisfies the complete path coverage of a program. As it is impossible to cover all the paths in a program, path testing method involves the selection of a subset of paths for executing and finding test data to cover it. Researchers have proposed several methods to generate test data automatically for path testing [3]. The generations of test data using random, symbolic and dynamic approach are insufficient to generate enough suitable test data Therefore there is need for generating test data using search based technique. This paper presents a new approach in the generation of test cases automatically for path testing using Negative Selection Algorithm NSA to ensure the complete coverage of the target path, which is one of the most important algorithms in artificial immune system [6]. NSA has been applied in different areas [7] but this is the first time that it has been used in generating of test cases.

The paper is organized as follows: section 2 presents the definition of path testing and steps to cover it, section 3 describes the related works; the proposed algorithm is presented in section 4; section 5 presents the discussion of the results, while conclusions presented in section 6.
2 Path Testing

Path coverage is the most effective criteria in structural testing. If the testing could be designed to execute all paths, each statement in the program will have been executed guarantee at least once and each condition will have been executed in false and true branches, ensuring maximum path coverage. The main task of path testing is generating test cases that fulfill path coverage; therefore, to achieve an efficient structural testing we must take path coverage as a testing criterion. There are four basic steps to generate test cases for path testing [8]:

1. Construction of Control Flow Graph: The source code of a program is converted to a graph, which represents the control flow of a program known as CFG.
2. Target Path Selection: In this step, extract the paths from the CFG, and select the target path that is the most significant.
3. Generation and Execution of Test Cases: Execute the method to create test cases automatically and execute a new path of control flow until all paths have been executed. At the end, suitable test cases that execute all paths have been generated.
4. Evaluation of Test Result: In this step, selected path has been executed to determine the satisfaction of the testing criterion.

Negative selection algorithm can be applied in structural testing to achieve path coverage if the paths and a suitable fitness function are well defined.

3 Related Works

The generation of test data that satisfy the adequacy criteria is one of the main difficulties in software testing. Many researches works have been done to solve this difficult problem. Different techniques were used to automate the generation of test data for the program under test based on structural testing for different coverage criteria. This paper focuses on path coverage criterion. Genetic Algorithm GA is the most common metaheuristic technique used in this field, which was used in several researches to accomplish path coverage. The study of Al-Zabidi et al. (2013) presented test case generation using Genetic algorithm with different types of software systems and the results were compared with random testing. The authors proposed a fitness function known as shifted modify similarity to achieve path coverage [8]. Srivastava and Kim (2009) presented a method for optimized software testing efficiency by identified the most critical path clusters in the program by developed a variable length genetic algorithm and they have proved that Genetic Algorithm finds
the most critical paths to improve the efficiency of software testing [9]. Genetic algorithm automatically generates test cases to test selected path by taking it as a target and executing sequences of operators iteratively for test cases to evolve. The evolved test case then leads the program execution to achieve the target path as shown by Nirpal and Kale (2011) who proposed a fitness function to achieve path coverage that included the distance between selected path and target path [10]. Ahmed and Hermadi (2008) attempted to generate test data for multiple paths using genetic algorithm [11]. Gupta and Gupta (2012) focused on the use of genetic algorithms for generating test data that can cover the most error-prone path, so that emphasis can be given to test these paths first [12]. Suresh and Rath (2013) used Genetic Algorithm to generate test data for feasible basis paths. The authors proposed a fitness function based on the condition of the predicate node. The results have shown that GA is more effective and efficient than random testing [13]. Rao et al. (2013) designed methodology for test data generation by using GA to cover the most critical path of a program [14]. Singh (2012) applied GA to generate test data for path coverage. The fitness function was used the hamming distance between target path and executed paths. The results showed that the quality of test cases is higher than the quality of test cases produced at random [15]. Liu et al. (2013) used GA to achieve both path and branch coverage of program in test data generation [16].

Particle Swarm Optimization PSO has been used in generating test cases. Nie (2012) applied PSO to achieve path coverage and the results show that PSO outperformed GA and enhanced the efficiency of test case generation [17]. Li and Zhang (2009) presented a method of generating all path test data of program based on PSO using new fitness function and registered the frequencies for all paths. The results showed that the efficiency had enhanced all paths compared with single path test data generation [18]. Hybrid algorithm that combines GA with PSO has been proposed by Li et al. (2010) to achieve path coverage. The results showed that the approach is simpler and more effective in generating test data automatically when compared with GA and ant colony optimization [19]. Memetic algorithms (MAs) proposed by Zhang and Wang (2011) that use both global and local search have been applied in test case generation by combining GA with simulated annealing to generate test data for path coverage. The results show that this method was superior to GA in effectiveness and efficiency [20]. Mansour and Salame (2004) applied GA and simulated
annealing in test cases generation to attain path coverage using hamming distance and the results were compared with hill climbing algorithm [21].

An additional metaheuristic technique that could be used in the field of automatic test case generation is Artificial Bee Colony ABC. Mala and Mohan used ABC to achieve path and branch coverage and compared the results with GA which showed that ABC generated test cases within fewer test runs with higher coverage percentage [22], while Kulkarni et al. used ABC to achieve full path coverage and compared the results with GA and Ant Colony Optimization [23].

4 The Proposed Algorithm

Path testing is the main strategy of structural testing and the basic way to solve the path testing is finding the specified input data that is likely to cover a path in the program under test. Many works have been done to generate test data automatically to achieve path coverage.

The aim of this research is to propose a method for generating test cases automatically to achieve path coverage that guarantees coverage of all paths of the program under test by using Negative Selection Algorithm. This algorithm was achieved by defining a set of self-samples that are generated according to the data set, then generating detectors that are one of the main components of NSA. The generation of detector begins with random generation of a candidate population of detectors that are then matured during an iterative process. Hamming distance [6] will then be used to find the distance between detectors. The proposed algorithm selects the detector with maximum distance and completes the algorithm in order to cover all search domain which will be iterated until the stopping criteria has been met. Here the stopping condition is the maximum number of test cases. The steps of proposed algorithm are shown in Figure (1).
Input: Program under test
Output: Set of test cases
Step 1: Program Instrumentation
Step 2: Construction of control flow graph from the source code of program
Step 3: Generate initial candidate detectors set randomly
(The detectors represent test cases)
Step 4: Repeat
Step 5: Generate a new detector
Step 6: Calculate the hamming distance between two detectors, hamming distance can be calculated from Equation 1,
\[ D_{A,B} = \sum_{j=1}^{n} |x_{A,j} - x_{B,j}| \]  
where A and B are any two detectors
Step 7: Continue until stopping criteria satisfied or exceed maximum number of generation.

Figure 1. The Proposed Algorithm Steps

5 Discussion of Results

In this work, NSA was used to generate test cases automatically for path coverage of structural testing. In order to investigate the performance of the proposed algorithm we compared the results with random test case generation. The triangle classifier benchmark program will be experimented as it widely used in software testing by many researchers [6][8][9][13] and others. The aim of this program is to determine the input of three edges to form a triangle and classifying the triangle type to determine if it is isosceles, scalene or equilateral.

Figure (2) gives the source code of the program and the corresponding construction of the control flow graph that contains four paths:

Path 1: d (Not Triangle)
Path 2: ae (Scalene)
Path 3: abf (Isosceles)
Path 4: abc (Equilateral)
In this work, we used 1000 generations for both random algorithm and negative selection algorithm NSA. Based on the results we can say that NSA can be used in test case generation. Table (1) shows the number of test cases of paths for Figure (2) in each generation for 10 generations and testing time execution in random test case generation. Figure (3) shows the number of test cases of the paths in each generation and for 10 generations that have been generated randomly.

Table 1. Number of Test Cases of the Paths for 10 Generations in Random Algorithm

<table>
<thead>
<tr>
<th>Generation</th>
<th>Not Triangle</th>
<th>Equilateral</th>
<th>Isosceles</th>
<th>Scalene</th>
<th>Time(ms)</th>
<th>Total no. of test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>509</td>
<td>0</td>
<td>20</td>
<td>471</td>
<td>0.093</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>506</td>
<td>0</td>
<td>24</td>
<td>470</td>
<td>0.094</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>528</td>
<td>0</td>
<td>22</td>
<td>450</td>
<td>0.094</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>0</td>
<td>18</td>
<td>482</td>
<td>0.078</td>
<td>1000</td>
</tr>
<tr>
<td>5</td>
<td>520</td>
<td>0</td>
<td>23</td>
<td>457</td>
<td>0.078</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>526</td>
<td>1</td>
<td>22</td>
<td>451</td>
<td>0.078</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>536</td>
<td>0</td>
<td>25</td>
<td>439</td>
<td>0.063</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>525</td>
<td>0</td>
<td>20</td>
<td>455</td>
<td>0.078</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>502</td>
<td>0</td>
<td>24</td>
<td>474</td>
<td>0.078</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>550</td>
<td>0</td>
<td>23</td>
<td>427</td>
<td>0.078</td>
<td>1000</td>
</tr>
</tbody>
</table>
Figure 3. Number of test cases of the paths in Table (1) for 10 generations in Random Algorithm

Table (2) shows the number of test cases of paths for Figure (2) in each generation for 10 generations and testing time execution by using the proposed algorithm NSA. Figure (4) shows the number of test cases of the paths in each generation and for 10 generations that have been generated by using NSA. From the comparison of the two algorithms, we can see that NSA is better than random algorithm in generated test cases for all paths and especially for equilateral path which is the most difficult path in terms of coverage. In addition, the proposed algorithm reduces the execution testing time as shown in the table below.

Table 2. Number of test Cases of the Paths for 10 Generations in NSA

<table>
<thead>
<tr>
<th>Generation</th>
<th>Not Triangle (d)</th>
<th>Equilateral (ae)</th>
<th>Isosceles (abf)</th>
<th>Scalene (abc)</th>
<th>Time(ms)</th>
<th>Total no. of test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>533</td>
<td>2</td>
<td>26</td>
<td>439</td>
<td>0.075</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>524</td>
<td>4</td>
<td>20</td>
<td>452</td>
<td>0.063</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>535</td>
<td>4</td>
<td>18</td>
<td>443</td>
<td>0.074</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>531</td>
<td>2</td>
<td>24</td>
<td>443</td>
<td>0.076</td>
<td>1000</td>
</tr>
<tr>
<td>5</td>
<td>509</td>
<td>2</td>
<td>17</td>
<td>472</td>
<td>0.05</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>516</td>
<td>4</td>
<td>17</td>
<td>463</td>
<td>0.092</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>519</td>
<td>6</td>
<td>14</td>
<td>461</td>
<td>0.061</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>519</td>
<td>2</td>
<td>20</td>
<td>459</td>
<td>0.067</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>506</td>
<td>2</td>
<td>21</td>
<td>471</td>
<td>0.082</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>517</td>
<td>6</td>
<td>17</td>
<td>460</td>
<td>0.071</td>
<td>1000</td>
</tr>
</tbody>
</table>
From the comparison between these two algorithms, we find that NSA has a great chance to generate test cases for the program paths especially for the most difficult path i.e. Equilateral path (ae) and this can be seen in Figure (5) which shows the average number of test cases generated by using NSA and Random testing.

### Figure 4. Number of Test Cases of the Paths in Table (2) for 10 Generations in NSA

![NSA Generation](image)

### Figure 5. Average Number of Test Cases for program paths using NSA and Random Testing

![Average no. of test cases of the four paths using Random Testing and NSA](image)

<table>
<thead>
<tr>
<th>Path</th>
<th>Random</th>
<th>NSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Triangle</td>
<td>520</td>
<td>522</td>
</tr>
<tr>
<td>Equilateral</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Isosceles</td>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>Scalene</td>
<td>456</td>
<td>458</td>
</tr>
</tbody>
</table>

### 6 Conclusion

In this paper, negative selection algorithm has been used in generation of test cases automatically for path coverage. This is the first time that it has been used in this field and its performance has been proven. By using triangle classification benchmark program, the results show that the proposed algorithm is more efficient and more effective than random generation because of the ability of the proposed algorithm to move the search to the desirable search range with less time.
For future work, the proposed algorithm will be under experimentation using more benchmark programs in order to evaluate the proposed algorithm performance as well as to compare results with other test case generation algorithms like genetic algorithm.

ACKNOWLEDGMENT

The researchers would like to thank Universiti Teknologi Malaysia for providing the facilities and support for the research, and to thank Ministry of Higher Education & Scientific Research of Iraq and University of Mosul for sponsoring the research.

References