Introduction to Data Flow Testing with Genetic Algorithm

Rijwan Khan
Department of Computer Engineering
Jamia Millia Islamia
New Delhi

Mohd Amjad
Department of Computer Engineering
Jamia Millia Islamia
New Delhi

ABSTRACT
Control flow diagrams are a keystone in testing the structure of software programs. With the help of control flow between the various components of the program, we can select the test cases in a particular domain. In this paper, we introduced a window-based tool for generating the CFG of a C Program automatically. The data flow testing, i.e., control flow testing depends on all def-use of the variables. So selecting the test cases for a particular data flow diagram is not an easy task. In this paper genetic algorithm has been used to generate the test cases automatically for data flow testing.

Keywords
Data-Flow Testing, Control-Flow Graph, Genetic Algorithms, Software Testing, Automatic Test Cases.

1. INTRODUCTION
Software testing is a process of analysing a software and also detect the errors that are in the software. Software testing is a process of ensuring that the developed software is bugs free and fulfill all the requirement of the customer. There are two main types of the software testing; these are black box testing and white box testing. The purpose of black box testing is only to check desired output for a given input. Black box testing is that which checks only the output you want the software/program. It is a validation technique in which tester verifies that software/program meets all the customer requirements or not. In white box testing, the tester has to test the functioning of the functions defined in the program/software. Data flow testing is a part of white box testing technique in which all du-paths are checked and find the path coverage for the software/program in its CFG.

In this paper, data flow testing is applied with the genetic algorithm. For data flow testing three steps are taken, the first step is to generate the control flow graph (CFG) for the program/software, the second step is to find all def-use of the variables and the third is to find the path coverage in the CFG. In the third step test cases plays an important role.

2. LITERATURE SURVEY
Meta-heuristic techniques particular the genetic algorithm has been applied extensively to generate test cases automatically. One of the problems faced is that the population may not contain any individual that encodes test data for which the execution path reaches the predicate node of the target branch. For solving this problem, three methods have been introduced DFS, BFS and path prefix strategy [1]. An open source tool can be used for search based testing named AUSTIN. Test data is generated by AUSTIN to achieve branch coverage for C functions. AUSTIN is at least as effective as and more efficient than the ETF in generating branch adequate test data. [2]. Most of the faults occur around the boundary area. With the use of fault detection probability and error detection speed, these errors have been found in the minimum time [3]. Label coverage has been defined a new testing criterion which appears to be both expressive and amenable to efficient automation [4]. The combination of two or more nature inspired algorithms gave a better result than an individual technique, combination of genetic and tabu search algorithms to obtain branch coverage criterion, the neural network with the genetic algorithm. [5, 7]. Complex test cases can be generated from the simple (unit) test cases [6]. Genetic Algorithm is also used for GUI Testing. The coverage analysis of GUI testing has been done on some simple applications as Notepad, WordPad and MS WORD [8]. In some of the applications, the Hybrid Genetic Algorithm (HGA) has been used for the automatic test case generation. [10]. Finding the most critical paths for improving the software testing efficiency GA have its important role [11]. Path coverage in the software testing is always an NP problem with the randomly generated test cases, with the genetic algorithm most paths have been found, the randomly generated test cases divided into different groups for the maximum path coverage. K-means algorithm is used to form different clusters and applied GA to generate the new test cases for maximum path coverage [12, 14].

3. SOFTWARE TESTING
Software testing is a technique to verify the correctness of the software. In the industries software is developed and before delivery to the customer checked for verification. If the software is correct and tested properly before delivery to the client, then it will be very useful. Software testing has been done at each level of the software development life cycle. If the software is not checked in starting then it is a very tuff task to find the faults in the software in the late stage.

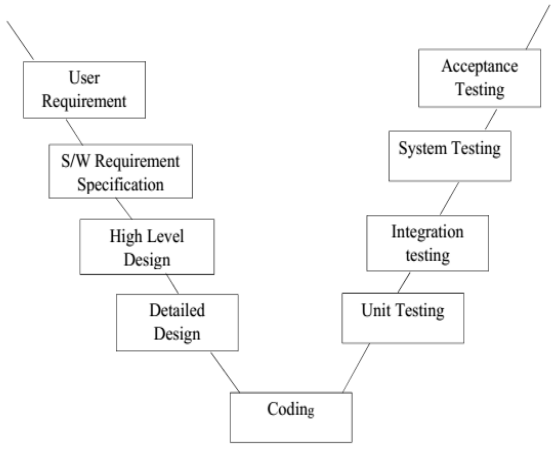

Fig 1: V-Model for Software Testing Process

The V-Model of the testing is shown in figure 1, at each level all types of testing have been described. Cross-pond to each level some different types of testing were defined. The user

requirement phase has acceptance testing, software requirement phase has system testing, the high level design has integration testing, and detail design phase has unit testing. Unit testing is a testing process where we have to check for boundary value analysis and path testing. Integration testing is the combination of the different functions in software/program. System testing is performance testing, load testing, smoke testing and stress testing.

3.1 Path Testing

Path testing is one type of the unit testing. In the path testing, first, a Control Flow Graph has been designed for the software/program. With the use of the cyclomatic complexity of the software/program, all the paths have been found and then applied all def-use. In def-use variables are defined and used. There are two types of use c-use, and p-use, in c-use variables, are used in the calculation, and other one p-use are used in the prediction of the variables

d- define, created, initialized
k- killed, terminated, undefined
u-used
c- used in the computation
p-used in a predicate

e.g.

Program to find the prime number within a given series starting from x and end at y.

```
Prime (int x, int y)
{
    int i,j;
    for (i=x;i<=y;i++)
   {
       for(j=2;j<=i/2;j++)
       {
           if(i%j==0)
              break;
        }
        if(j>i/2)
           printf(" The prime no =%d",i);
    }
}
```

Statements and nodes of CFG

Table 1: Nodes corresponding statements

Statements	Nodes
1,2,3	1
4,5	2
6,7	3
8	4
9	5
10	6
11	7
12	8
13	9
14	10

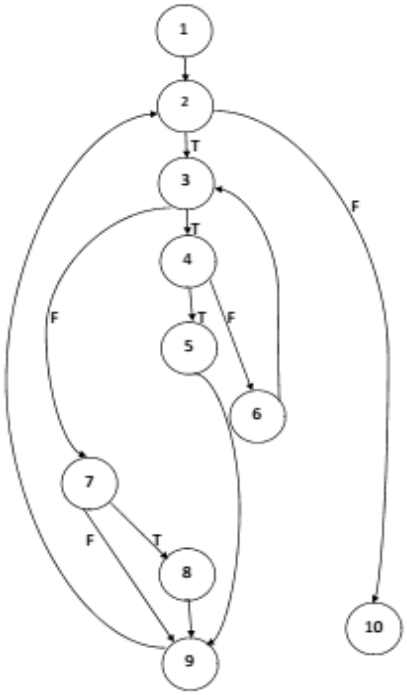

Fig 2: CFG of given example to find the prime number between two numbers.

The paths in the CFG are calculated with the help of the cyclomatic complexity.

Cyclomatic Complexity= E-N+2= 13-10+2=5

So the number of the paths are 5. These are given below.

1-2-3-4-6-3-7-8-9-2-10
1-2-3-4-5-9-2-10
1-2-3-7-8-9-2-10
1-2-3-7-9-2-10
1-2-10

All def-use in this flow graph is defined as

Table 2: Variables and their c and p-uses.

Variables	Defined node	c-use (node)	p-use (node)
x	1	2	
y	1		2
i	1, 2	2	3
j	1, 3	3	

4. GENETIC ALGORITHM

John Holland developed Genetic Algorithm and its basic concepts in 1975. Genetic Algorithm is applied to different types of the problems mainly search based and optimization problems. GAs draw inspiration from the natural search and selection processes leading to the survival of the fittest individuals. The genetic algorithm in software testing plays an important role. The basic genetic algorithm is

```
Simple Genetic Algorithm ( )
{
initialize population;
evaluate population;
while termination criterion not reached
{
select solutions for next population;
```

perform crossover and mutation;
evaluate population;
}
}

Genetic Algorithms operations are the initialization of population, selection, crossover and mutation.

4.1 Genetic Algorithms Operations

Genetic Algorithm has following operations

i Representation: The initial population represented in the different format. In this, the initial population has been represented in 8 bits binary form. For example, if a number is taken as 14, then its equivalent binary should be 00001110.

ii Selection: Selection is a process of selecting the population for checking the fitness function. If not fit then applied crossover or mutation operations. In this paper ranked based selection process has been used.

iii Crossover: Crossover is an operation in which some bits on one individual exchanged with some bits of another individual to generate the new population.

iv Mutation: Mutation operation is that in which one bit is flipped. e.g. 1 in the revert to 0 or 0 to 1.

5. PROPOSED METHOD

In this section, we proposed a method which will be explained in next section. Our proposed method has been divided into three phases. In the first step, a CFG will generate with the help of our designed tool. For the given an example, in the second step, all the paths have been found in the CFG with the support of the cyclomatic complexity and last the most critical stage where random test data generated and applied on the CFG to cover the maximum path, these test cases refined and optimized by the genetic algorithm.

5.1 Proposed Algorithm

The proposed algorithm has been described here.

i Take a program written in C language.
ii Generate the CFG of the C program with the tool.
iii Find all the possible paths in the CFG.
iv Generate the random set of test cases and apply it on the CFG.
v Define the fitness function to calculate the path coverage of the CFG.
vi Check for the path coverage.
vii If path coverage is satisfied, then stop. Otherwise, go to next step.
viii Select the initial population with the rank based selection process.
ix Apply the GA's operations (Crossover and mutation) to generate the new population.
x Goto step 6.

The fitness function for this proposed method is path coverage percentage. i.e.

(Number of the paths covered on applying the input sets/total number of the paths)*100.

5.2 Flow Chart of the Proposed Algorithm

The flow chart for the proposed method is shown in figure 3.

6. EXPERIMENTAL SETUP

Here a C program is written to find the roots of a quadratic equation. This given program is written in turbo C and the CFG of the program is generated with our developed tool.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main ()
{
int a,b,c;
printf("solution of a quadratic equation\n\n\n"); printf("plz i/p cofficient of x square i.e. a\n"); scanf("%d",&a);
printf("plz i/p cofficient of x i.e. b\n");
scanf("%d",&b);
printf("plz i/p constant term i.e. c\n");
scanf("%d",&c);
if(a==0)
{
    printf("eq. is linear and soln=%f",-1*(c/b));
}
else
{
 if(b*b==4*a*c)
{
printf("eq. has only one root and soln=%f",-1*(b/2*a));
}
else
{
 if(b*b>4*a*c)
{
printf("roots of the eq is real and solution are:: "); printf("first root=%f",((-1*b)+(sqrt(b*b-4*a*c))) /2*a);
printf("second root=%f",((-1*b)-(sqrt(b*b-4*a*c)))/2* a);
}
else
{
printf("roots of the eq is imaginary and solution are:: ");
printf("real part of first root=%f",((-1*b)/(2*a)));
printf("imaginary part of first root=%f",((sqrt (4* a*c - b*b)))/2*a);
printf("real part of second root=%f",((-1*b)/(2*a)));
printf("imaginary part of second root=%f",(-1*(sqrt(4*a*c-b*b)))/2*a);
}
}
}
getch();
}
```

For this given program CFG is given below. Till date we don't have so good window based tool to generate the CFG of C program. Our developed tool is window based tool which generate CFG of a given program automatically as shown in figure 4.

6.1 Explanation of Proposed Method with an Example

In this CFG there are 9 nodes and 4 paths.

First Path: 1-2-3-9
Second Path: 1-2-4-5-9
Third Path: 1-2-4-6-7-9
Forth Path: 1-2-3-4-6-8-9

The range of Inputs 0 to 15 for the value of a, b and c.

We have taken this range for inputs so that they easily convert in binary numbers. All inputs are converted 8 bits binary numbers.

Initially, random test cases have been generated and applied to cover the path of given program. According to mutation probability pm>.8 and crossover probability ps<.8, Genetic Algorithms operations are applied. If path coverage is 100%, then stop otherwise applied two point crossover and mutation operation.

Table 3: Table for path coverage and GA operation

S. No	Set of Inputs	Random Test Cases	Paths	Path Coverage	Operation
1	5	[10,12,11] [1,14,13] [0,1,4] [1,5,6] [6,12,11]	4, 3, 1, 3, 4	75%	Crossover
2	9	[5,13,2][11,8,6] [1,0,12] [3,1,4] [5,2,8] [1,8,6] [11,0,7] [3,11,4] [5,12,8]	3, 4, 4, 4, 4, 3, 4, 3, 4	50%	Crossover
3	10	[10,7,6][10,6,7][1,6,2] [3,6,3][5,7,10][0,7,6] [10,6,1][1,6,12] [15,10,7] [5,11,1]	4, 4, 3, 2, 4, 3, 1, 4, 4, 3	100%	Satisfied
4	7	[4,2,1] [3,4,5] [4,2,7] [0,3,1] [5,8,7] [2,6,4] [13,7,9]	4, 4, 4, 1, 4, 3 ,4	75%	Crossover

We have four paths and 4 set of inputs which have different data and 3 different operations mutation, crossover and satisfied. How many paths are covered in each set and which operation is applied is given in figure 5.

As crossover operation is applied to each input set for that first, we have to convert the set of all input data into binary. When data converted in binary we applied crossover operation on it and again generate the new test cases. Now with this new data, we again check for path coverage. In our example, this iteration process has gone 40 times with crossover and mutation operation and after 40 iterations of the algorithm, we achieved 100% path coverage. The results of final iteration is shown in table number 4.

Table 4: Final table for path coverage

S. No	Set of Inputs	Random Test Cases	Paths	Path Coverage	Operation
1	5	[9,10,11] [3,12,9] [2,4,2] [1,8,3] [0,9,11]	4, 3, 2, 3, 1	100%	Satisfied
2	9	[4,14,3][15,7,4] [0,8,10] [3,6,3] [4,2,10] [2,9,5] [10,2,6] [4,14,7] [15,8,8]	3, 4, 1, 2, 4, 3, 4, 3, 4	100%	Satisfied
3	10	[11,5,9][8,4,12][1,8,3] [3,6,3][0,7,10][1,7,6] [14,7,2][2,4,11] [11,8,7] [5,9,2]	4, 4, 3, 2, 1, 3, 4, 4, 4, 3	100%	Satisfied
4	7	[0,2,1] [7,5,4] [9,3,7] [3,6,3] [9,6,7] [1,8,4] [14,6,8]	1, 4, 4, 2, 4, 3 ,4	100%	Satisfied

After 40 iteration of the GA operation 100% path covered shown in figure 6.

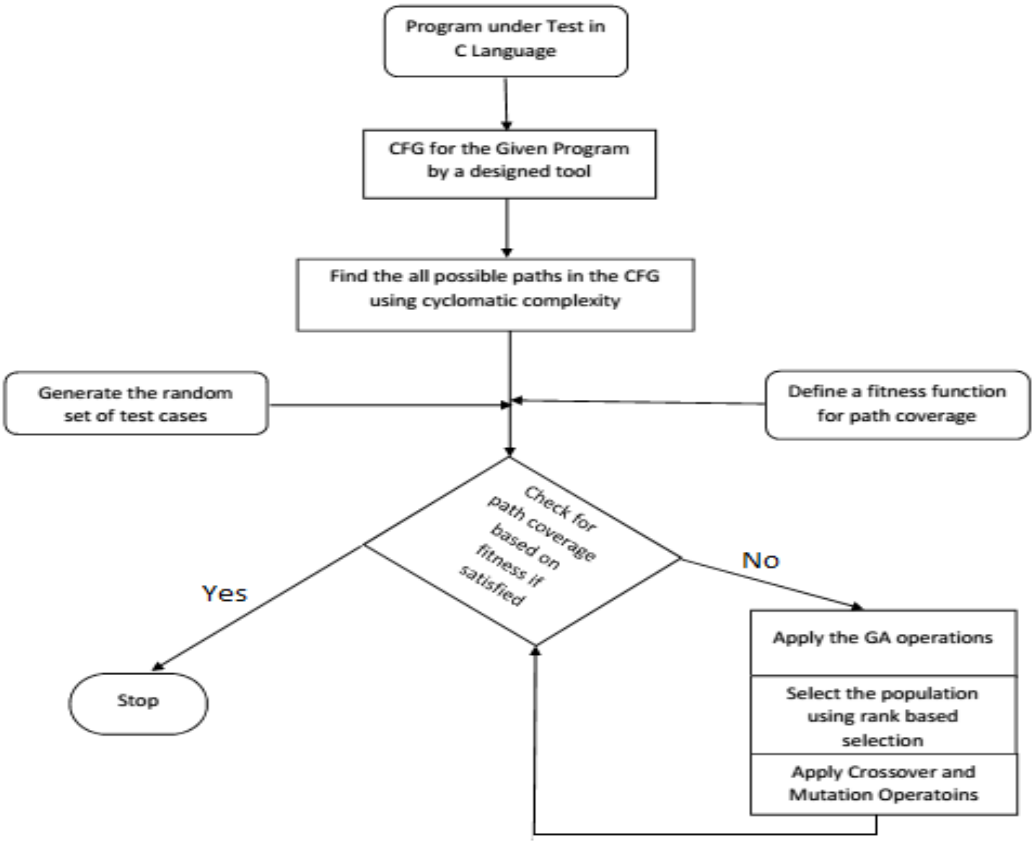

Fig 3: Flow chart for proposed method.

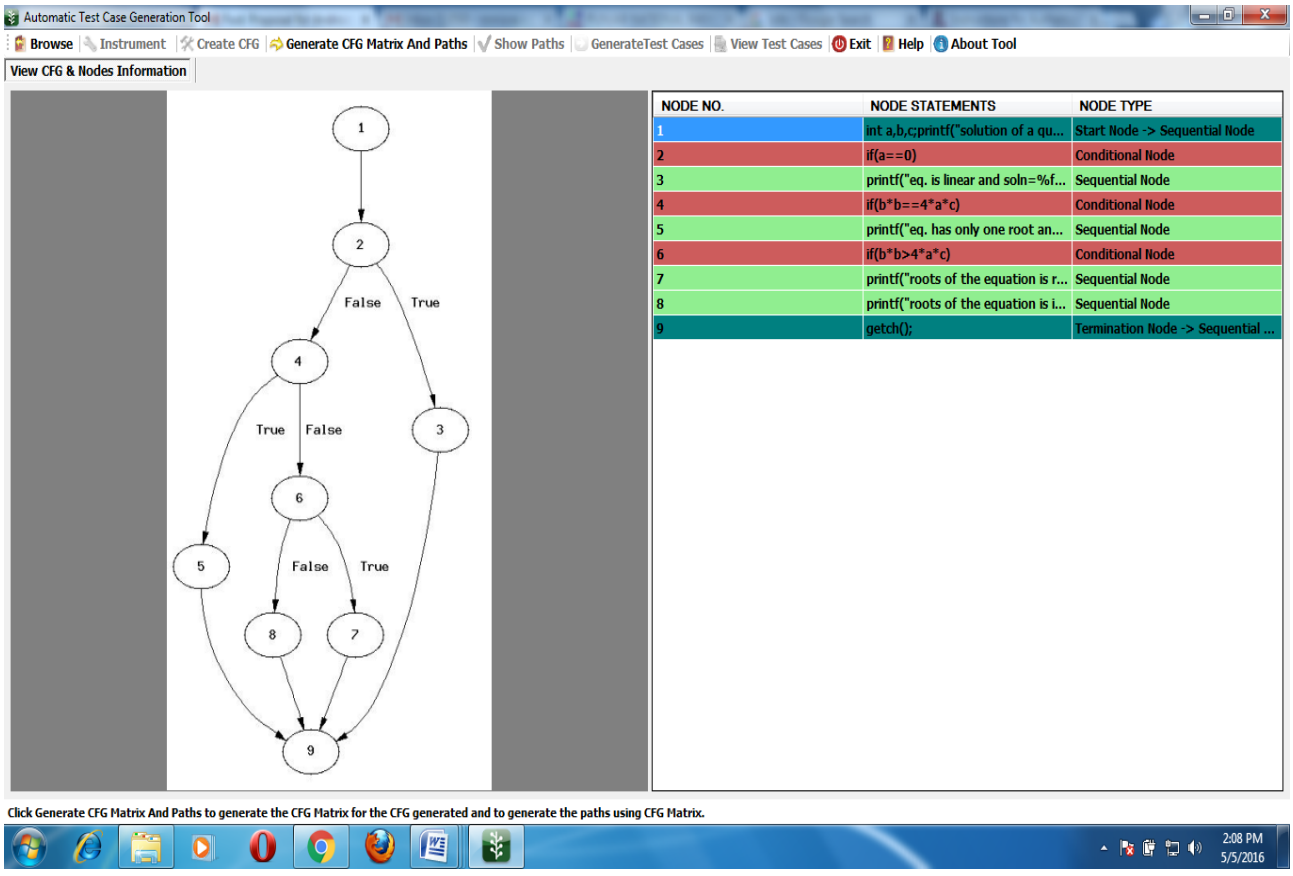

Fig 4: CFG for finding the roots of quadratic program

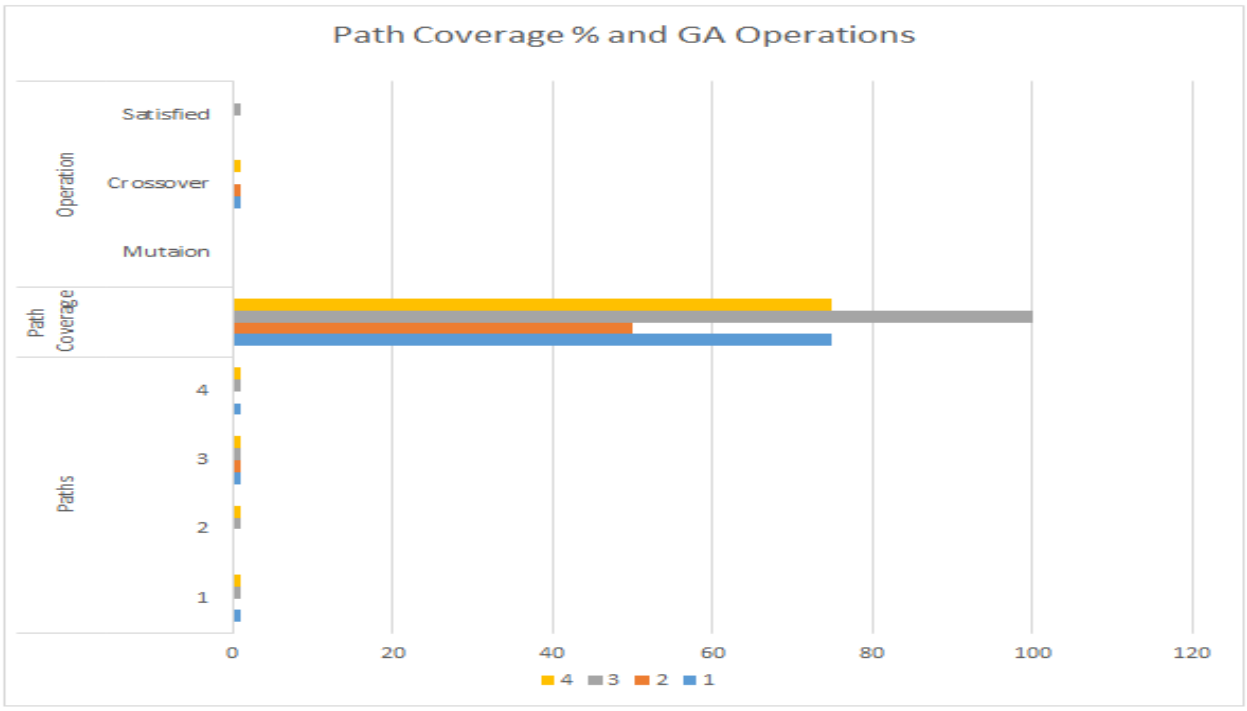

Fig 5: Path Coverage and GA Operations

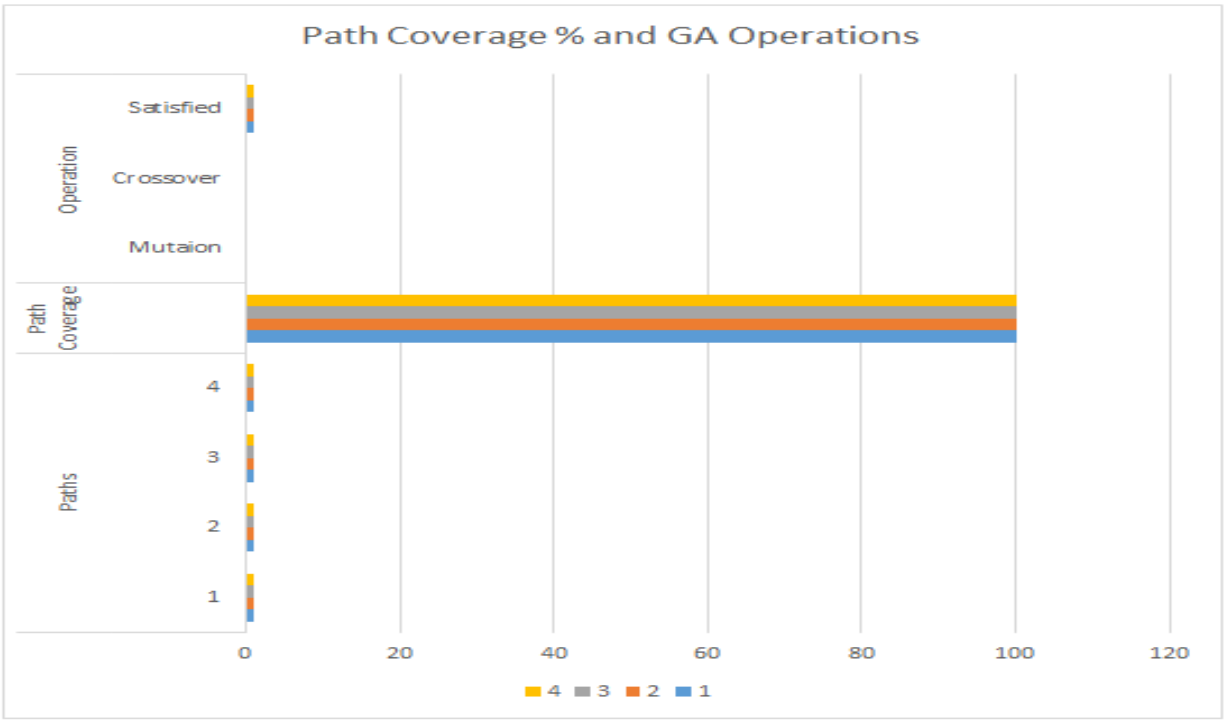

Fig 6: Path Coverage % and GA Operations after 40 iterations

7. CONCLUSION

Software testing is a process of delivering an error/fault free software to the customer. In this paper a method has been proposed, random test cases have been generated and found all du paths. With the random test case generation only some paths of the program/software covered and only few test cases are useful. So genetic algorithm has been used here for refining the test cases among the randomly generated test cases and applied the genetic algorithm approach to generate the new test cases and optimized these test cases. Proposed method and experimental setup showed that it achieved the maximum coverage with the optimized test cases.

8. REFERENCES

[1] Pachauri, Ankur, and Gursaran Srivastava. "Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism." Journal of Systems and Software 86.5 (2013): 1191-1208.

[2] Lakhotia, Kiran, Mark Harman, and Hamilton Gross. "AUSTIN: An open source tool for search based software testing of C programs." Information and Software Technology 55.1 (2013): 112-125.

[3] Moadab, Shahram, and Hassan Rashidi. "Automatic path-oriented test data generation by boundary hypercuboids." Journal of King Saud University-Computer and Information Sciences 28.1 (2016): 82-97.

[4] Bardin, Sébastien, Nikolai Kosmatov, and François Cheynier. "Efficient Leverage of Symbolic ATG Tools to Advanced Coverage Criteria." arXiv preprint arXiv:1308.4045 (2013).

[5] Mayan, J. Albert, and T. Ravi. "Test Case Optimization Using Hybrid Search Technique." Proceedings of the 2014.

[6] International Conference on Interdisciplinary Advances in Applied Computing. ACM, 2014.

[7] Pezze, Mauro, Konstantin Rubinov, and Jochen Wuttke. "Generating effective integration test cases from unit ones." Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on. IEEE, 2013.

[8] Sharma, Chayanika, Sangeeta Sabharwal, and Ritu Sibal. "A survey on software testing techniques using genetic algorithm." arXiv preprint arXiv:1411.1154 (2014).

[9] Rauf, Abdul, Arfan Jaffar, and Arshad Ali Shahid. "Fully automated guide testing and coverage analysis using genetic algorithms." International Journal of Innovative Computing, Information and Control (IJICIC) Vol 7 (2011).

[10] Sivanandam, S. N., and S. N. Deepa. Introduction to genetic algorithms. Springer Science & Business Media, 2007.

[11] Mala, D. Jeya, and V. Mohan. "Quality improvement and optimization of test cases: a hybrid genetic algorithm based approach." ACM SIGSOFT Software Engineering Notes 35.3 (2010): 1-14.

[12] Rao, K. Koteswara, G. S. V. P. Raju, and Srinivasan Nagaraj. "Optimizing the software testing efficiency by using a genetic algorithm: a design methodology." ACM SIGSOFT Software Engineering Notes 38.3 (2013): 1-5.

[13] 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.

[14] Burjorjee, Keki M. "Explaining optimization in genetic algorithms with uniform crossover." Proceedings of the twelfth workshop on Foundations of genetic algorithms XII. ACM, 2013.

[15] Khan Rijwan, and Mohd Amjad. "Automatic Generation of Test Cases for Data Flow Test Paths Using K-Means Clustering and Generic Algorithm. "International Journal of Applied Engineering Research 11.1 (2016): 473-478.

[16] Khan Rijwan and Mohd Amjad, " Automatic test case generation for unit software testing using genetic algorithm and mutation analysis", 2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)