International Journal of Engineering and Science Invention (IJESI)
ISSN (Online): 2319-6734, ISSN (Print): 2319-6726
www.ijesi.org || Volume 10 Issue 11 Series II || November 2021 || PP 67-71

Software Path Testing Optimization Using Genetic Algorithm

Jasmine Satapathy, Suchismita Sahoo, Lopamudra Prusty
Department of Computer Science and Engineering, ABIT, Cuttack
Department of Computer Science and Engineering, ABIT, Cuttack
Department of Computer Science and Engineering, ABIT, Cuttack

Abstract: *Computers have become a necessity in our everyday lives, and are applied in a variety ofsystems, ranging from sophisticated ones to home appliances. Software plays a majorrole in many of these systems. As a result, the usage of software for a variety ofpurposes in separate domains of modern life is rapidly increasing.Software testing is one of the main activities to be carried out in the software development life cycle (SDLC). It consumes more money and time, which leads to automation that reduces the human effort in finding bugs and errors. Automation in the last phase of system development is similar to manual testing. Combinatorial testing is a promising technique for testing high configurable systems.Software systems become larger and more complex every day, and due to the time and cost limitations, it is infeasible to test everything in software with a large configurationspace, or a graphical user interface with many settings and events.The testing generatesan interaction test suit to discover faults cause by parameter interactions using systematic procedure. In this paper. we proposed to generate test suit (set of test cases) for object-oriented software using control flow graph (CFG) diagrams. Test cases are optimized using the evolutionary algorithm, Genetic Algorithm.*

Keywords: *Software Testing, Control Flow Graph, Genetic Algorithm, SDLC.*

I. INTRODUCTION

In software development life cycle (SDLC) process, the software engineering process is to achieve a high quality and high reliable software. One of the important activities in every SDLC is the software testing. The software testing process needs much effort with a human interface. In the Testing process, the Testing is a major aspect of the SDLC. Basically, it is the process of testing the newly progressed software, initial to its actual use.

A. Software testing

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product.

Mainly there are three approaches in software testing namely, Black-box testing, White-box testing and grey box testing.

- Black-box testing is testing the functionality against software specifications. Testers determine what input should be given, what output should be generated, and testers analyze the external behavior of the software.
- White-box testing is an examination of the logic, and the procedure used in the software. It considered on testing the data structures, conditions, branches, loops, conditions, objects, messages, critical paths generated.
- Grey-box testing is a combination of both white and black box testing. In grey box testing tested doesn't have complete information of the system but has some idea about the system. Grey box testing is not completely white box or black box testing Software testing is a verification process which promises the clients expectations and requirements about the software.

B. Path Testing

In the path testing, they are all based on the source code of the program tested, not on specification. Because of code-based testing methods are very manageable to accurate definitions, mathematical analysis and useful measurement. The path testing also known as structure testing is a white box testing method to design various the test cases. For example, using McCabe's Control Graph shown in figure 1.

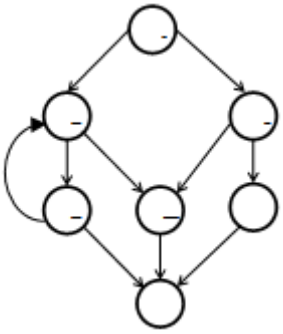

Figure 1: Find path using McCabe's Control Graph in path testing.

If 1 is start node and 7 is end node of the graph in figure 1, then there are various path of the graph. These paths are given below.
Path A: 1237
Path B: 12567
Path C: 14567
Path D: 1467
Path E: 12 (32) x 37
Path F: 12 (32) x 567

PROGRAM GRAPH

A program written in an imperative high level programming language, its program graph is a directed graph or diagraph in which nodes are fragments and edges or lines represent flow of control. Deriving a program from a given program (or source code of programming language) is an easy process. There are four of the basic structured programming shown in figure 2.

If-Then-Else While loop or Pre-test Do-While loop or Post-test Switch-Case

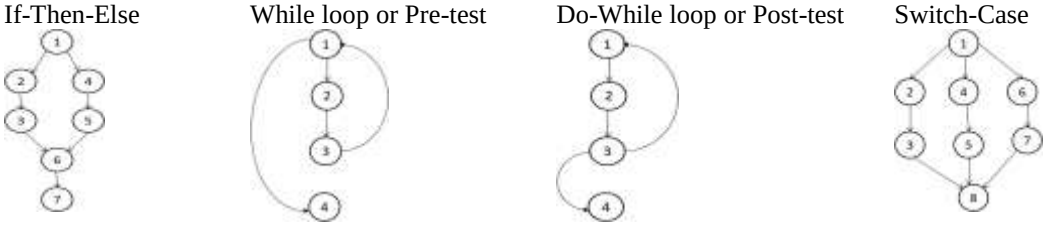

Figure 02: The Program Graphs of Structure Programming Construct.

In the software development techniques, structured Analysis and design methodology has been replaced by object-oriented analysis and design. An important design artifact in structured approach is the Data Flow Diagram (DFD). DFD is very important for the modernization of old legacy systems. It is also very useful in requirement elicitation. However, DFD lacks formalism and by representing DFD formally, ambiguity and inconsistencies can be removed.

II. RELATED WORK

In [1], a tool is developed to automatically detect the infeasible paths that may exist in sourcecode and that are caused by the logically inconsistent predicates related to dead codes, and by thecorrelated conditional statements with respect to a certain variable. The application tool isevaluated against four source codes, and the experimental results showed that the tool caneffectively detect infeasible paths except in the source codes that contain while loop structures.Further, we studied on the main reasons for the infeasible paths, and detecting automatically andstatically two types of infeasible paths; the paths that contain the logically inconsistent predicates(which is a sort of dead codes) and the paths that may be caused by the correlated conditionalstatements. The first type is caused by the existence of contradicting clauses that are related bythe conditional operator. The second type may be caused by the correlation between twopredicates with respect to a certain variable which is def-clear between them.There are plenty of studies on test data generation based on GA and data-flow testing, but there

is a lack of comparable techniques to our method, which tried to reduce the instrumentation costsby identifying entities subset and finding an efficient approach to generate test data for DUPscoverage. In [2], it is presented six metrics that could be used as criteria to determine theadequacy of testing based on data flow analysis of programs. In [3, 4], it is introducedan approach for testing the data-flow interaction between procedures and proposed an algorithmfor computing inter-procedural data dependencies. Wedyan et al. [5] is presented an approach formeasuring data-flow coverage based on state variables defined in base classes or aspects inAspect programs. In [6], it is presented an approach to track intra-procedural definition use associationsat run-time. In [7], it is presented Genetic Algorithm can be used to minimize the testcases. Genetic search can be applied in Software Testing field to minimize the number of testcases by finding most error prone test cases based on the criticality of path. Fitness scaling hasbeen used to overcome the limitations of simple GA which was pre-mature convergence ofindividuals that leads to local optimal value rather than global optima. In [8] it is presented the test case generation by means of UMLsequence diagram using Genetic Algorithm from which best test cases can be optimized.Moreover, this method for test case generation inspires the developers to improve the designquality and to find multiple test cases ready for execution. In [9], we presented a newly improved genetic algorithm for generating softwaretest data. It makes some improvement in traditional genetic algorithm and adopts fitness scalingalgorithm. Results of tests show that the algorithm can avoid precocious phenomena to a certainextent and it has higher rate of convergence.

III. GENETIC ALGORITHM

Genetic Algorithmsare a part of Evolutionary Computing, which is a rapidly growing area of Artificial Intelligence (AI). Also, GAs is a technique to solve problems that needs optimization. It is based on Darwin's Evolution Theory. Here we have used Genetic Algorithm to optimize Software testing paths. The terminology and processing shown in figure 3 and figure 4.

- Population: It is a subset of all the possible (encoded) solutions to the given problem.
- Chromosomes: A chromosome is one such solution to the given problem.
- Gene: A gene is one element position of a chromosome.
- Allele: It is the value a gene takes for a particular chromosome.
- Genotype: Genotype is the population in the computation space.
- Phenotype: Phenotype is the population in the actual real world solution space in which solutions are represented in away they are represented in real world situations.
- Fitness Function: A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output.

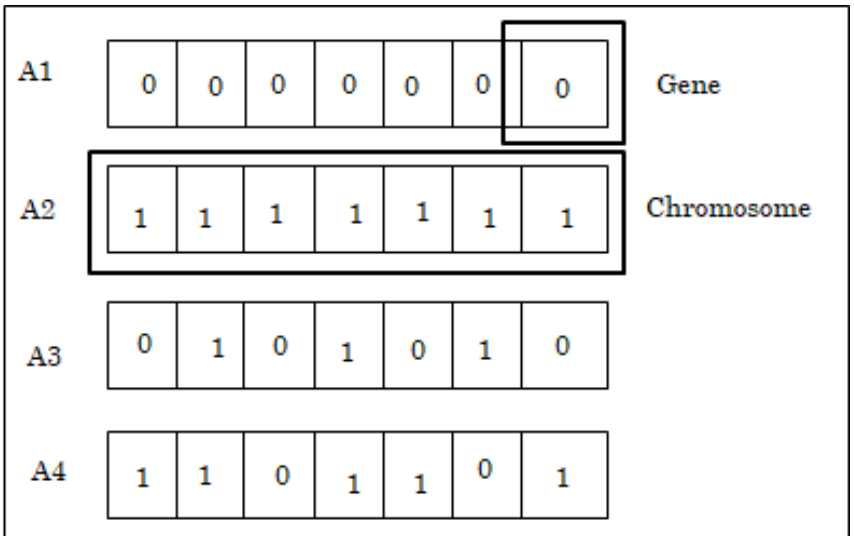

Figure 03: The Terminology of GA.

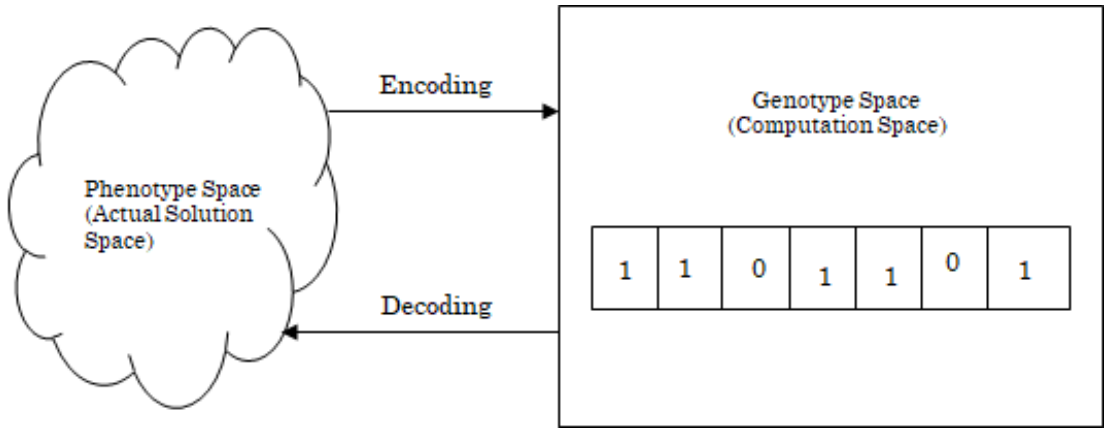

Figure 04: The Processing of GA.

A. Operators in GA:

Selection: This is starting step consists in selecting individuals for reproduction. Randomly,this selection is done with a probability depending on the relative fitness of the individuals so that best ones are chosen for reproduction than poor ones.

Crossover: This operator is applied to combine the genetic information of two parents to generate new offspring.

Mutation: Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.

IV. PROPOSED WORK

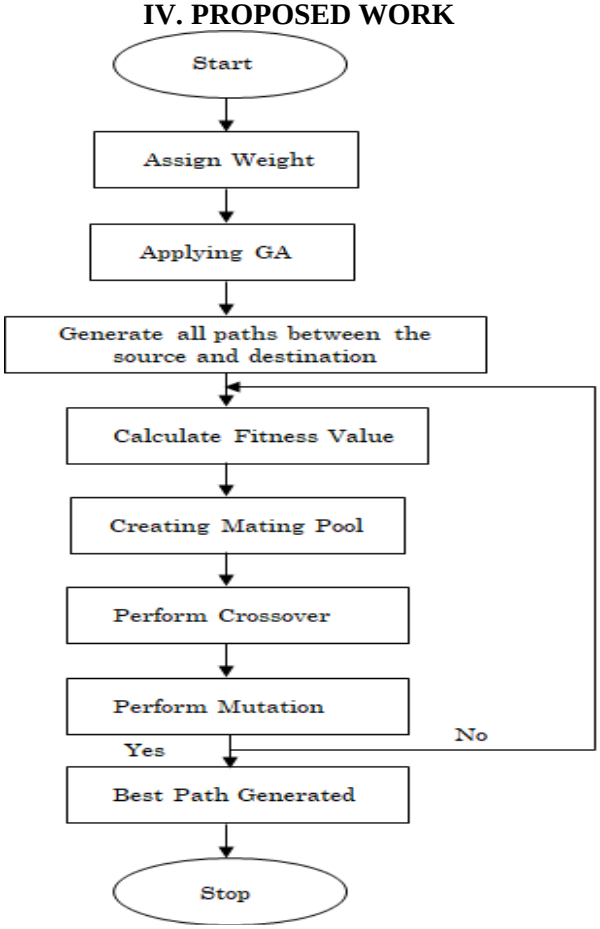

Figure 05: The Flowchart of Proposed work.

We draw the Flowchart shown in figure 05. Assign weights to the nodes as the parent's weight is the weight of the node. If a node has multiple parents, then add the weights of the parents to be the weight of the node. Apply

Genetic Algorithm to the Flowchart.Generate all paths between the source and destination with loops.We calculate fitness value:

- For each path calculate the cost of the path, here the path's cost will be the sum of all the costs assigned to each link in the path
- Apply the fitness functions as F(X)=X*X
- Calculate the probability of the individual as p(i)= F(x)/ΣF(x)

We select the individuals from large initial population produce a new generation of solutions by picking from the existing pool of solutions with a preference for solutions which are better suited than others.The probability range is divided into bins, sized according to the relative fitness of the solution which they represent.By generating random values and seeing the bin where it falls into, to pick the individuals that will form the basis of the next generation. We perform crossover, mate the first two strings together and the second two strings together and so on. For the first two pairs perform single point crossover. For the next two pairs also perform single point crossover. The Mutate if random number generated is less than 0.2 to obtain the best path.Reevaluate fitness of the new generation. And Repeat this process until the fitness value minimizes or all the paths have been covered or maximum number of generations is reached.Hence, the Best Test path generated.

V. CONCLUSION

We present this method for the test case generation in software testing using Genetic Algorithm, for which the optimized path can be found. Using this multiple test cases can be made ready for execution.

REFERENCES

[1] Burhan Barhoush, Izzat Alsmadi,Infeasible Paths Detection Using Static Analysis, TheResearch Bulletin of Jordan ACM, Volume II(III), 120-126, April 2013.

[2] Rapps, S., Weyuker, E.J.: 'Selecting software test data using data flow information', IEEETrans. Softw. Eng., 1985, 11, (4), pp. 367–375

[3] Harrold, M.J., Soffa, M.L.: 'Computation of inter- procedural definition and use dependencies. Proc. Int. Conf. on Computer Languages, New Orleans, Louisiana, March 1990,pp. 297–306

[4] Harrold, M.J., Soffa, M.L.: 'Selecting and using data for integration testing', IEEE Softw.,1991, 8, (2), pp. 58–65

[5] Wedyan, F., Ghosh, S., Vijayasarathy, L.R.: 'An approach and tool for measurement of statevariable based data-flow test coverage for aspect-oriented programs', Inf. Softw. Technol., 2015,59, (C), pp. 233–254

[6] Chaim, M.L., Araujo, R.P.A.: 'An efficient bitwise algorithm for intraprocedural data-flowtesting coverage', Inf. Process. Lett., 2013, 113, (8), pp. 293–300

[7] Bhawna, Gaurav Kumar, Pradeep Kumar Bhatia, Software Test Case Reduction usingGenetic Algorithm: A Modified Approach, IJISET - International Journal of InnovativeScience, Engineering & Technology, Vol. 3 Issue 5, May 2016

[8] V.Mary Sumalatha, G.S.V.P.Raju,Object Oriented Test Case Generation Techniqueusing Genetic Algorithms, International Journal of Computer Applications (0975 – 8887)Volume 61– No.20, January 2013

[9] Wang Xibo, Su Na,Automatic Test Data Generation for Path Testing Using GeneticAlgorithms,2011 Third International Conference on Measuring Technology and MechatronicsAutomation.