International Journal of Applied Engineering Research ISSN 0973-4562 Volume 11, Number 1 (2016) pp 473-478

Automatic Generation of Test Cases for Data Flow Test Paths Using K-Means Clustering and Generic Algorithm

Rijwan Khan
Research Scholar, Department of Computer Engineering
Jamia Millia Islamia, Jamia Nagar, New Delhi, India.
E-mail: rijwankhan786@gmail.com

Mohd Amjad
Assistant Professor, Department of Computer Engineering
Jamia Millia Islamia, Jamia Nagar, New Delhi, India.
E-mail: mamjad@jmi.ac.in

Abstract
Automatic test case generation is one of the major difficulties in software testing that satisfy a given adequacy criterion. This paper presents an evolutionary structural testing approach to automatically generate test cases. The genetic algorithm (GA) is an evolutionary algorithm which is used here for the generation of test cases automatically to cover its def-use associations. The GA conducts its search by constructing new test data from previously generated test data that are evaluated as effective test data. In this paper the proposed algorithm will produces a set of random generated test cases within a domain, these test cases have been divided in different groups (clusters) using K-Mean clustering algorithm. Each cluster find a group of path covered, refined these clustered based on path covered and apply genetic algorithm to produce new test cases in each cluster for increasing the path coverage.

Keywords: Automatic Test Case Generation, Genetic Algorithm, Path Coverage Testing, K-Means Clustering.

Introduction

Test data generation and application of test data adequacy criterion are two main aspect of software testing process. Test data generation is a tool that generate the test cases and to insure that testing process has been finished is an adequacy [1]. The major difficulty which has been identified in software testing is automatic generation of test cases for satisfaction of a given adequacy criteria. Control flow based and data flow based test adequacy criteria have been already proposed. In this paper data flow based testing has been considered.

Testing basically include input domain and cross ponding outputs depend on these inputs. Testing all possible inputs clear the picture of program behaviour. It is not possible to take all the inputs related to a program, so it's a clear message to take a small input domain to test the program. This input domain produce the general behaviour of the program. Test data selection criteria is based on measurement of code coverage. These criteria are statement coverage and branch coverage [2]. This paper is based on branch coverage. Data flow analysis focuses on how variables are bound to values, and how these variables are to be used. The data flow criteria presented in this paper take the input randomly, following them as they are modified using genetic algorithm, until they are ultimately used to produce desire output.

Search Based Software Testing (SBST)

Search based software testing techniques are very much popular in recent years. As interest of automatic software testing industries are growing these days, the search based software testing is also growing accordingly. This paper used genetic algorithm for the automation of test cases. Search based optimization techniques have been applied to a number of software engineering activities such as requirement engineering, project planning and cost estimation through testing, to automate maintenance, service oriented software engineering.

Path Coverage Testing

In this part all-uses and data flow criteria have been described. The control flow of any program can be represented by set of nodes and edges. Each individual node is combination of some statements and each individual edge between two nodes is transfer of control between these two nodes. A path is simply finite number of nodes connected with edges. A completed path in that with start form the first node, travels some nodes between first and last node and end on last node. Data flow analysis depends on interaction between variables definitions and references. The definition of variable is called *def* and reference is *use*. The use of the variable can be split in two parts c-use (used in computation) and p-use (predicate use) [3].

Genetic Algorithm

Genetic algorithms (GAs) is proposed by Holland in his book Adaptation in Natural and Artificial Systems in 1975 on the concept of survival of fittest. GAs are applied on the variety of problems such as search based, optimization problems and machine learning with AI domain. GAs search techniques are rooted in mechanism of assessment and nature genetics. The survival of fittest individuals comes from GAs inspiration form the natural search and selection processes. Each individual is represented by chromosome, GAs select these

chromosome to generate a sequence of populations by using a different selection mechanism.
Genetic Algorithms have the following basic operations

- Selection
- Reproduction
- Evolution
- Replacement

The first operation is selection in which individuals are selected for reproduction. So many selection operation already defined, anyone can be applied for selection of individual. These selection operations are Roulette Wheel Selection, Linear Ranking Tournament Selection etc. In reproduction offspring are bred with selected individuals. Crossover and Mutation operations are used in reproduction. In crossover operation two individuals exchange some information (if individuals are represented in binary then some binary numbers) at one point or at multiple points. While in mutation only one bit has been change. The evaluation process checks the fitness on new generated individuals after reproduction. In replacement the new individuals takes the position of old individuals. The old individuals are killed in this process.

```
Simple Genetic Algorithm ( )
{
        Select the individuals as initial population.
        Evaluated this population.
        While Termination Criteria is not matched
        {
            Select the solution for the next generation.
            Perform crossover and mutation operations.
            Evaluate population.
        }
}
```

This algorithm will stop when fitness meet or until a maximum number of iteration has been taken.

Related Work

In software testing process, control flow graph has been designed and found all the paths from start to the end of the program. Some inputs have given at the start of the program, found variable declaration and used of given variables. These variables were declared at some nodes and used on other nodes in a particular path. All those paths have been identified for all def-use of the variables [2]. An automatic test data generation technique had been applied that used a genetic algorithm (GA), guided by the data flow dependencies in the program, to search for test data to cover its def-use associations [3]. Multi population genetic algorithm had been used as a search technique. Randomly initialization of sub-population had been done and some different sub problems worked separately with fitness function and have different search in search space [4]. Some techniques are depend on data dependence analysis, the data dependence analysis automatically identifies statements. These identified statement effects the execution of the selected program [5]. In recent years some techniques has been proposed which are based on genetic algorithm to generate the test data. [6-8].

Dynamic Domain Reduction

DDR procedure is an automatic test case generating method that uses constraints derived from the test program that control the execution paths in a CFG to reduce domains of variables progressively until test data that satisfy these constraints are found for the test program. The DDR has the several information like a data flow graph, initial domains for the input variables and start and end nodes in the data flow graph. In DDR finite set of paths have been found and then applied these inputs to find the predicate of the variable and reduced the size of input domain step by step. Reducing the size of domain is also a search based problem. The DDR has several drawbacks that prevent it from working in some situations.

Proposed Method

This paper presents an automatic test case generation technique using genetic algorithm which first divide the input domains to sub-domains and applied a check on these sub-domains to cover paths. If all paths have not covered then applied genetic algorithm of each sub- domain to generate the new test cases based on the previous cover. When the 100% coverage comes from all subdomains this algorithm will be stopped.

A. Proposed Algorithm

Step 1: Take the initial inputs domain.
Step 2: Randomly generate test cases in the given domain.
Step 3: Make the different clusters for inputs given in a domain using K-Means algorithm.
Step 4: Apply inputs of clusters to the CFG.
Step 5: Check the path coverage for each cluster.
Step 6: If the path coverage of all given clusters are 100% then stop. Otherwise go to next step.
Step 7: Choose the clusters not having 100% path coverage.
Step 8: Select those clusters which covered minimum paths. Check if these paths are also covered in any one cluster then drop these clusters. Go to next step.
Step 9: Apply the Genetic Algorithm operation to generate the new test cases within the clusters and go to step 6.

Experimental Setup

Taken an example of x^y for implementing of given proposed method.

```
1.  Power (x, y)
2.  if(x==1)
3.  return 1
4.  if(y==1)
5.  return x
6.  i=1,t=1
7.  while(i<=y)
8.  { t=t*x
9.  i=i+1}
10. return t
```

Now designing the DFD for this algorithm X^Y. All the statements of the algorithm are represented by nodes in the DFD. The edges in the DFD shows control flow among the statements. DFD for this algorithm is given in figure 1.

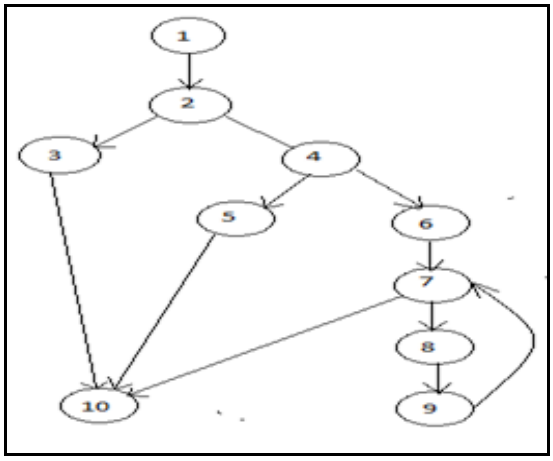

Figure 1: DFD for the program X^Y

In the example of x^y total number of the paths are 4.

These are
Path 1: 1-2-3-10
Path 2: 1-2-4-5-10
Path 3: 1-2-4-6-7-8-9-7-10
Initially some random value of x and y are generated.
Example we have generated 47 values of x and y with in the input domain 1 to 15.
These values are

TABLE 1: X^Y

S. No	X	Y	X^Y
1	2	3	8
2	1	4	1
3	6	7	279936
4	9	3	729
5	2	5	32
6	10	5	100000
7	8	3	512
8	4	9	262144
9	7	6	117649
10	15	5	759375
11	2	4	16
12	12	6	2985984
13	6	7	279936
14	8	5	32768
15	5	6	15625
16	5	5	3125
17	6	6	46656
18	7	7	823543
19	3	7	2187
20	1	9	1
21	1	8	1
22	4	12	16777216
23	13	3	2197
24	2	10	1024
25	3	14	4782969
26	4	6	4096
27	7	6	117649
28	2	7	128
29	3	9	19683
30	12	1	12
31	13	2	169
32	14	3	2744
33	15	4	50625
34	11	5	161051
35	10	3	1000
36	9	7	4782969
37	8	8	16777216
38	7	9	40353607
39	6	10	60466176
40	5	11	48828125
41	4	13	67108864
42	3	14	4782969
43	2	15	32768
44	1	15	1
45	2	7	128
46	6	8	1679616
47	8	1	8

Now let there are 4 clusters G1,G2,G3,G4 and has the centroids (2,3), (1,4), (6,7) and (9,3).

After applying K-Means clustering. Each cluster G1, G2, G3 and G4 has the following sets of inputs.
G1= (1,2,5,11,16,19,20,21,26,28,45)
G2= (22, 24, 25, 40, 41, 42, 43, 44)
G3= (3,8,9,13,14,15,17,18,27,29,36,37,38,39,46)
G4= (4,6,7,10,12,23,30,3,1,32,33,34,35,47)

TABLE 2: CLUSTER 1

S. No	X	Y	X^Y
1	2	3	8
2	1	4	1
5	2	5	32
11	2	4	16
16	5	5	3125
19	3	7	2187
20	1	9	1
21	1	8	1
26	4	6	4096
28	2	7	128
45	2	7	128

TABLE 3: CLUSTER 2

S. No	X	Y	X^Y
22	4	12	16777216
24	2	10	1024
25	3	14	4782969
40	5	11	48828125
41	4	13	67108864
42	3	14	4782969
43	2	15	32768
44	1	15	1

TABLE 4: CLUSTER 3

S. No	X	Y	X^Y
3	6	7	279936
8	4	9	262144
9	7	6	117649
13	6	7	279936
14	8	5	32768
15	5	6	15625
17	6	6	46656
18	7	7	823543
27	7	6	117649
29	3	9	19683
36	9	7	4782969
37	8	8	16777216
38	7	9	40355607
39	6	10	60466176
46	6	8	1679616

TABLE 5: CLUSTER 4

S. No	X	Y	X^Y
4	9	3	729
6	10	5	100000
7	8	3	512
10	15	5	759375
12	12	6	2985984
23	13	3	2197
30	12	1	12
31	13	2	169
32	14	3	2744
33	15	4	50625
34	11	5	161051
35	10	3	1000
47	8	1	8

Applying the elements of each cluster to the CGF x^y.

TABLE 6: PATH COVERAGE FOR EACH CLUSTER

Clusters	Input Data	Paths	Coverage
Cluster 1	(2,3),(1,4),(2,5),(2,4), (5,5),(3,7),(1,9),(1,8), (4,6),(2,7),(2,7)	3,1,3,3,3, 3,1,1,3,3, 3	67%
Cluster 2	(4,12),(2,10),(3,14), (5,11),(4,13),(3,14), (2,15),(1,15)	3,3,3,3,3, 3,3,1	67%
Cluster 3	(6,7),(4,9),(7,6),(6,7), (8,5),(5,6),(6,6),(7,7), (7,6),(3,9),(9,7),(8,8), (7,9),(6,10),(6,8)	3,3,3,3,3, 3,3,3,3,3	33%
Cluster 4	(9,3),(10,5),(8,3),(15,5), (12,6),(13,3),(12,1), (13,2),(14,3),(15,4), (11,5),(10,3), (8,11)	3,3,3,3,3, 3,2,3,3,3, 3,3,3	67%

Cluster 3 has covered only path 3 while path 3 is also covered in all the clusters so drop it. Cluster 1 and Cluster 2 have the same path coverage so drop cluster 2. After refined only two clusters are remaining, these are cluster 1 and cluster 4.

TABLE 7: CLUSTERS AFTER K-MEANS ALGORITHM

Clusters	Input Data	Paths	Coverage
Cluster 1	(2,3),(1,4),(2,5),(2,4), (5,5),(3,7),(1,9),(1,8), (4,6),(2,7),(2,7)	3,1,3,3,3, 3,1,1,3,3,3	67%
Cluster 4	(9,3),(10,5),(8,3), (15,5),(12,6),(13,3), (12,1),(13,2),(14,3), (15,4),(11,5),(10,3), (8,11)	3,3,3,3,3, 3,2,3,3,3, 3,3,3	67%

Now applying GA operations if coverage is less than 75%, then apply crossover operation otherwise apply mutation operation.

TABLE 8: GA OPERATIONS FOR CLUSTERS

Clusters	Input Data	Paths	Operation	Coverage
Cluster 1	(2,3),(1,4),(2,5), (2,4),(5,5),(3,7), (1,9),(1,8),(4,6), (2,7),(2,7)	3,1,3, 3,3,3, 1,1,3, 3,3	Crossover	67%
Cluster 4	(9,3),(10,5),	3,3,3,	Crossover	67%

	(8,3),(15,5), (12,6),(13,3), (12,1),(13,2), (14,3),(15,4), (11,5),(10,3), (8,11)	3,3,3, 2,3,3, 3,3,3, 3		

After applying Genetic algorithm in MATLAB individual are selected form cluster 1 and cluster 4 for the reproductions are given in figure below. After 51 generation as given in figure 2 and figure 4 each cluster changed its input data as given in figure 3 and 5. To apply these new data on the x^y example all the path covered and 100% path coverage is completed.

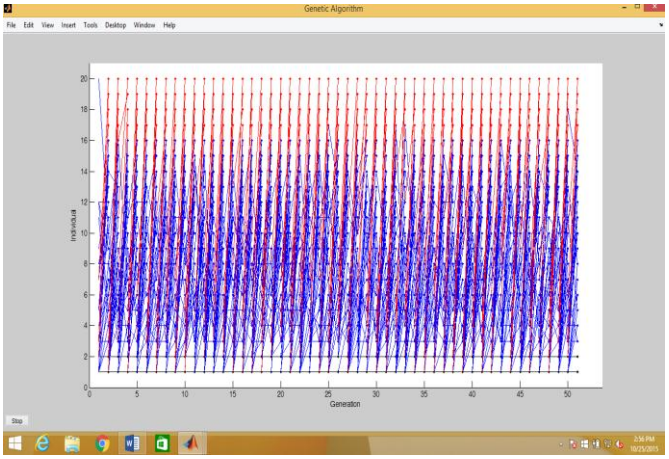

Figure 2: Generation of Individuals for Cluster 1

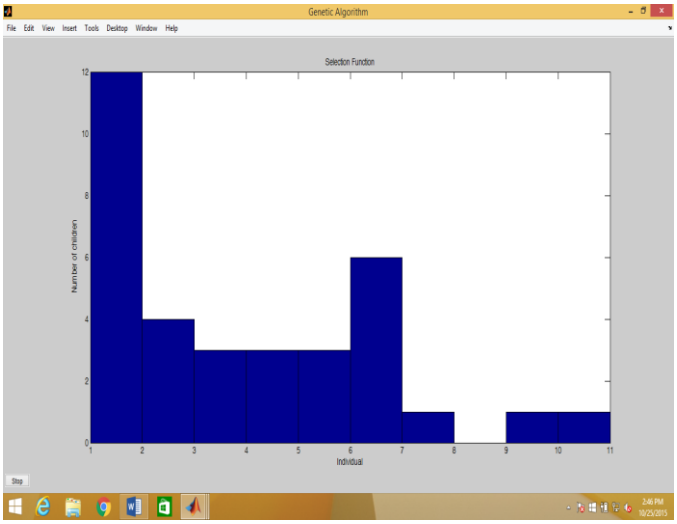

Figure 3: Selection Function for Cluster 1.

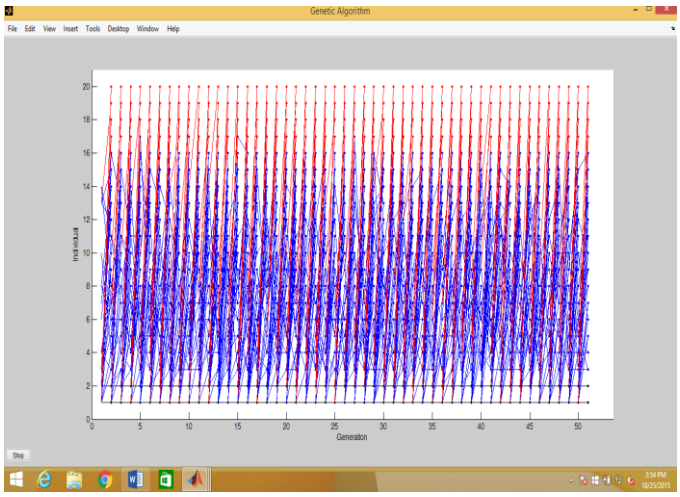

Figure 4: Generation of Individuals for Cluster 4

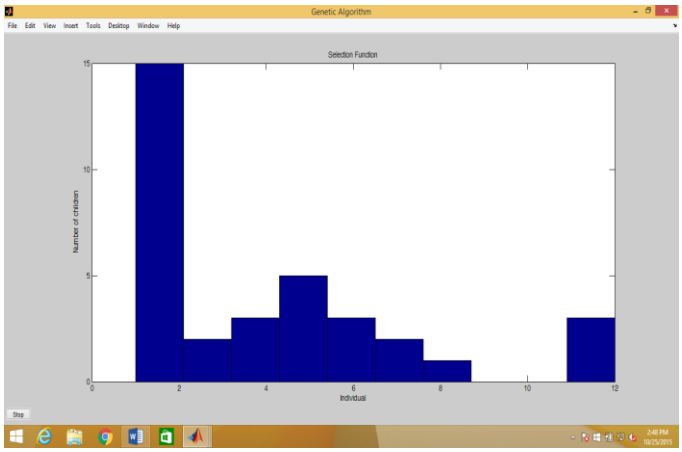

Figure 5: Selection Function for Cluster 4

Conclusion

This paper presented a technique which uses K-Means clustering algorithm to divide the input domains in K individual clusters. Each cluster has a centroid, depends on the distance from the centroid, K clusters have been designed. The input from each cluster is applied for the path coverage. On the basis of their coverage criteria clusters are chosen for the Genetic Algorithm (GA) operations. With the help of the Genetic Algorithm new inputs are generated for path coverage. In this paper Genetic Algorithm gave 100% path coverage over the random generation.

References

[1] Frankl, Phyllis G., and Stewart N. Weiss. "An experimental comparison of the effectiveness of branch testing and data flow testing." Software Engineering, IEEE Transactions on 19.8 (1993): 774-787.

[2] Rapps, Sandra, and Elaine J. Weyuker. "Selecting software test data using data flow information." Software Engineering, IEEE Transactions on 4 (1985): 367-375.

[3] Girgis, Moheb R. "Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm." J. UCS 11.6 (2005): 898-915.
[4] Minj, Jasmine. "Feasible Test Case Generation using Search based Technique." International Journal of Computer Applications 70.28 (2013).
[5] Korel, Bogdan. "Automated test data generation for programs with procedures."ACM SIGSOFT Software Engineering Notes. Vol. 21. No. 3. ACM, 1996.
[6] Pargas, Roy P., Mary Jean Harrold, and Robert R. Peck. "Test-data generation using genetic algorithms." Software Testing Verification and Reliability 9.4 (1999): 263-282.
[7] Bueno, Paulo Marcos Siqueira, and Mario Jino. "Automatic test data generation for program paths using genetic algorithms." International Journal of Software Engineering and Knowledge Engineering 12.06 (2002): 691-709.
[8] Lin, Jin-Cherng, and Pu-Lin Yeh. "Automatic test data generation for path testing using GAs." Information Sciences 131.1 (2001): 47-64.
[9] Khamis, Abdelaziz, Reem Bahgat, and Rana Abdelaziz. "Automatic test data generation using data flow information." Doğuş Üniversitesi Dergisi 1.2 (2011): 140-153.
[10] Mahajan, Manish, Sumit Kumar, and Rabins Porwal. "Applying genetic algorithm to increase the efficiency of a data flow-based test data generation approach." ACM SIGSOFT Software Engineering Notes 37.5 (2012): 1-5.