Journal of Computer Sciences and Applications, 2015, Vol. 3, No. 2, 56-60
Available online at http://pubs.sciepub.com/jcsa/3/2/7
© Science and Education Publishing
DOI:10.12691/jcsa-3-2-7

Test Data Generation Using Computational Intelligence Technique

Harsh Bhasin[1,*], Naresh Chauhan[2], Sandhya Pathak[3]

[1]Department of Computer Science, Jamia Hamdard, Delhi, India
[2]Department of Computer Science, YMCAUST, Faridabad, India
[3]M.Tech Scholar, CSE Department, DITMR, Faridabad, India
*Corresponding author: i_harsh_bhasin@yahoo.com

Received March 18, 2015; Revised April 03, 2015; Accepted April 24, 2015

Abstract Testing is one of the most important parts of Software Development Life Cycle. It requires the crafting of a good test. This crafting can be done only after deep analysis and knowhow of the working of the software. This work presents the Genetic Algorithm based approach for the generation of test data. The approach would automate the test data generation process and hence facilitates the process of testing. The work would also help in the elimination of human biases. The work has been implemented in C#, verified with a set of 10 moderate size software. The results are encouraging. The work is part of a larger endeavor to develop a comprehensive testing system for C #software. This work is based on a comprehensive literature review which has helped develop a sound theoretical base.

Keywords: *genetic algorithms, software testing, test data generation, branch coverage*

Cite This Article: Harsh Bhasin, Naresh Chauhan, and Sandhya Pathak, "Test Data Generation Using Computational Intelligence Technique." *Journal of Computer Sciences and Applications*, vol. 3, no. 2 (2015): 56-60. doi: 10.12691/jcsa-3-2-7.

1. Introduction

Software testing is a complex task which takes a considerable amount of time [1]. Testing can be done by seeing the input and the output or by intricate inspection of a given code. The former is called black box testing whereas later is refer to as white box testing. The quality of test cases determines the quality of software. The test cases, therefore, become important in order to a certain the quality of given Program Under Test (PUT). The crafting of test cases is generally done manually however this process consumes a large amount of time. Therefore there is a need of an automated test data generator. The present work proposes an algorithm for the same. It is an extension of our earlier attempt for developing an automated test data generator [2,3,4]. This work uses Genetic Algorithms.

GAs are heuristic search processes based on survival of fittest [5]. These algorithms are generally used for optimization problems. These algorithms involve the application of operators like Crossover and Mutation describe later in the paper. These are known for generating a good result for many NP Complete problems [6,7]. The present work uses GA's for generating test cases. A test case consists of values of the input, expected output and the output obtained. The test cases in this work use an involved analysis of the program and consider the criteria of branch coverage [8]. The problem has been mapped with GA's to accomplish the task.

This paper has been organized as follows. Section 2 of the paper presents literature review. Section 3 briefly describes the concept of GA. Section 4 presents the Proposed Work which has been exemplified in section 5.

Section 6 presents the results and conclusions. The work paves way for the application of GA in Test Data Generation.

2. Literature Review

Literature review is one of the most important steps in a research. A good review not just finds gaps in the existing works but also points towards the solutions of the problems in the existing works. This section presents a brief review of the techniques in Test data Generation. The review has been largely carried out from the following databases.

1. IEEE
2. ACM
3. Springer
4. Science Direct
5. Wiley

Except for the above, some other important papers have also been included owing to their contribution in the field. Table 1 presents the author's name, technique used and the verification methods used.

The Literature Review bought forth the point that although Test Data Generation is an extensively researched topic, its automation is still in the naïve stage. There is an immediate need to develop methodologies and software for the automation of the process. This would help not only in the reduction in the development time and also produce only few important works, the actual review was done by taking around 69 papers. The comprehensive analysis of those papers has already been submitted for consideration in one of the journals.

Table 1. The Literature Review

Ref. No.	Authors name	Technique used	Verification method used
9	Praveen Ranjan Shrivastava and Tai- hoon Kim	Variable length Genetic Algorithm	In this paper comparison was done with random test data generation technique.
10	Harmen Hinrich Sthamer	Genetic Algorithm	In this paper comparison was done with pure random testing for software developed in ADA.
11	Jin-Cherng Lin and Pu-Lin Yeh	Normalized extended hamming distance and Genetic Algorithm	In this paper best results were obtained when 1000 test cases were applied in one generation.
12	Christoph C micheal, Mc Graw G and Scatz M A	Genetic Algorithm	In this paper function is minimized by using GA and effectiveness of the test cases are tested.
13	Joachim Wegener et. al.	Evolutionary testing	In this paper the verification is done with 6 test objescts in C language.
14	Dunwei Gong et. al.	Multi population parallel Genetic Algorithm	The verification was done by using 7 benchmark programs and compared with the method proposed in this paper.
15	Raquel Blanco et al.	Scatter search Metaheuristic technique	The verification was done by using 13 benchmark programs and compared with different test case generators.
16	Moheb R Girgis	Genetic Algorithm	The discovered criteria was compared with 3 previous evolutionary multiple path generators. Comparison was done with two versions of bubble sort.
17	Eugania diaz et. al	Tabu search metaheuristic search	The technique was compared with random approach.
18	R landa Baccerra et al.	Differential Evolution	Verification is done with 5 programs and comparison is done with BGA technique.

3. Proposed Work

The proposed algorithm for Test Data Generation is a continuation of one of our previous work. The work crafts the test cases as follows. The possible paths are crafted. This is followed by identification of variables that are defined variables. The values of those variables are then extracted from the specification and with the help of Genetic Algorithm the most suitable value substituted in order to craft a Test Case.

3.1. Algorithm

```
ALGORITHM (TEST CASE GENERATION)
INPUT - Program Under Test (PUT)
OUTPUT - SET OF TEST CASES
Where each test case has:
• Value of requisite variables
• Expected output
• Actual output
|| the test Colum can be empty and may be fitted when the requisite test case executed ||
{Generate Paths
∀Paths in paths
{∀ Variable xi in path
{Extract domain of xi
Test value: xi= random (from domain)
}
∀ xi in path
Trace path by putting the values of xi in the above path
Generate result
Expected output= result
}
Return paths
}
```

4. Illustration

The procedure has been explained using Merge Sort. The Program Under Test (PUT) (in this case Merge Sort) is given as follows. This is followed by the generation of the Control Flow Graph (CFG). This is followed by the creation of a minimized CFG. The accomplishment of the above task would be followed by finding of all the possible paths from the starting node to the end node. The variables used and those whose values are changed are found and segregated. Table 2 shows the list of original nodes of CFG which are replaced by the minimized nodes.

4.1. PUT: Merge Sort

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace MergeSort
{
class MergeSortArray
{
void mergeArray(int[] arr, int start, int mid, int end)
{
1 int[] temp = new int[end - start + 1];
2 int i = start, j = mid+1, k=0;
3 while (i <= mid && j <= end)
4 {
5 if (arr[i] < arr[j])
6 {
7 temp[k] = arr[i];
8 k++;
9 i++;
10 }
11 else
12 {
13 temp[k] = arr[j];
14 k++;
15 j++;
16' }
16 }
17 while(i<=mid)
18 {
19 temp[k] = arr[i];
20 k++;
21 i++;
22 }
23 while (j <= end)
24 {
25 temp[k] = arr[j];
26 k++;
27 j++;
28 }
29 k=0;
30 i=start;
31 while (k < temp.Length && i <= end)
32 {
33 arr[i] = temp[k];
34 i++;
35 k++;
36 }
37 }
38 void Mergesort (int[] arr, int start, int end)
39 {
40 if (start < end)
41 {
42 int mid = (end + start) / 2;
43 mergesort(arr, start, mid);
44 mergesort(arr, mid + 1, end);
45 mergeArray(arr, start, mid, end);
46 }
47 }
48 static void Main(string[] args)
49 {
50 int[] arr = {5,9,2,3,6,4,11,10,8,14 };
51 MergeSortArray merge = new MergeSortArray();
51' merge.mergesort(arr, 0, arr.Length-1);
52 foreach (int a in arr)
53 {
54 Console.Write(a + " ");
55 }
56 }
57 }
```

4.2. Control Flow Graph of the above PUT is given in the Appendix 1

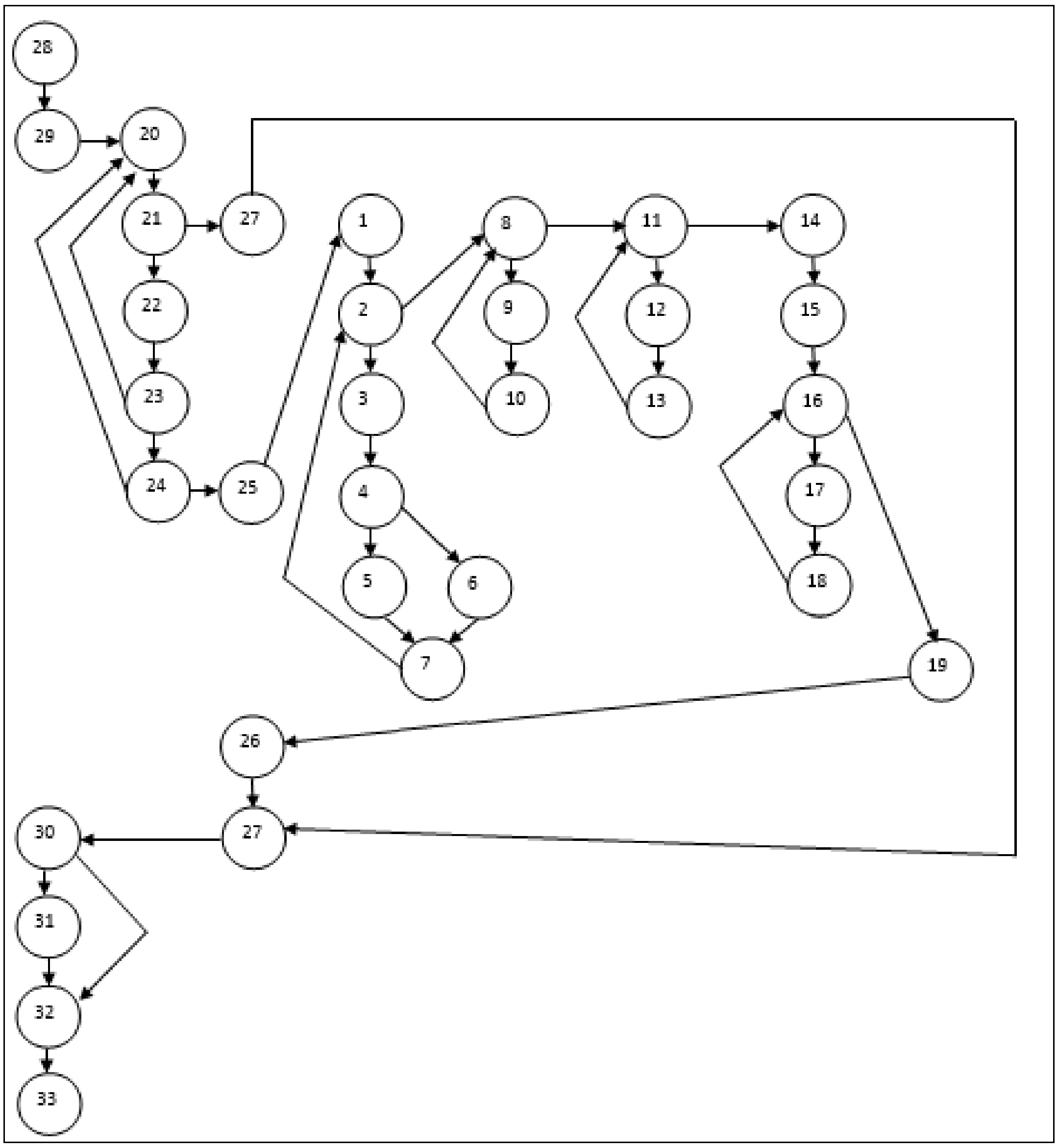

Figure 4. Minimized CFG

Table 2. Nodes Replaced

Previous node	Minimized node	Previous node	Minimized node
1, 2	1	36	18
3	2	37	19
4	3	38	20
5	4	39, 40	21
6, 7, 8, 9, 10	5	41, 42	22
11, 12, 13, 14, 15	6	43	23
16	7	44	24
17	8	45	25
18, 19, 20, 21	9	46	26
22	10	47	27
23	11	48, 49, 50	28
24, 25, 26, 27	12	51, 51’	29
28	13	52	30
29	14	53, 54, 55	31
30	15	56	32
31	16	57	33
32,33,34,35	17		

4.3. Example Paths

1. 28-29-20-21-27-30-31-32-33
2. 28-29-20-21-22-23-24-25-1-2-3-4-5-7-11-12-13-14-15-16-17-18-19-26-27-30-31-32-33
3. 28-29-20-21-22-23-24-25-1-2-3-4-6-7-8-11-12-13-14-15-16-19-26-27-30-31-32-33
4. 28-29-20-21-22-23-24-25-1-2-3-4-6-7-2-8-9-10-8-11-12-13-11-14-15-16-17-18-16-19-26-27-30-31-32-33

4.4. Crafting of Test Cases

In order to find random values from domain, Gas have been used in which a population is generated. It is mapped with the domain. This is followed by crossover to generate new Chromosomes & mutation to break the local maximum. Finally process stops when the value of the fitness function becomes constants

Table 3. Test Data Generated

PATH	INPUT		OUTPUT (SORTED ARRAY)
	N	UNSORTED ARRAY	
1	1	4	4
2	2	2, 4	2, 4
3	2	4, 2	2, 4
4	2	2, 4, 3	2, 3, 4

5. Results and Conclusion

The work presents a novel method of generating test cases using Gas, considering the branch coverage criteria. The method has been explained and exemplified in the above work. The work successfully extends our previous work. It may also be stated here that the technique is applicable for conditional constructs, loops and nested controls. The above technique will now be implemented to medium software of around 4K lines of code and the test cases would be compared to those generated in the testing phase. These cases will also be judged on the basis of their ability to find bugs. The extension of this work would also include the concept of procedure calling.

References

[1] Strigini. L, Bertolino A, “Use of testability measures for depenmdability assessment,” *IEEE Transaction Software Enginnering*, 1996.

[2] Singla N, Bhasin H, “Cellular Genetic Test Data Generation,” *ACMSIGSIOFT Software Engineering Notes*, 2013.

[3] Harman. M, Binkley D, Tonella. P, McMinn Phill, “Species for path approach search based test data generation,” *ACM.*

[4] Bhasin. H, “Artificial life and cellular automata based automata test case generator,” *ACM SIGSOFT Software Engineering Notes*, vol. 39, 2014

[5] Goldberg. D. E, “Simple Genetic Algorithm and minimal deceptive problem,” *University of Albana*, 1986.

[6] Singla. N. Bhasin H, “Genetic based algorithm for N-Puzzle problem,” *International Journal of Computer Application*, pp. 44-50.

[7] Mahajan. R. Bhasin H, “Genetic algorithm based solution to maximum clique problem,” *International Journal of Computer Science and Engineering*, pp. 443-1448.

[8] S. Mehmood, “Systematic Review of Automatic Test Data Generation TEchnique,” 2007.

[9] Shrivastava P. R, Kim. T, H, “Application of GA in Software Testing,” *International Journal of Software Engineering and its Application*, 2009.

[10] Sthamer. H, “automatic generation of software test data usoing GA,” *PhD Thesis university of Glamorgan Pontypridd Wales*, 1996.

[11] Yeh P. L. Lin J C, “ATDG for path testing using GA,” *Information Science*, 2001.

[12] Micheal, McGraw G E, Schatz M A, Walton C C. Christoph C, “GA for Dynamic TDG,” *In Proceedings of 12th international Conference ON Automated Software Engineering*, 1997.

[13] Beresel A, Sthamer H, Wegener J, “Evolutionary test enviorment for automatic structural testing,” *Journal of Information and Software Technology*, 2001.

[14] Zhang W, Yao X. Gong D, “Evolutionary generation of test data for many paths coverage based on groupings,” *Journal of System and Software*, 2011.

[15] Tuya J, Adenso D. B, Blanco R, “ATDG using a scatter search approach,” *Information and Software Technology*, 2009.

[16] Girgis M, “ATDG for Data Flow Testing using genetic algorithm,” *Journal of Universal Computer Sciences*, 2005.

[17] Tuya J, Blanco R, Dolado J. J, Diaz E, “A tabu search algo for structural software testing,” *Computers and Operations Research*, 2008.

[18] Sagama. R, Yao. X, Becerra R.L, “evaluation of differential evolution in software TDG,” *IEEE Conference on Evolutionary Computation*, 2009.

Appendix 1: Control Flow Graph of PUT

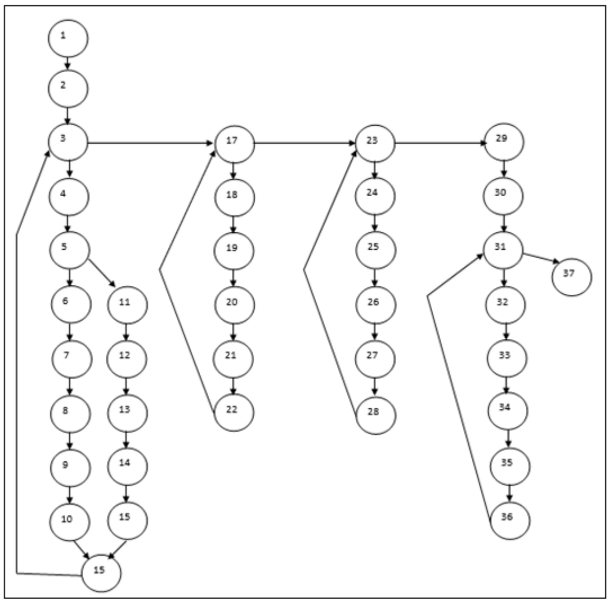

Figure 1. Merge Array

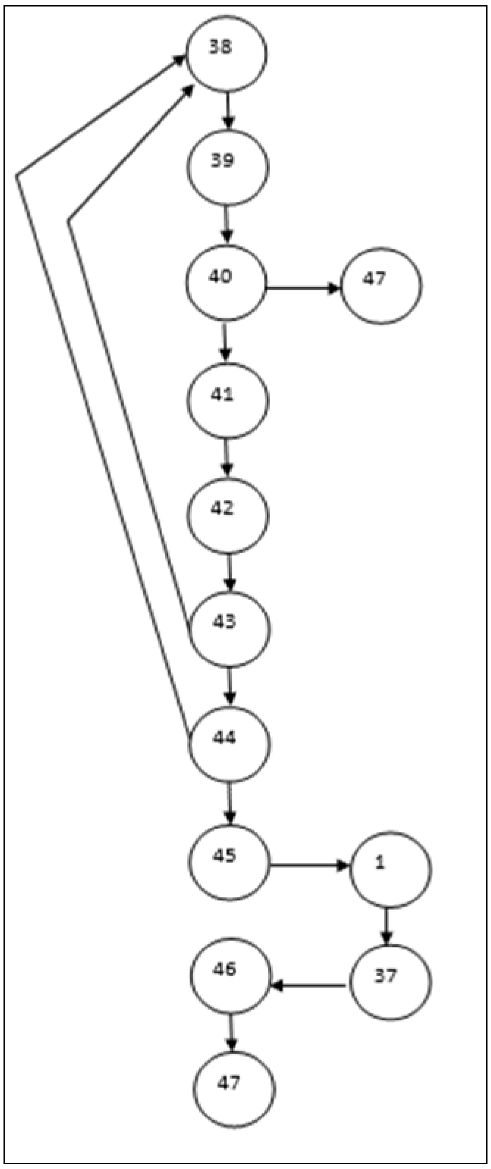

Figure 2. Merge Sort ()

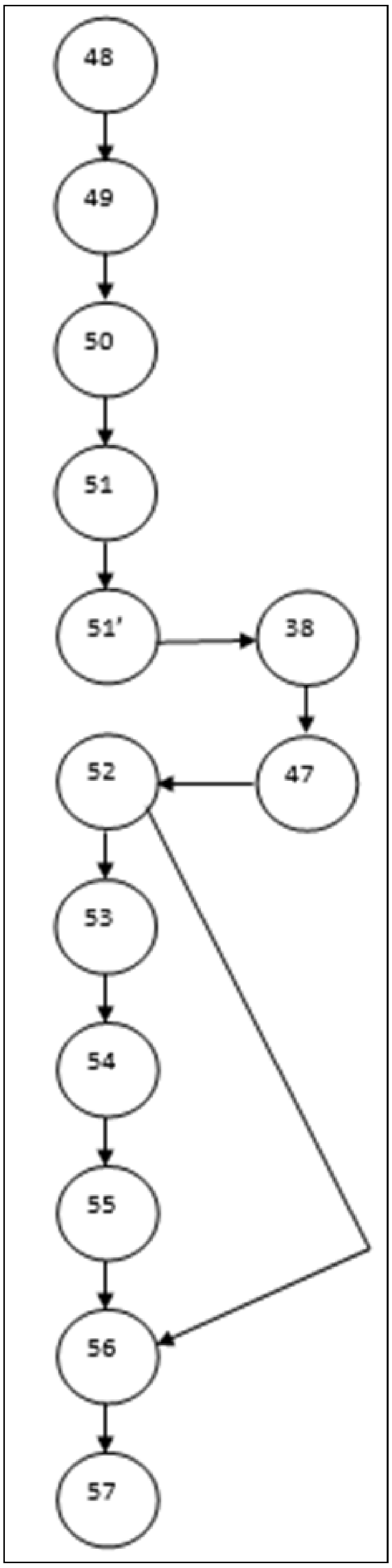

Figure 3. Main ()