Available Online at http://www.journalajst.com

ASIAN JOURNAL OF SCIENCE AND TECHNOLOGY

ISSN: 0976-3376

Asian Journal of Science and Technology
Vol. 6, Issue 04, pp. 1354-1358, April, 2015

RESEARCH ARTICLE

EFFICIENT AUTOMATIC TEST CASE GENERATION FOR DATA FLOW TESTING USING NEIGHBORHOOD CROSSOVER METHOD IN GENETIC ALGORITHM

***Anushree and M. Mohan**

Department of Computer Science and Engineering, SRM University, India

ARTICLE INFO

Article History:
Received 13th January, 2015
Received in revised form
17th February, 2015
Accepted 02nd March, 2015
Published online 30th April, 2015

Key words:
Software Testing, Automatic Test Case Generation, Data Flow Testing, Genetic Algorithm, Neighborhood Crossover.

ABSTRACT

This paper presents the automatic test case generation for data flow testing. This paper applies Genetic Algorithm using neighborhood crossover method which significantly increases the efficiency and reduce the effort. For enhancing the efficiency in generating the test cases this approach includes a mechanism for adapting the range of neighborhoods according to the evolutionary progress. There are three types of neighborhood crossover: 4 neighbor crossover, 3 neighbor crossover and 2 neighbor crossover.

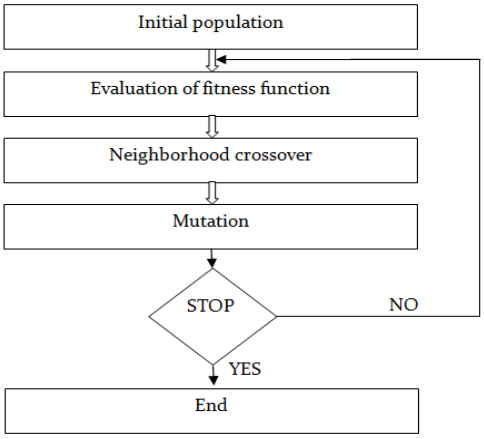
INTRODUCTION

The development of software goes through various phases such as planning, design, coding, testing, deployment and maintenance. In these phases software testing is the most time and effort consuming. Software is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase various types of testing are done. Structural testing is in one of these testing. Data flow testing is considered to be a form of structural testing. Data flow testing is based on selecting paths through the program's control flow in order to explore sequences of events related to the status of data objects. Recently the main focus in testing is generating the test cases automatically. Since manual testing can be laborious and time consuming. In addition, a manual approach might not always be effective in finding certain classes of defects. This technique offers a possibility to perform testing effectively and that can be run quickly and repeatedly. In case of data flow testing, if done manually, testers may not be able to find perfect test cases. The problem in data flow testing is that the testers may have generated repeated test case or have not covered all paths of the CFG (control flow graph) of the program. This paper presents the approach to generate the test cases automatically for data flow testing that uses a neighborhood crossover in Genetic Algorithm.

****Corresponding author: Anushree,***
Department of Computer Science and Engineering, SRM University, India.

Genetic Algorithm includes the 3 operations: Selection, Crossover and Mutation. This approach has used Data dependence analysis which is used to guide the process of test case generation. The approach followed by this paper merges the Selection and Crossover operation by the application of Neighborhood crossover. This approach conducts its search by constructing new test cases from the previous effective test cases. The neighborhood crossover takes the either initial population or fittest population from the previous test data. The neighborhood crossover can be applied on either 4 or 3 or 2 populations. In case if there are n (n>4) fittest test data it will automatically consider 4 populations. This approach can be effectively used for large programs as well as the programs with or without loops and procedures.

Overall Block Diagram



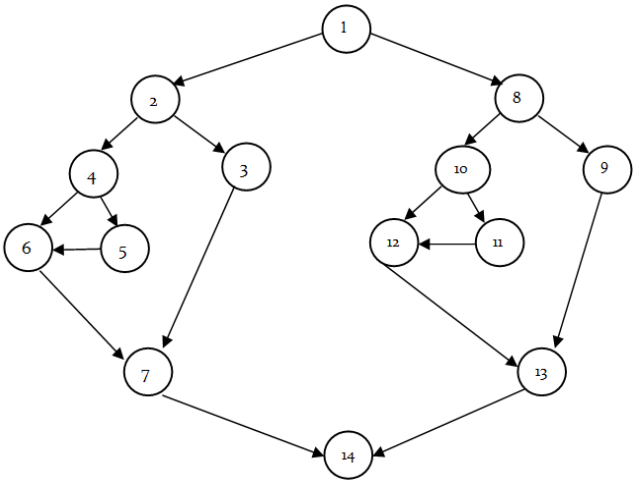
Proposed Methodology Description

Source Code

```
1    1  INTEGER X,Y,Z
2    1  READ(5,*)X,Y,Z
3    1  MID=Z
4    1  IF(Y.LT.Z)THEN
5    2  IF(X.LT.Y)THEN
6    3  MID=Y
7    4  ELSE
8    4  IF(X.LT.Z)THEN
9    5  MID=X
10   6  END IF
11   7  END IF
12   8  ELSE
13   8  IF(X.GE.Y)THEN
          14  9   MID=Y
15   10 ELSE
16   10 IF(X.GT.Z)THEN
17   11 MID=X
18   12 END IF
19   13  END IF
20   14  END IF
21   14  PRINT*,'MIDDLE VALUE= ', MID
22   14  END
```

Source Code to be Tested

(1ST column represent the statement number and 2ND column represent node number in its CFG)



Control Flow Graph of the Source Code

Control Flow Graph

The Control Flow Graph is referred as the graph or tree like structure that states the flow of the inputs and output throughout the source code. This can be defined by nodes and edges and can be explained by the following definition

- Defs- defs states the definition of the input variables and denoted by the nodes of the CFG.
- Uses- uses states the use of the input variable and represented by the edges of the CFG.
- Reach(i)- It defines the set of all variable that reaches to node i.
- Avail(i)- It is set of all the variables defs that are available at node i.
- C-use(i)- It is the set of variables for which node i contains a global c-use.
- P-use(i)- It is the set of variables for which edge(i,j) contains a p-use.
- Dcu(i)- dcu(i) is obtained by
- Dcu(i)=reach(i) ∩ c-use(i)
- Dpu(I,j)- it is obtained by
- Dpu(i,j)=avail(i) ∩ p-use(i,j)

Initial Population

The proposed methodology is based upon the chromosomes. Each chromosome is a test case which is represented by the binary string. Initially there is take pop_size (no of chromosomes) chromosomes. Each chromosome is a string of length of m (bits). Further each chromosome is converted into k decimal numbers where k is the number of inputs in the given example program.

Neighborhood Crossover Method

Once the chromosomes are taken, it is needed to act upon these chromosomes. Here the proposed methodology is going to apply. Crossover is the operator applied in the Genetic algorithm. Crossover is a process that exchanges the substring of the two parent chromosomes at a random position pos . The number pos is referred as the crossing point from where the bits of the chromosomes will be swapped.

Suppose there are two parent chromosomes are
110100110
100110100
And the position pos is 6 then the offspring will be
110100100
100110110

The proposed methodology of crossover is Neighborhood Crossover. Neighborhood crossover is done by following three types

- 4 Neighbor crossover
- 3 Neighbor crossover
- 2 neighbor crossover

These crossovers are based upon the numbers of chromosomes. Suppose there are n chromosomes and n>4 then by the fitness function it will select 4 fittest chromosomes and will apply 4 Neighbor crossover. If there are 3 fittest chromosomes, 3 Neighbor crossover will be applied otherwise 2 Neighbor crossover will be applied.

a. 4 Neighbor crossover

b.

If there are 4 chromosomes to be crossed over then this method is applied. This method is applied in the following way:

Suppose there are 4 chromosomes a, b, c and d and the crossing point is pos the

ab1 & ab2	a	ad1 & ad2
b	crossover position pos	d
bc1 & bc2	c	cd1 & cd2

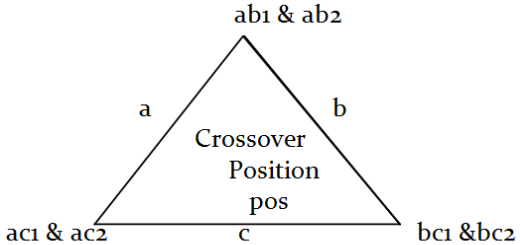
4 Neighbor crossovers

In the above mentioned method

ab1 and ab2 are offspring obtained from the crossover of a and b at crossover position pos,
bc1 and bc2 are offspring obtained from the crossover of b and c at crossover position pos,
ad1 and ad2 are offspring obtained from the crossover of a and d at crossover position pos,
cd1 and cd2 are offspring obtained from the crossover of c and d at crossover position pos.

3 Neighbor crossover

If there are three fittest chromosomes then this method can be applied in the following way:



3 Neighbor crossover

In the above mentioned method

ab1 and ab2 are offspring obtained from the crossover of a and b at crossover position pos,
bc1 and bc2 are offspring obtained from the crossover of b and c at crossover position pos,
ac1 and ac2 are offspring obtained from the crossover of a and c at crossover position pos.

2 Neighbor crossover

If there are 2 fittest chromosomes then this method is applied. According to this method if there two chromosomes a and b they will be simply crossed over at position pos and the new off springs will be ab1 and ab2.

Mutation

Mutation is another operator applied by the Genetic Algorithm and always operated after the crossover operation. Mutation operation just flips the bits of the given chromosome from 0 to 1 or vice versa. In any chromosome every bit has an equal chance to undergo mutation.

Representation

The proposed method uses the m bit binary string which is named as chromosome and act as a test case for the program. To implement the test case we need to convert into the decimal. Each chromosome is converted into k decimal numbers where k is the number of the inputs in the program. For each decimal number there is given the domain and precision. Domain is represented by $D_i[a_i,b_i]$ and states the length of decimal number as b_i-a_i . Precision is represented as d_i.

Suppose a chromosome is divided into k substring where k is the no of input variable and each substring has length m_i

where i=1,2…….k

and $(b_i-a_i)*10^{di} \leq 2^{mi}-1$

Then the each variable is converted into decimal form by the following formula:

$$x_i=a_i+x_i' \cdot \frac{b_i-a_i}{2^{mi}-1}$$

Fitness Value

The whole genetic algorithm is based upon "survival of the fittest". It means that the fittest chromosome will be able to proceed further. The fittest chromosome is obtained from its fitness value. The fitness value of the chromosomes states if it will survive anymore or not.

The fitness value is represented by eval(v_i) for the chromosome v_i where i=1,2……pop_size. Its is calculated as

$$Eval(v_i)= \frac{\text{No. of def-use paths covered by } v_i}{\text{Total no. of def-use path}}$$

Overall Algorithm

The proposed method is applied by the following algorithm

Input

Program P to be tested
List of def-use path to be covered
Crossover probability
Mutation probability
Max_gen
No. of input variables
Domain and precision of input variables
Population size pop_size

Output

Set of test cases
List of uncovered def-use path , if any

Process

Initial population
Def-use coverage vector=0
4 neighbor crossover
Mutation
Convert current chromosomes in decimal

Execute program P with these test data
Evaluate these test data
If(some def-use path is covered)
$Case_n = Case_n + 1$
Add this test case to the list of test cases
Update def-use coverage vector
Update def-use coverage percentage
End if
While(coverage percentage□≠□□□□□and no. of generation ≤max_gen)
Find X effective chromosome
{
If(X <4)
Select 4 fittest chromosome
Apply 4 neighbor crossover
Else if(pop_size=3)
Apply 3 neighbor crossover
Else
Apply 2 neighbor crossover
}
Mutation
Convert current chromosomes in decimal
Execute program P with these test data
Evaluate these test data
If(some def-use path is covered)
$Case_n = Case_n + 1$
Add this test case to the list of test cases
Update def-use coverage vector
Update def-use coverage percentage
End if
End while

Example

To illustrate the algorithm, here is given the solution that uses the approached methodology. We have implemented the above example with the approached methodology.

Pop_size : 4

Crossover probability: 0.8
Mutation probability: 0.15
No. of input variables: 3
Domain and precision of input variable: 1-20, 0; 1-20, 0; 1-20, 0
Initially population

a. 000011100101110
b. 111111111010101
c. 110001110011110
d. 101101011010100

1. 000011111010101 2. 111111000101110	000011100101110 a	5. 000011100101110 6. 101101100101110
111111111010101 b	crossover point 5	d 101101011010100
3. 111111110011110 4. 110001111010101	c 110001110011110	7. 101101110011110 8. 110001011010100

Mutation

Mutation point= 5

1. 000001111010101 1,19,14
2. 111101100101110 19,16,10
3. 111101110011110 19,18,19
4. 110011111010101 16,18,14
5. 000001100101110 1,14,13
6. 101111100101110 15,16,10
7. 101111110011110 15,18,19
8. 110011011010100 16,14,13

Case 1. 1,19,14
Traversed path- 1,2,4,6,7,14
Dpu path- 1,2,7,8,11,12
Dcu path- 5
Def-use coverage=24.13%
Accumulated Def-use coverage=24.13%

Case 2. 19,16,10
Traversed path- 1,2,3,7,14
Dpu path- 5,6
Dcu path- 6,1
Def-use coverage= 13.79%
Accumulated Def-use coverage= 37.92%

Case 3. 19,18,19
Traversed path- 1,8,10,11,12,13,14
Dpu path- 3,4,15,16,17,18
Dcu path- 4,9
Def-use coverage=27.58%
Accumulated Def-use coverage= 65.5%

Case 4. 16,18,14
Traversed path- 1,2,4,5,6,7,14
Dpu path- 9,10
Dcu path- 2,7
Def-use coverage= 13.79%
Accumulated Def-use coverage= 79.30%

Case 5. 1,14,13
Traversed path- 1,8,10,12,13,14
Dpu path- 19,20
Dcu path- no path
Def-use coverage= 6.90%
Accumulated Def-use coverage= 86.20%

Case 6. Not selected

Case 7. 15,18,19
Traversed path- 1,8,9,13,14
Dpu path- 13,14,
Dcu path- 8,3
Def-use coverage= 13.80%
Accumulated Def-use coverage= 100%

Generated test cases

1,19,14
19,16,10
19,18,19
16,18,14
1,14,13
15,18,19

Table. The def-use coverage vector of the source code

DCU path	1	2	3	4	5	6	7	8	9	
Test Case	2	4	7	3	1	2	4	7	3	
DPU path	1	2	3	4	5	6	7	8	9	10
Test case	1	1	3	3	2	2	1	1	4	4
DPU path	11	12	13	14	15	16	17	18	19	20
Test case	1	1	7	7	3	3	3	3	5	5

Table: Comparison of results of the test case generation from Genetic algorithm and Genetic Algorithm with Neighborhood Crossover for the above source code

Method	No. of path case generated	No. of test case generated	Mutation probability P_m	Crossover probability p_c	Crossover operation	Mutation operation	D-U path coverage
Genetic Algorithm	12	6	0.15	0.8	4 offspring	56 offspring	100%
Genetic Algorithm with Neighborhood Crossover	7	6	0.15	0.8	8 offspring	112 offspring	100%

RESULTS

The results shown here are given by comparing the simple genetic Algorithm and the Genetic Algorithm with Neighborhood Crossover.

Conclusion and Future Work

The result shows that the approached methodology is more efficient as compare to simple Genetic Algorithm. It took less number of path case generations for finding the test cases. This can be more efficient if fuzzy logic is applied along with Genetic Algorithm for test case generation for data flow testing.

REFERENCE

Chayanika Sharma, Sangeeta sabharwal, and Ritu Sibbal, 2013. "A Survey On Software Testing Techniques using Genetic Algorithm", *IJCSI International Journal of Computer Science Issues*, Vol 10, Issue 1, No 1, January.

D.J Berndt, A. Watkins, "High volume software testing using genetic algorithms", Proceedings of the 38th International Conference on system sciences (9), IEEE, 2005, pp. 1- 9.

Girgis, "Automatic test generation for data flow testing using a genetic algorithm*", Journal of computer science*, 11 (6), 2005, pp. 898 – 915.

McMinn, "Search based software test generation: A survey", Software testing, Verification and reliability 14 (2), 2004, pp. 105-156.

Mark Last et. al., "Effective black-box testing with genetic algorithms", Lecture notes in computer science, Springer, 2006, pp. 134 -148.

Maha alzabidi et. al., "Automatic software structural testing by using evolutionary algorithms for test data generations", *International Journal of Computer science and Network* Security 9 (4), 2009, pp. 390 – 395.

Naresh Chauhan, Software Testing: Principles and Practices, Oxford University Press, 2010.

Sthamer "The Automatic Generation Of Software Test Data Using Genetic Algorithms". Phd thesis, university of Glamorgan, Pontyprid, wales, Great Britain, 1996.

Stefan Wappler, Frank Lammermann, "Using Evolutionary Algorithms For Unit Testing Of Object Oriented Software" GECCO. ACM, 2005, pp.1925 - 1932.

Sangeeta sabharwal et. al., "Prioritization Of Test Case Scenarios Derived From Activity Diagram Using Genetic Algorithm". ICCCT . IEEE, 2010, 481- 485.

Sangeeta sabharwal et. al., "Applying Genetic Algorithm For Prioritization Of Test Case Scenarios Derived From Uml Diagrams" *International journal of computer science issues* 8 (3), 2011, 433 - 444.

Timo Mantere, "Automatic software testing by Genetic Algorithms" Phd thesis, University of Vaasa, Finland, 2003.

Velur Rajappa et. al., "Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory" International Conference on emerging trends in Engineering and Technology, IEEE, 2008, pp. 298 - 303.

Xuan Peng, Lu Lu, "A New Approach For Session - Based Test Case Generation By Ga". IEEE, 2011, pp. 91- 96. IJCSI *International Journal of Computer Science Issues*, Vol. 10, Issue 1, No 1, January 2013 ISSN (Print): 1694-0784 | ISSN (Online): 1694-0814 www.IJCSI.org 393

****