Automatic Software Testing Framework for All def-use with Genetic Algorithm

Rijwan Khan, Akhilesh Kumar Srivastava

Abstract*: The primary objective of automatic testing is to reduce the repetitive manual work and avoid redundancy such that end user gets error free Software. Testing in most of projects/Software has been manual, requiring high number of resources for a significantly large period of time resulting in high project cost and other glitches like efforts, cumbersome tests, and poor result maintenance. Getting most of it out with automation is a process of evaluating the test goals and matching the right tool for the Software/program e.g. choosing right tool and designing appropriate test cases. Several other researchers used numerous techniques to create test cases automatically. Many Algorithms which are nature inspired e.g. Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Genetic Algorithms (GA) etc. are used in research for automation of test case generation. In the current article, authors have used a method based on Genetic Algorithm (GA) to generate the test cases automatically. The key purpose of generating the test cases with the help of the genetic algorithm is to reduce time of input data.

***Index Terms*: Software Testing, Automatic Test Cases, White Box Testing, Genetic Algorithm, Test Case Generation.**

I. INTRODUCTION

Process of Software testing helps in recognizing the quality of Software developed and give the view on correctness and completeness of the Software. Software testing starts from the development phase of the Software. Presently many approaches of Software testing are applied during the Process of Software development. The quality of the application can vary widely from system to system. Some of the common attributes of the Software have been tested before delivery. These quality attributes are portability, reliability, maintainability, usability, and stability. Software are developed for different sectors like education, farming, banking, health etc. for a particular purpose hence validation and verification of the Software is essential. Two policies can be applied for Software testing namely dynamic and static approach. [3, 4]. Usually, a large amount of time is spent on testing process. In real life projects and automation of services, about 40% of the time spent in the testing of the applications. In this paper, an automatic test case generation application introduces which help in generating the test cases automatically such that testing time and cost can be reduced. Optimization approach Genetic algorithm is used for generation of the test cases automatically.

Revised Manuscript Received on June 10, 2019.

Dr. Rijwan Khan, Department of CSE, ABESIT, Ghaziabad, UP, India.
Mr. Akhliesh Kumar Srivastava, Department of CSE, ABESEC, Ghaziabad, UP, India

II. SOFTWARE TESTING

Testing process of Software Development in industry is done manually most of the time. Authors proposed a method for automatic testing which reduced time and cost of testing for project/Software/program [7, 8, 9, 10]. V model of testing process is a very popular. It works as described below.
The V model of testing in Figure 1 mainly focuses on 4 types of the testing namely Unit, Integration, System and User Acceptance Testing (UAT). In this paper only unit testing has been described with genetic algorithm. Part of Unit Testing 1.Path Testing & 2.Mutation Testing are discussed in coming sections.

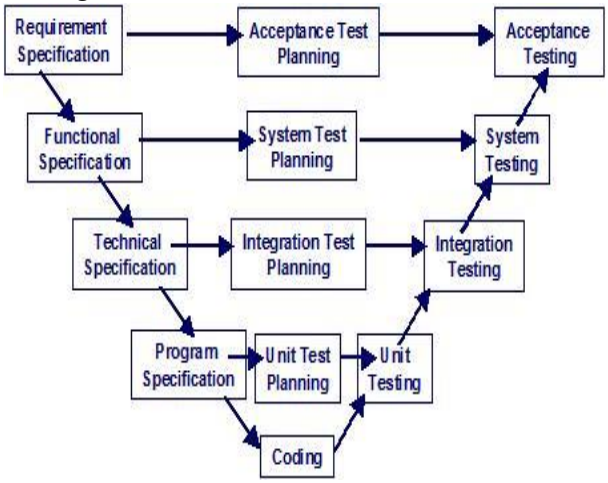

Fig1. V model of Testing

Software testing process is not used only for the removal of the bugs and scrapping the errors, it also assures the delivery of error free and fault free Software to the customers. Software testing process may be dynamic or static in nature. As a part of Static Testing process, white box testing is performed whereas dynamic testing deals with the black box testing. In the process of dynamic Testing input is given and focus is only on the Outputs. The process of Software Testing may also be defined as the process of verification and validation of Software to make sure that the Software meets up the expectation in terms of technical as well as business requirements. The broader categorization of Software testing techniques are dynamic testing and static testing. While doing the Static Testing, Documents e.g. specification and Design Documents and source code of Software under test (SUT) are utilized. Here the source code gets inspected with each of the statement but the execution of Software is not performed.

Hence Static Testing process comprises of inspection, desk checking and code review etc. The process of Dynamic Testing involves the execution of Software under test (SUT) on Test Data input and the output is witnessed. Significance and quality of complete testing gets affected by Test cases set used during the Testing. Hence automatic test case generation can be used for reduction of the cost of Software development process to an extent. Optimal Test cases are required for increasing the efficiency of Software Testing [3, 4].

A. Path Testing

The purpose of going through path testing is to collect the inputs for the program (Test Cases) and to traverse all logical paths in the program/Software. There are two steps in Path Testing Process: Target Path Generation and Test Data Generation [1, 2, 5]. Path of Logical Execution in the program that should be looked into during the testing process is target path generation. In target path generation the source code is needed to construct a Control Flow Graph (CFG). This CFG is generated automatically by a tool [12]. A logical path is flow of input data from entry point to an end point of the CFG. The complete number of paths is identified with the help of the Cyclomatic Complexity [1].

In path testing, main target is to cover all the Program's logical paths. For this purpose input data (Test Cases) should be taken carefully [11]. This input data (Test Cases) is mostly generated randomly which usually is not much effective to cover all the logical paths [1].

B. Mutation Testing

Mutation Testing can be categorized as structural testing wherein some of the mutants are introduced in the program. The purpose of the same is that program will be modified slightly. Each mutated version injected in the program is called mutant, these mutants are searched and killed during the testing process. Different types of mutation testing are used for testing of programs, some of these are value mutations, decision mutations and statement mutations etc. Consider the writing of a Program given below. In this program operators are changed in the program to induce errors. e.g. This Function of a program finds the area of a given Triangle.

```
   void AreaOfTriangle(int Tbase, int Theight)
   {
   float TArea;
   TArea=0.5*Tbase*Theight;
}
```

To induce mutants (errors) in the given program function operator '*' is changed with with operator '+'.

```
   void AreaOfTriangle(int Tbase, int Theight)
   {
   float TArea;
   TArea=0.5+Tbase+Theight;
}
```

The data values of base and height in the above program function with mutants checks for number of mutants that can be discovered in the function. In fact analysis of mutant's will examine appropriateness of test cases to scruitinize the program. There are several operators such as one of the relational operator will get replaced with another relational operator, one Arithmetic operator will get changed with another arithmetic operator etc [4].

III. GENETIC ALGORITH FOR PATH TESTING

Genetic algorithm is used to generate automatic test cases frequently. Basically genetic algorithm has the following operations for path testing.

Representation
Initial Population
Evaluation Function
Selection
Recombination [6]

There are different types of the representation of the data for GA operations. In this paper, for automatic test case generation, input data is represented in binary string of fixed length L [13]. Initial population is randomly generated in fixed size binary string & after the generation of initial population the main role of evaluation function comes into picture. The evaluation function is also called fitness function which has been designed for computing the path coverage. Selection function is used to select the parent for recombination. There are different processes by which individuals can be selected, namely Rank based selection, Roulette Wheel Selection etc. Recombination is the process for generating the new generation population. Two methods: crossover and mutation can be applied for the same [2, 3, 4].

According the size of the program or the possible number of the paths of the program user can select different population size. Usually, the more the paths of the program the more we select for the population size.

Test data problem to a genetic algorithm is represented in an abstract form in terms of a chromosome which is directly relates to a chromosome of a living being. The chromosomes are composed of genes, each of which may be assumed as one of a number of possible values or alleles. While in an organism a gene may represents sex or eye's color, the gene in the test data generation representation sense is one of the variables input. The genetic algorithm manipulates the coding of the set of gene values making up the chromosome at binary string level. This is equivalent to operate on the set of values of all input variables. This is one of the basic distinction from the traditional method such as direct optimization method. The GA operates on a population of the sets of input values rather than a single set of input values [14]. In the paper of an introduction to Data Flow Testing, authors have presented the concept of Control Flow Testing. Authors have stated that the Control Flow Diagrams became a key in figuring out the structure of a Software program. By testing the Control flow among numerous components, authors of this paper formulated and selected the test cases. As a definition, Data-flow testing can be termed as control-flow testing which can analyze the life span of the given data variables.

The key goal of their paper was to elaborate the theory of Data-flow Testing and applying the same to a real time scenario [15]. The concept of messy-GA was used in [16] for coverage of transition of Simulink / state Flow models. Authors of this paper have given an introduction a Software Tool that helped them implement their approach and assess the same on 3 parameters embedded system Simulink models. The messy-GA proved to attain mathematically superior coverage when it was compared with random search and with a professional tool for Simulink/ State Flow model testing. Sandra Rapps et. al. Proposed how one select can test data appropriate for a program using data flow information? Authors of this paper proposed a process much similar to optimization of compiler. In the proposed method authors used variable definition to operate these variables. Authors included each location where variables are defined and used [11]. In [17] authors proposed a method of generation of inter-procedural test data of crucial programs. Interactive programs like this contain variables of type integer, float and Boolean.The key model of this paper focuses on maximum path coverage. Appropriate path is found first. Interprocedural control flow graph (ICFG) is used to represent the test program. The authors of this paper recommended the use of Algorithm for optimum path coverage for generation test cases. For the program which is going to be tested it is required to execute the set of testing path as per the requirement of Structural testing. It is a critical task to generate the set of paths. This may impact not only overall costing of testing activity but efficacy as well. Hence for simplifying the process of Testing, automation will play a vital role. Basis path testing is among the strongest criteria for structural testing. As a study it was proved that Basis path testing requires as many basis paths (test paths) as the cyclomatic complexity of the program [18]. This is done so as each path be an independent path. Here every edge in the CFG (control-flow graph) should be covered by all the paths included in the set of paths called basis set. Along with this, whichever path is not covered in the set of basis paths, can be formulated by a linear combination of the paths in this set. Some techniques for Optimization which are Search-based e.g. genetic algorithms, simulated annealing, and genetic programming are being deployed successfully on a variety of Software Engineering projects throughout the Software engineering life-cycle beginning from requirements gathering [19] to project planning and estimation of cost, through testing to automated maintenance service oriented Software Engineering, optimization of compiler and assessment of quality. Among the recent papers, Harman et al. [20] proposed a detailed review and comprehensive classification of literature on search-based Software Engineering. In the paper [20] it was identified the trends in research and relations among the numerous methods applied and the supplications where they have been deployed. It also highlighted the shortcomings in the available articles and directions for future research scope. Application areas listed in research paper [20] inhibits that Optimization methods which are search-based can be utilized in all dimensions of Software Engineering pursuit [20]. A variety of optimization techniques and probe/search methods can be and have been utilized. Among the most popular used techniques are the local search, genetic algorithm, genetic programming, and simulated annealing.

Yan and Zhang [21] proposed a technique for generation of feasible path F of finite sets that persuades the criterion of basis path coverage. In this article, authors have formulated a subset S from the set F which is minimal so as S always follows the Criterion for test coverage. The methods should investigate the feasibility of all the available paths as a very first task. Feasibility Check consumes lot of time. Exploring the discovery of set of linearly independent paths, Zhonglin and Lingxia [22], Qingfeng and Xiao [23] used cyclomatic complexity concept. Several among the basis paths generated are usually are not feasible because of data dependencies exist among the variables those are engaged in the decision node. Baseline method was combined with the dependence relationship in order to avoid detection of infeasible paths in the research article. [22] and [23] were not able to handle the blocks having loops. Among the most popular search oriented technique is the Genetic algorithm used in the activities pertaining to Software Testing. E.g. for obtaining test data Bint et al [24] used a technique for path generation which was utilized the concept of Genetic Algorithm. Limitation for this technique includes: ‘Generated set of paths was unable to cover all the edges available in the Control Flow Graph’. The same happens because of omission of operations involved in the loop. Along with this, generation of basis path is not possible with the given technique. Just to overcome the limitations of the above mentioned technique, authors of this paper have introduced a fresh Test path generation technique utilizing Genetic Algorithm principle. In order to understand why genetic algorithms should be used, from the previous discussion, it is absolutely clear that the basis path generation is a demanding task. Also the utilization of Genetic Algorithm in the search-based techniques in on the rise in the research articles related to automatic test case generations. The same has been utilized with in the deployment throughout the Software Engineering lifecycle. For the mentioned reasons authors of this paper will utilize principles of Genetic algorithms for generation of the basis test paths. Methods based on the exhaustive search lead to stack in the local minima. On the contrary, Genetic Algorithms are the robust method for wide scale optimization problems which are nonlinear in nature [25]. Another benefit over the traditional techniques is that it yields global sampling in place of being local (e.g. iterative least squares). Hence it lessen the likelihood to fall in the local minima. It avoids the reliance on a presumed starting level model. The authors also shared a prudent attribute of the local methods wherein authors incorporate and take advantage of collected information around model space sampling which results in a significantly productive at the same time vigorous optimization method [25].

IV. PROPOSED ALGORITHM

Authors of this paper propose a method in which the entire work is divided into two parts. First section is about discussing the developed tool for generating a CFG [12]. The second part is about generation of test cases automatically.

The Algorithm for generation of Automatic Test cases in given as

1. Generate CFG for a given program using tool [12].
2. Find all target paths using cyclomatic complexity.
3. Randomly generate the input data set (test cases).
4. Apply the test cases on the CFG.
5. Find the path coverage using fitness function.
6. If path coverage is satisfactory then Goto Step10.
7. Select the individuals with the help of GA selection operation (for reproduction).
8. Apply reproduction operations (Crossover and Mutation) for new generation of input data set (test cases)
9. Goto step 4
10. Halt

In this proposed method, authors used Roulette Wheel Selection process for selection of individuals in reproduction. In crossover operation, authors choose two point crossover and in mutation only one bit is flipped.

V. EXPERIMENTAL SETUP

This Section has been divided into two parts. First part is about the introduction of developed tool to generate the CFG of given program and the second is about applying Genetic Algorithm to ensure the generation of improved Test Cases and cover max def-use.

PART 1

A. Generation of CFG (Control Flow Graph) for the Program

Authors have designed a tool in .Net framework to generate the CFG of a given program. There are two outputs of the tool. One the CFG and the other one node's infomration in CFG (a node is either sequential or conditional).

In this paper a program is taken to generate the prime number between a given ranges. For the same program CFG is formed using the tool and Node's information is generated.

```
/* Program in C for displaying all prime numbers in a given range (intervals) input by the user. */
#include <stdio.h>
#include<conio.h>
int main()
{
  int LowerLimit, UpperLimit, i, Valid, j;
  printf("\n Enter a range to find Prime Numbers: ");
  scanf("%d %d", &LowerLimit,&UpperLimit,);
  printf("Prime numbers in the range %d and %d
  are: ", low, high);

  for(i=LowerLimit+1; i<UpperLimit; i++)
  {
     Valid =0;
     for(j=2; j<=i/2; j++)
     {
      if(i%j==0)
         {
      Valid=1;
            break;
         }
     }
     if(Valid==0)
      printf("%d ",i);
 }
 return 0;
}
```

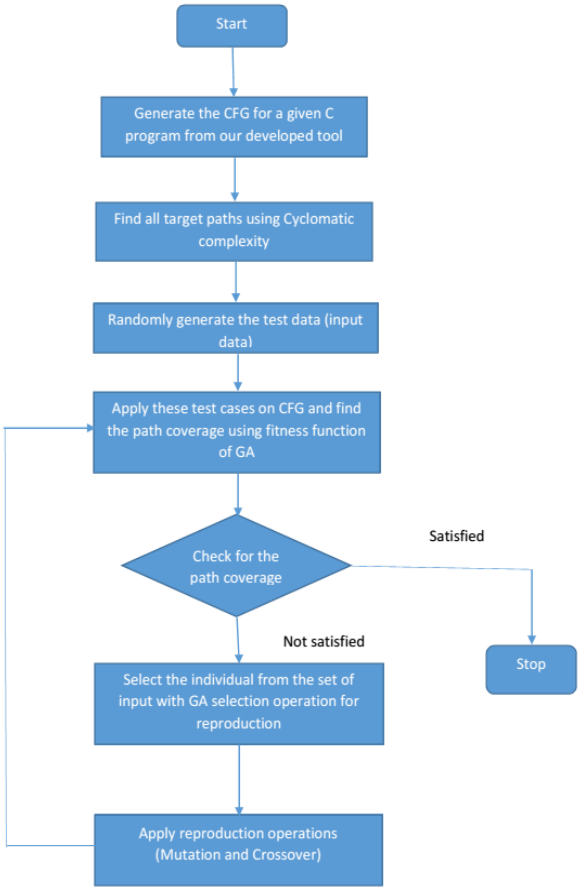

Fig 2. Flow chart of proposed method

When this program was tested in the tool for generating the CFG then the results obtained by the tool is show in in Fig 3.

B. Find all target paths using cyclomatic complexity

Cyclomatic complexity is obtained with the formula C= E-N+2

Here E and N signifies the no of edges and no of Nodes in a Control Flow Graph

For CFG in Fig 3,

Cyclomatic complexity = 13 – 10 + 2 =5.

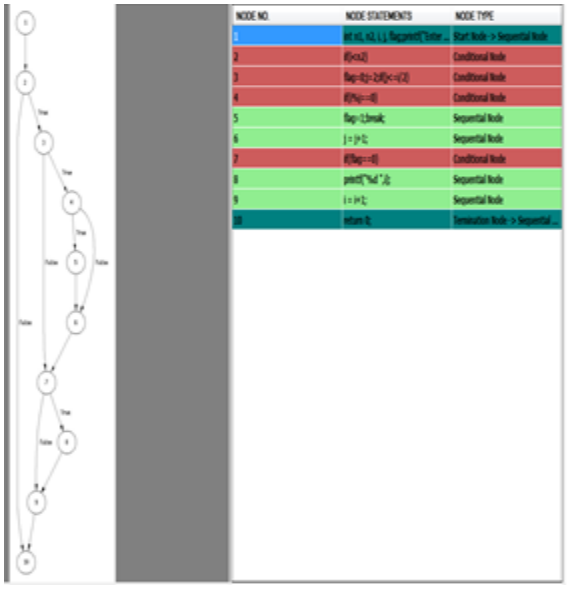

Fig 3. CFG of given program with nodes information

PART 2

C. GA Operation and Fitness Functions

Representation, Mutation, Crossover, Selection and evaluation functions are applied to find the maximum coverage. A C Program was developed by authors to execute all GA operations. This C Program used 12 bit binary representation for input data, 1 bit flip for mutation and two point crossover operation. For the selection operation Roulette Wheel Selection method is used. Evaluation function (fitness function) is equal to number of du-paths covered divided by the total number of du paths.

D. All du-path coverage and results

A program was developed in C language by the authors for def use path coverage. In this program total 25 nodes were found in CFG using the tool. Range of inputs was set as a positive number within a given range. All def-use is givne after applying different iterations.

In first iteration about 60% def use were covered. After 25 iterations, about 80% du paths were covered as shown in fig 5.

```
current population:
111000011011
111000011011
111000101011
111000011011

1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:
(5,8)   (6,8)   (6,17)  (7,17)  (7,20)
1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:

1,2,3,4,5,6,7,8,15,16,17,18,21,22,23,

covered def use pairs:
(6,18)
1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:

covered def use pairs=60.000000%
```

Fig 4. Result of first iteration of program to cover du path

```
current population:
111000011011
110110000111
111000011011
111000011011

1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:
(5,8)   (6,8)   (6,17)  (7,17)  (7,20)
1,2,3,4,5,6,7,8,9,10,11,14,22,23,

covered def use pairs:
(5,10)  (5,11)  (7,10)
1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:

1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:

covered def use pairs=80.000000%
```

Fig 5. All du paths coverage after 25 iterations

```
current population:
110101000111
110110000111
111000011011
110110000111

1,2,3,4,5,6,7,8,9,10,12,13,14,22,23,

covered def use pairs:
(5,8)   (5,10)  (6,8)   (7,10)  (7,13)
1,2,3,4,5,6,7,8,9,10,11,14,22,23,

covered def use pairs:
(5,11)
1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:
(6,17)  (7,17)  (7,20)
1,2,3,4,5,6,7,8,9,10,11,14,22,23,

covered def use pairs:

covered def use pairs=90.000000%
-
```

Fig 6. All du paths coverage after 45 iterations

```
111000011011
110110000111
110110000111
110110000111

1,2,3,4,5,6,7,8,15,16,17,19,20,21,22,23,

covered def use pairs:
(7,20)
1,2,3,4,5,6,7,8,9,10,11,14,22,23,

covered def use pairs:

1,2,3,4,5,6,7,8,9,10,11,14,22,23,

covered def use pairs:

1,2,3,4,5,6,7,8,9,10,11,14,22,23,

covered def use pairs:

covered def use pairs=100.000000%

100
```

Fig 7. All du paths coverage after 60 iterations

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

VI. CONCLUSION

Generation of Testing Data in Software Testing Process plays a vital role in the process of automation of Software Testing Step of software Engineering life cycle. In the current paper authors focused on a self-designed tool for the automatic generation of CFG. It is a useful Window based tool. All the others tools these days are available mostly on LINUX platform. The value of applying GAs to structural testing has been established by covering all branches in a varity of procedure. A method for generating test cases automatically to Unit Testing is proposed, and all def used paths have been found with the help of a C program. Due to its simplicity, GA is used here as in most of the search based optimization problems, GAs are used as a popular mechanism. Hence the proposed method with Genetic Algorithms covered 100% def-used paths in less number of iterations for small programs.

REFERENCES

1. Hermadi, Irman, Chris Lokan, and R. Sarker. "Dynamic stopping criteria for search-based test data generation for path testing." Information and Software Technology 56.4 (2014): 395-407.
2. Girgis, Moheb R., Ahmed S. Ghiduk, and Eman H. Abd-Elkawy. "Automatic Generation of Data Flow Test Paths using a Genetic Algorithm." International Journal of Computer Applications 89.12 (2014): 29-36.
3. Khan, Rijwan, and Mohd Amjad. "Automatic Generation of Test Cases for Data Flow Test Paths Using K-Means Clustering and Genetic Algorithm. “International Journal of Applied Engineering Research 11.1 (2016): 473-478.
4. 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) pages: 1-5.
5. Khan, R., Amjad, M., Performance testing (Load) of web applications based on test case management, Perspectives in Science (2016), http://dx.doi.org/10.1016/j.pisc.2016.04.073
6. Holland, J. H. "Adaptation in Natural and Artificial Systems, the University of Michigan Press, Ann Arbor, MI. 1975." (1975).
7. 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.
8. Haga, Hisashi, and Akihisa Suehiro. "Automatic test case generation based on genetic algorithm and mutation analysis." Control System, Computing and Engineering (ICCSCE), 2012 IEEE International Conference on. IEEE, 2012.
9. Girgis, Moheb R. "Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm." J. UCS 11.6 (2005): 898-915.
10. Ghiduk, Ahmed S., and Moheb R. Girgis. "Using genetic algorithms and dominance concepts for generating reduced test data." Informatica 34.3 (2010).
11. Srivastava, Praveen Ranjan, and Tai-hoon Kim. "Application of genetic algorithm in Software testing." International Journal of Software Engineering and its Applications 3.4 (2009): 87-96.
12. Rijwan Khan, Mohd Amjad, Manish Mishra, “Tool for Generating Control Flow Graph (CFG) Used in Unit Testing”, INDIACom-2017, IEEE International Conference, pp: 1161-1166.
13. S. N. Sivanandam, S. N. Deepa, Introduction to Genetic Algorithm, Springer, book.
14. Wang Xibo, Su Na, Automatic Test Data Generation for Path Testing Using Genetic Algorithms, 2011 Third International Conference on Measuring Technology and Mechatronics Automation, 978-0-7695-4296-6/11 $26.00 © 2011 IEEE.
15. Janvi Bandlaney, Rohit Ghatol, Romit Jadhwani, An Introduction to Data Flow testing, NCSU CSC TR-2006.
16. Jungsup Oh, Mark Harman, Shin Yoo, Transition Testing for Simulink/Stateflow Model Using Messy Genetic Algorithms, ACM, GECCO'11, July 12-16, 2011, Dublin Ireland.
17. Peng Lin, Xiaolu Bao, Zhiyong Shu, Xiaojuan Wang, Jingmin Liu, Test Case Generation Based on Adaptive Genetic Algorithm, 978-1-4673-0788-8/12/ ©2012 IEEE.
18. T.J. McCabe, Structural Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric, NIST Special Publication 500-99, NIST, Washington, D.C., 1982.
19. A. Bagnall, V. Rayward-Smith, I. Whittley, The next release problem, Inf. Softw. Technol. 43 (14) (2001) 883–890.
20. M. Harman, S. Afshin Mansouri, Y. Zhang, Search-based Software Engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (1) (November 2012), article no. 11.
21. J. Yan, J. Zhang, An efficient method to generate feasible paths for basis path testing, Inf. Process. Lett. 107 (3–4) (2008) 87–92.
22. Z. Zhonglin, M. Lingxia, An improved method of acquiring basis path for Software testing, in: Proceedings of 5th International Conference on Computer Science & Education, China, 2010, pp. 1891–1894.
23. D. Qingfeng, D. Xiao, An improved algorithm for basis path testing, in: Proceedings of the International Conference on Business Management and Electronic Information (BMEI), 2011, pp. 175–178.
24. J.R. Bint, Renate Site, Optimizing testing efficiency with error prone path identification and genetic algorithms, in: Proceedings 2004 Australian Software Engineering Conference (ASWEC’04), Australia, 2004, pp. 106–115.
25. K. Gallagher, M. Sambridge, Genetic algorithms: A powerful tool for large-scale nonlinear optimization problems, J. Comput. Geosci. 20 (7–8) (1994) 1229–1236.

AUTHORS PROFILE

First Author Dr. Rijwan Khan is B.Tech (CSE), M.Tech (CSE), and PhD (CSE). Currently he is working as Head of Department CSE in ABESIT, Ghaziabad. He is author of more than 25 journal papers and 12 IEEE conference papers, one book chapter and 3 books. He has already published several research papers on software testing area.

Second Author Mr. Akhilesh Kumar Srivastava is a Computer Science Graduate from K.N.I.T. Sultanpur and Post Graduate from Dr APJ Abdul Kalam Technical University Lucknow. He is registered for PhD at UPES Dehradun. Author is GATE and UGC Net Qualified. He has authored 3 Books and several research Papers in International/ National Journals, presented papers in International Conferences. Author runs his own You Tube Educational Channel with substantial viewership across the globe.