Automated path testing using the negative selection algorithm

Shayma Mustafa Mohi-Aldeen*
Faculty of Computing,
Universiti Teknologi Malaysia,
81310 UTM Skudai, Johor, Malaysia
and
College of Computer Science and Mathematics,
University of Mosul,
41002, Mosul, Iraq
Email: shaima_mustafa76@yahoo.com
Email: mmshayma2@live.utm.my
*Corresponding author

Radziah Mohamad and Safaai Deris
Faculty of Computing,
Universiti Teknologi Malaysia,
81310 UTM Skudai, Johor, Malaysia
Email: Radziahm@utm.my
Email: Safaai@utm.my

Abstract: Software testing is an important step in the software development process, accounting for more than 50% of software development cost as it is laborious and time-consuming. Generating path test data is the most critical stage in software testing and many approaches have been developed by researchers to automate it. Negative selection algorithm (NSA) has been used in this paper to generate test data for path testing automatically. The proposed algorithm has been applied to the most commonly used benchmarking program which is triangle classifier. The experimental results show that the proposed algorithm is more efficient in time of execution and more effective in the generation of test data when compared with random testing and genetic algorithm.

Keywords: path testing; automatic test data generation; ATDG; negative selection algorithm; NSA.


Biographical notes: Shayma Mustafa Mohi-Aldeen has completed her MSc in Computer Science from the University of Mosul in 2000 and BSc degree in the same university in 1998. Since 2000, she has been working as a Lecturer in the University of Mosul. She is presently doing her PhD research at the University Technology Malaysia. Her areas of interest are software engineering, software testing and artificial intelligent methods.
Automated path testing using the negative selection algorithm

Radziah Mohamad is a Senior Lecturer at the Software Engineering Department, Faculty of Computing, Universiti Teknologi Malaysia. She received her Bachelor of Science (Hons) in Software Engineering from Sheffield Hallam University, Master of Science degree (Computer Science) and Doctor of Philosophy degree (Computer Science) from the Universiti Teknologi Malaysia. She has been a Lecturer at the Software Engineering Department for more than ten years. Her research interests include service-oriented design, pattern-oriented design, formal methods, component-based software engineering and agent-oriented software engineering.

Safaai Deris is a Professor of Artificial Intelligence and Software Engineering at the Faculty of Computing, Universiti Teknologi Malaysia. Currently, he is the CIO and Director of the Center for Information and Communication Technology. His previous positions include Deputy Dean of the School of Graduate Studies and Head of the Software Engineering Department. He is also the Head of the Artificial Intelligence and Bioinformatics Research Group. He received his MEng in Industrial Engineering, and the DEng in Computer and System Sciences, both from the Osaka Prefecture University, Japan. His research interests include the software engineering, applications of intelligent techniques in planning, scheduling, bioinformatics and system biology.

This paper is a revised and expanded version of a paper entitled ‘Automatic test case generation for structural testing using negative selection algorithm’ presented at 1st International Conference of Recent Trends in Information and Communication Technologies-IRICT 2014 Proceeding, Universiti Teknologi Malaysia, Johor, Malaysia, 12-14 September 2014.

1 Introduction

Software testing is an important activity of the software development life cycle; it is a process of executing a program to discover errors (Myers and Sandler, 2004). Software testing is costly and laborious, taking up to 50% of software development costs and can be performed either automatically or manually (Keyvanpour et al., 2011).

The most critical step of software testing is the generation of test data (set of program input data) for test cases. The process of generating test data 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 time consuming process, particularly because of the manual effort devoted to solve the problem. The automation of this generation can improve the process to satisfy the required test adequacy criterion. This process is called an automatic test data generation (ATDG). As a result, many papers have been proposed to automate the testing process in order to reduce the cost and time and to increase confidence in the results (Anbarasu and Anitha Elavarasi, 2012). Some researchers aim to optimise and improve test data quality (Harman and McMinn, 2010).

In structural testing, test data generation means searching for test data to execute the internal structure of the program under test which satisfy a chosen testing criterion. There are three major types of structural coverage criteria: statement coverage (i.e. every statement in the program has been executed at least once), branch coverage (i.e. every logical branch in the program is executed with both outcomes at least once), and path
coverage (i.e. every distinct path in the program is executed at least once). Path coverage is the most important in terms of structural coverage since it includes all the previous types of the test coverage (Parnami et al., 2012) and it is the focus of this paper.

Test data generation is an undecidable problem and can be non-deterministic NP-hard or existing solutions are infeasible. The extremely nonlinear construction of the program presents a challenge to search algorithms to find efficient and optimal test data from a nonlinear, complex and discontinuous input in the search space (Lam et al., 2012).

Path testing is a structural testing method that guarantees that every path through a program will be executed at least once. One of the main difficulties is how to generate an effective test data that traverse all paths of a program within a short period of time. 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 (Parnami et al., 2012). The generation of test data using random, symbolic and dynamic approaches are insufficient to generate enough suitable test data (Ghiduk, 2008). Therefore, there is a need for generating test data using search-based technique.

This paper presents a new approach in the generation of test data automatically for path testing using negative selection algorithm (NSA), which is one of the most important algorithms in artificial immune system (Dasgupta and Nino, 2009) to ensure the complete coverage of the target path and reduce the manual work. NSA has been applied in different areas (Dasgupta, 2006) but it has not been used in generating test data.

The paper is organised as follows: Section 2 presents the motivation of this research; Section 3 describes the related works; the proposed algorithm is presented in Section 4; Section 5 presents the discussion of the results, while conclusions are presented in Section 6.

2 Motivation

Path coverage is the most effective criteria in structural testing. The main task of path testing is generating an effective test data that fulfil all logical paths through the program. Every program under test needs to build its related control flow graph (CFG) in order to compute and identify its paths. A program with N decisions has $2^N$ possible paths which are equal to its cyclomatic complexity (CC), while the existence of loops in a program will increase the number of paths, i.e. each different number of iterations of a loop are considered to be a different path. It is extremely costly, laborious, time-consuming and impossible to achieve a high percentage of path coverage by writing manual test data for any program. Even automated test generation tools may have a hard time generating test data to achieve complete path coverage, especially when the program contains nested loops. Exhaustive enumeration of program inputs is infeasible for any reasonable-sized program. Similarly, random methods are unreliable and unlikely to exercise ‘deeper’ features of software that are not exercised by simple chance. Dynamic test data generation techniques are ineffective, consume time and tend to get stuck in local optima of the domain space of input data (Al-Zabidi et al., 2013). These problems motivated the researchers to create test data generator based on metahuristic search algorithms which can test the structure of the program and automatically generate test data adequately to traverse and cover all software paths to reduce time and effort, e.g., genetic algorithms
Automated path testing using the negative selection algorithm

Although different works have been done to generate test data automatically, work must still be done to improve the efficiency and effectiveness of the process.

The aim of the work is to investigate the effectiveness of NSA over random testing and to automatically generate an effective test data to traverse all software paths with less time and effort.

3 Related works

The generation of test data that satisfy the adequacy criteria is one of the main difficulties in software testing. Many research 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 and each algorithm has its own strengths and weaknesses. This paper focuses on path coverage criterion.

Although GA is the most common metaheuristic technique used in this field, it has problems delivering stable results and the convergence is slow.

The study of Al-Zabidi et al. (2013) presented test case generation using GA with different types of software systems and the results were compared with random testing. The authors proposed a fitness function known as shifting modify similarity to achieve path coverage.

Srivastava and Kim (2009) presented a method for optimised software testing efficiency by identifying the most critical path clusters in the program by developing a variable length GA and they have proved that GA finds the most critical paths to improve the efficiency of software testing. GA automatically generates test data to test selected path by taking it as a target and executing sequence of operators iteratively for test data to develop. The evolved test case, then leads the execution of program to accomplish 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. Ahmed and Hermadi (2008) attempted to generate test data for multiple paths using GA. Gupta and Gupta (2012) focused on the use of GAs for generating test data that can cover the most error-prone path, so that emphasis can be given to test these paths first. Suresh and Rath (2013) used a GA 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. Rao et al. (2013) designed a methodology for test data generation by using GA to cover the most critical path of a program. Singh (2012) applied GA to generate test data for path coverage. The fitness function used the hamming distance between target path and executed paths. The results showed that the quality of test data is higher than the quality of test data produced at random. Liu et al. (2013) used a GA to achieve both path and branch coverage of program in test data generation. Ghiduk (2014) proposed a new GA to generate set of test paths automatically, depending on the length of chromosome which differs from one iteration to another depending on the changing of the path length, the method could also check the feasibility of the paths that have been generated.

Particle swarm optimisation (PSO) has been used in generating test data as it offers fast convergence and short computation time. However, it suffered from prematurity, which decreased the efficiency of test case generation. Nie (2012) applied PSO to achieve
path coverage and the results show that PSO outperformed GA and enhanced the efficiency of test case generation. Li and Zhang (2009) presented a method of generating all path test data of the 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. 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 ACO but it needed more time than PSO. 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, but it increases the cost. Mansour and Salame (2004) applied GA and simulated annealing in test data generation to attain path coverage using hamming distance and the results were compared with hill climbing HC algorithm. The results show that GA outperformed the local search problems and SA gives better coverage than HC.

An additional metaheuristic technique that could be used in the field of ATDG is artificial bee colony (ABC), Mala and Mohan (2009) used ABC to achieve path and branch coverage and compared the results with GA which showed that ABC generated test data within fewer test runs with higher coverage percentage, while Kulkarni et al. (2011) used ABC to achieve full path coverage and compared the results with GA and ACO. Lam et al. (2012) combined the three bees to make the generation of feasible independent paths and software test suite optimisation faster, and the results have been compared with GA, TS and ACO.

This paper presents a new technique to generate test data automatically which have not been used previously within this field and the results give an effective test data that could traverse all program paths early within less time when compared with the other techniques.

4 The proposed algorithm

Path testing is the main strategy of structural testing and the basic way to solve 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 data automatically to achieve path coverage that guarantees coverage of all paths of the program under test by using NSA. This algorithm was achieved by defining a set of self-samples that are generated according to the dataset, 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 (Dasgupta and Nino, 2009) will then be used to find the distance between the detectors. The proposed algorithm selects the detector with maximum distance and completes the algorithm in order to cover all search domains which will be iterated until the stopping criterion has been met. Here the stopping condition is the maximum number of test data. The steps of the proposed algorithm are shown below:
Automated path testing using the negative selection algorithm

Input: Program under test
Output: Set of test data

Step 1: Input the program under test source code
Step 2: Construct a CFG from the source code of program
Step 3: Compute the CC value from Equation 1,

\[ CC = P + 1 \]  

where \( P \) is the number of predicate nodes where the node has more than one neighbour node in the CFG. This value determines the number of independent paths that must be covered.

Step 4: Generate initial candidate detectors set randomly
(These detectors represent the test dataset which is the value of CC)
Step 5: Repeat
Step 6: Generate a new test case (new detector)
Step 7: Calculate the fitness value which is the hamming distance between two detectors that can be calculated from equation (2),

\[ D_{A,B} = \sum_{j=1}^{n} |x_{ja} - x_{jb}| \]  

where \( A \) and \( B \) are any two detectors
Step 8: Select the detector which has maximum fitness value for the new generation
Step 9: Continue until stopping criteria has been satisfied or the maximum number of generations has been exceeded.

5 Results discussion

In this paper, the NSA was used to generate test data automatically for path testing in order to reduce execution time and generate effective test data. In order to investigate the performance of the proposed algorithm we compared the results with random test case generation. The selected case study in this paper is the triangle classifier benchmark program which has been widely used in software testing by many researchers (Al-Zabidi et al., 2013; Gupta and Gupta, 2012; Lam et al., 2012; Srivastava and Kim, 2009) 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 not a triangle, isosceles, scalene or equilateral.

It is important to mention that this program is used common because even when using a large range of integers, only a few combinations satisfy a particular branch in the code. For example, using range from 1 to 10 for the three sides there are only 12 combinations out of the possible 1,000 that are scalene triangle. Also, the most difficult path for random search is the equilateral triangle path, because it has to find three equal integer values. This makes the triangle program an excellent test-bed for software testing and test data generation research.
Figure 1 gives the source code of the program and the corresponding construction of the CFG. The CC value is 4 which is the number of predicate nodes plus 1 (predicate node is the node having more than one neighbour node in the CFG). The value of CC identifies the number of independent paths of the program.

Table 1

<table>
<thead>
<tr>
<th>Path no.</th>
<th>Triangle type</th>
<th>Path covered</th>
<th>Test data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Not triangle</td>
<td>S → 1 → 7 → E</td>
<td>(9, 4, 3)</td>
</tr>
<tr>
<td>2</td>
<td>Equilateral</td>
<td>S → 1 → 2 → 4 → 6 → E</td>
<td>(5, 5, 5)</td>
</tr>
<tr>
<td>3</td>
<td>Isosceles</td>
<td>S → 1 → 2 → 4 → 5 → E</td>
<td>(7, 7, 3)</td>
</tr>
<tr>
<td>4</td>
<td>Scalene</td>
<td>S → 1 → 2 → 3 → E</td>
<td>(8, 6, 5)</td>
</tr>
</tbody>
</table>
Automated path testing using the negative selection algorithm

For example, consider an input domain as 1 to 10 (for simplicity) for the three integer parameters to classify the triangle \((x, y, z)\). The test data generated to cover the program paths can be presented in Table 1.

In this work, we used 1,000 test data for both random algorithm and NSA. Based on the results we can say that NSA can be used in test data generation and the test data quality produced by NSA is better than the quality produced randomly since it can direct the generating of test data to the required path earlier. Table 2 shows the number of test data paths of Figure 1 within each generation. For example and in order to compare with a research in literature, we select ten generations. Table 2 also presents testing time execution in a random test data generation. Figure 2 shows the number of test data paths within each generation and for ten generations that have been generated randomly.

Table 2 The number of test data paths for ten generations in random algorithm

<table>
<thead>
<tr>
<th>Generation</th>
<th>Not triangle ((d))</th>
<th>Equilateral ((abc))</th>
<th>Isosceles ((abf))</th>
<th>Scalene ((ae))</th>
<th>Time ((ms))</th>
<th>Total no. of test data</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>1,000</td>
</tr>
<tr>
<td>2</td>
<td>506</td>
<td>0</td>
<td>24</td>
<td>470</td>
<td>0.094</td>
<td>1,000</td>
</tr>
<tr>
<td>3</td>
<td>528</td>
<td>0</td>
<td>22</td>
<td>450</td>
<td>0.094</td>
<td>1,000</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>0</td>
<td>18</td>
<td>482</td>
<td>0.078</td>
<td>1,000</td>
</tr>
<tr>
<td>5</td>
<td>520</td>
<td>0</td>
<td>23</td>
<td>457</td>
<td>0.078</td>
<td>1,000</td>
</tr>
<tr>
<td>6</td>
<td>526</td>
<td>1</td>
<td>22</td>
<td>451</td>
<td>0.078</td>
<td>1,000</td>
</tr>
<tr>
<td>7</td>
<td>536</td>
<td>0</td>
<td>25</td>
<td>439</td>
<td>0.063</td>
<td>1,000</td>
</tr>
<tr>
<td>8</td>
<td>525</td>
<td>0</td>
<td>20</td>
<td>455</td>
<td>0.078</td>
<td>1,000</td>
</tr>
<tr>
<td>9</td>
<td>502</td>
<td>0</td>
<td>24</td>
<td>474</td>
<td>0.078</td>
<td>1,000</td>
</tr>
<tr>
<td>10</td>
<td>550</td>
<td>0</td>
<td>23</td>
<td>427</td>
<td>0.078</td>
<td>1,000</td>
</tr>
</tbody>
</table>

Figure 2 The number of test data paths within each generation and for ten generations in random algorithm (see online version for colours)
Table 3 shows the number of test data paths in Figure 1 and testing time execution by using the proposed algorithm NSA. Figure 3 shows the number of test data paths within each generation and for ten generations.

Table 3  Number of test data paths for ten generations in NSA

<table>
<thead>
<tr>
<th>Generation</th>
<th>Not triangle (d)</th>
<th>Equilateral (abc)</th>
<th>Isosceles (abf)</th>
<th>Scalene (ae)</th>
<th>Time (ms)</th>
<th>Total no. of test data</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>1,000</td>
</tr>
<tr>
<td>2</td>
<td>524</td>
<td>4</td>
<td>20</td>
<td>452</td>
<td>0.063</td>
<td>1,000</td>
</tr>
<tr>
<td>3</td>
<td>535</td>
<td>4</td>
<td>18</td>
<td>443</td>
<td>0.074</td>
<td>1,000</td>
</tr>
<tr>
<td>4</td>
<td>531</td>
<td>2</td>
<td>24</td>
<td>443</td>
<td>0.076</td>
<td>1,000</td>
</tr>
<tr>
<td>5</td>
<td>509</td>
<td>2</td>
<td>17</td>
<td>472</td>
<td>0.05</td>
<td>1,000</td>
</tr>
<tr>
<td>6</td>
<td>516</td>
<td>4</td>
<td>17</td>
<td>463</td>
<td>0.092</td>
<td>1,000</td>
</tr>
<tr>
<td>7</td>
<td>519</td>
<td>6</td>
<td>14</td>
<td>461</td>
<td>0.061</td>
<td>1,000</td>
</tr>
<tr>
<td>8</td>
<td>519</td>
<td>2</td>
<td>20</td>
<td>459</td>
<td>0.067</td>
<td>1,000</td>
</tr>
<tr>
<td>9</td>
<td>506</td>
<td>2</td>
<td>21</td>
<td>471</td>
<td>0.082</td>
<td>1,000</td>
</tr>
<tr>
<td>10</td>
<td>517</td>
<td>6</td>
<td>17</td>
<td>460</td>
<td>0.071</td>
<td>1,000</td>
</tr>
</tbody>
</table>

Figure 3  Number of test data paths in Table 2 for ten generations in NSA (see online version for colours)

From Tables 1 and 2, we find that from 1,000 test data of the classify triangle problem two test data have been generated to traverse the equilateral path from the first generation using the proposed algorithm. It is worth noting that there is no chance to generate test data for the same path using random testing and there is just one chance to traverse it within the sixth generation as shown in Table 1. From the comparison between these two algorithms NSA has a greater chance to generate test data for the program paths, especially in the most difficult path which is an equilateral path (abc). This can be seen in Figure 4 which shows the average number of test data generated by using NSA and random testing.
Automated path testing using the negative selection algorithm

Figure 4  Average number of test data of program paths using NSA and random testing (see online version for colours)

<table>
<thead>
<tr>
<th></th>
<th>Not Triangle</th>
<th>Equilateral</th>
<th>Isosceles</th>
<th>Scalene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>522</td>
<td>0</td>
<td>22</td>
<td>456</td>
</tr>
<tr>
<td>NSA</td>
<td>519</td>
<td>3</td>
<td>19</td>
<td>459</td>
</tr>
</tbody>
</table>

Figure 5  Comparison of NSA with random algorithm and GA for equilateral path (see online version for colours)

From the comparison of the two algorithms, we can see that the NSA is more successful than random algorithm in generating test data to cover all paths of the program and especially for equilateral path which is the most difficult path in terms of coverage, e.g. the equilateral path was covered three times using NSA while it was covered only 0.1 times using random method and this can be seen in Figure 4. The proposed method
improves the coverage by more than 80% compared to random; in addition, it reduces the execution testing time by 67% when compared with a random algorithm.

In order to evaluate the results, the proposed method has been compared with Nirpal and Kale (2011) research from the literature which used a GA to generate triangle program test data automatically for the same type of coverage of 1,000 test data and for ten generations. Figure 5 shows the comparison between the proposed algorithm with random and GA for the equilateral path since it is the difficult path to cover. From the comparison, we can see that the proposed method has a greater chance to generate test data to cover the equilateral path and is 25% better than GA and 82% better than random method in terms of coverage.

6 Conclusions

In this paper, a NSA has been used in the generation of test data automatically for path coverage. The triangle classification benchmark program has been used as a case study; 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 desired search range with less time. In order to evaluate the results, the proposed algorithm has been compared with GA from literature and the results show that our proposed method can improve the quality of test data. For future work, the proposed algorithm will be under experimentation using more benchmark programs which contain loops and with different data types as well as real world programs in order to evaluate the proposed algorithm performance.

Acknowledgements

We would like to thank Ministry of Higher Education Malaysia for sponsoring the research through the FRGS grant and Universiti Teknologi Malaysia for providing the facilities and support for the research. The author also would like to thank Ministry of Higher Education & Scientific Research of Iraq and University of Mosul for sponsoring the research.

References


