# Automatic Generation of Test Suits by Applying Genetic Algorithm

Mohd Athar
Research Scholar
Shri Venkateshwara University,
Gajraula, India

Avdhesh Gupta
IMS Engineering College
Ghaziabad (U.P.) India

## ABSTRACT
The only objective of programming is not to determine the algorithm to accomplish a result, but relevance and correctness of the result also need to be ascertained. Correctness can be insured by applying testing to the result. Testing is most critical practice which is performed for supporting quality assurance. It is substantial but also arduous to warrant the quality of software; half of the cost is consecrated to testing when we converse about software development. Efficient ways can reduce percentage of cost and time incurred in testing. In spite of scads of theoretical work in field of Software Testing, its advancement is slow towards automation. In this approach, Genetic Algorithm (GA), which is a meta-heuristic algorithm, is employed for optimizing path testing to achieve total code coverage.

## General Terms
Genetic Algorithm, Software Under Test, Test Suits

## Keywords
Software Testing, SUT, Code Coverage.

## 1. INTRODUCTION
Software testing is important but it possesses some fundamental challenges. It poses two essentially arduous jobs; selecting tests and assessing test results. Selecting test cases are hard as there is enormous number of potential test inputs in varied sequences but only some of them unwrap failures. In Evaluation/assessment, the real output of test run is compared with expected result. This evaluation is done in opaque-testing. Test Suite(TS) generation from operational profile can be automated but it poses substantial hardheaded problems. Time plays a foremost constraint in case of testing. Another vital component is cost. Due to these two constraints, it is intricate task to execute all test cases. When coverage is taken as optimization parameter then target is formulation of TS that could give 100 % code coverage. Optimization problems can be unriddled by GA which can be regarded as computer model of biological evolution. It works on principle of evolution, where superior chromosomes (having greater fitness value) are chosen for mutation and crossover operations. Evolution continues until the optimized solution is achieved. Good results are found astoundingly speedily when GA is implemented. Generating optimized TS is meta-heuristic problem which can be resolved by GA.

## 2. LITERATURE SURVEY
Testing can be; Opaque box and glass box. Program is viewed as "black box" in opaque box testing approach. Test cases are grounded on system specifications. Glass box study internal structure of program, i.e., it utilizes control structure of the procedural design to obtain test cases. Opaque box and glass box are two broad categories of testing. In opaque box, code structure is not analyzed. Aim of opaque box testing is checking the functionalities of Software Under Test (SUT) only. Glass box testing complement opaque box testing and it is employed for examining codes' structure. Path, statement and branch testing are measures of glass box testing.

In nearly every field, optimization problems rise up and engineering domain is not an elision. As consequence, different types of optimization techniques have been formulated. Yet, these techniques quite frequently have troubles with functions which aren't continuous (differentiable) all over, multi-modal (multiple peaks) & noisy. Hence, robust optimization techniques are required which may be adequate to handle these troubles. John H. Holland coined one of such optimization techniques known as GA which is grounded on biological evolution. GA constitutes a class of adaptive search techniques & routines founded on Darwin's principal which says that weak perish while fitter lingers, famously quoted as "survival of fittest". Artificial chromosomes exchange morphological data among themselves. GAs can be seen as computer model of biological evolution. Good results are found astoundingly speedily when GA is implemented. In software testing, the canonical idea is to seek domain for input variables which meet testing's goal.

Creatures are ideal trouble solver. They have to handle diverse tasks such as adaptation, changing environment. To the frustration of software developers, they do better than the finest computer programs because they develop their capability by the evidently directionless evolution mechanism. The elementary idea of GAs is evolution of succeeding generations of progressively superior combinations of chromosomes. The progression is grounded on parameters which notably regulate the performances of a design. The evolution stratagem is the foundation of GA. The software designer has to face a hitch; it is indispensable to be acquainted with "what to do", beforehand, for each circumstance which may confront a program but to the advantage of evolution; this hindrance isn't faced by it. Evolution is governed by two elementary actions; natural selection and recombination/combining. The natural selection influences which member of population is chosen, lasts & reproduces while recombination ascertains that the genes(or entire chromosome) will amalgamate to form new ones.

A strong non-linear exploration technique is proposed by GA. The GA accomplishes the optimal answer by the stochastic exchange of data between progressively fit samples. It is modeled on natural selection whose incentive is to design and implement stout adaptive system. GA is being used to unbridle array of problems and have emerged as crucial tool in function optimization & machine learning. Natural selection is used to produce adaptation.

The evaluation phase embarks on with configuration of first populace of chromosomes. Chromosomes are differentiated according to their fitness. Fitness in GA is determined through

a non-negative function and maximizing fitness is major target. The intent is the upgrading of the average fitness of population. Healthier chromosomes are given more probability of engagement during selection and reproduction phases & weaker are tossed out. The adequate discriminatory distinction between chromosomes can be made out by only fitness. The GAs is blind, so, does not know anything of the problem except the fitness information.

In the selection stage, chromosomes can be picked for the reproduction in many ways; they may be picked arbitrarily, or inclination may be towards the fitter members.

In the reproduction phase, two chromosomes are chosen from populace. Mutation and Crossover are carried out to fabricate two offspring for following generation.

Crossover operator is practiced on two parents (chromosomes) which are picked by selection operator & swap subsequent substrings of the parents to yield two offspring. The plan is to construct superior chromosomes by fusing genetic material (genes) of fitter parents.

The mutation operator interpolates one or more genes of chosen structure with a low probability. This secures certain diverseness in the chromosomes and precludes doldrums near a local optimum.

Entire procedure is repeated over generation by generation until a global optimum is discovered or some stopping condition is accomplished. One of the minuses can be the overweening iteration numbers and therefore time taken for computation can be high. GA is a robust optimization technique which constitutes a class of adaptive search routines. Working of GA has been explicated through flow diagram in this chapter as well algorithm is also discussed. GA has found application in various problems like game theory, scheduling and state assignment problem.

Now talks about the particulars of all literature reviews accomplished in this paper for resolving problem statement. Now discuss on an imminent analysis of software testing, problems confronted by software testers, different methods used for Test Suits (TS) optimization, importance of GA in software testing, comparison of GA with other alternatives. Problems linked to search and optimization can be figured out by GA. It has been enforced to both Black Box Testing (BBT) and White Box Testing (WBT). Testing tools can be put in two class; dynamic & static.

*Static analyzers:* It probes programs thoroughly and automatically. It is employed on particular language, i.e., it is language dependent.

*Code inspectors:* It scrutinizes program to vouch that it hold on minimum quality criteria. Some COBOL tools abide it, for e.g., AORIS librarian system.

*Output comparators:* It checks weather anticipated and obtained outputs are same or not. For e.g. JUnit is such a tool.

*Coverage analyzers:* It finds degree of coverage. One of its e.g. CodeCover tool.

Coverage analyzers and output comparators are dynamic testing tools while static analyzers and code inspectors are static testing tools. JUnit and CodeCover are extremely noted testing tools.

*JUnit:* It is UT framework for Java. It is applied for testing of single component, IT and system testing.

*CodeCover tool:* It is a well-known Eclipse plug-in, employed as white box coverage tool. This tool is very apposite to assure weather TS is giving full code coverage or not.

It's important to have adequate test cases for accomplishment of testing and making software more dependable. Making system reliable is vital as flunking it could sustain massive losses. BBT is optimized by applying GA. It's implemented in Matlab. Test cases of SUT are heavily influenced by GA. GA depends on various parameters. Population size is vital parameter. Bigger population size brings variation in initial populace at cost of more function evaluations and longer completing times.

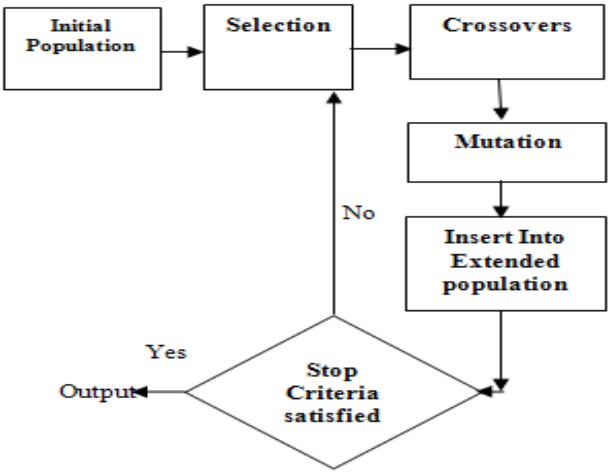



Figure 1: Proposed solution

Figure 1 shows that fitness function is defined, threshold fitness is set, population size is set, crossover rate taken 0.6, mutation rate taken .001.While halt condition is not met, reproduce by crossover and mutation

Testing job is reckoned to be optimization problem whose intent is maximization of noticing errors with minimization of effort. GAs with specification can obtain results with superior quality in lesser time.

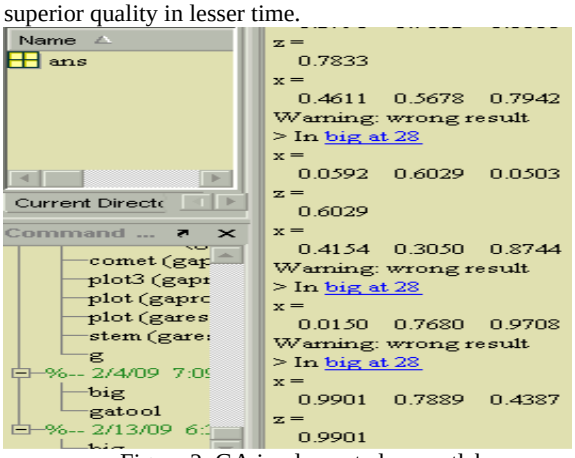



Figure 2: GA implemented on matlab

Figure 2 shows the implementation of GA on test cases of a program. Implementation is done on matlab. Value which differs from specification (expected value) of the software is depicted as error.

Two central issues in process of evolution of genetic search are population diversity and selective pressure. There may be chances of converging early to local optimum solutions if strong pressure is given on selection. On other hand, weaker selection pressure may leads to unproductive search and optimization. Fitness is the key for selection of parents. Fitter chromosomes have more likelihood to get selected. Crossover works on two chromosomes which are chosen as parents to construct two children. In this research work, point crossover is employed; two parents are divided partly, with both children getting half of each of parents. Mutation is applied to amend genes in chromosome. For example, let a

chromosome be: 'abdfag', Mutation may pick gene at position 2 and transform it to a 'z', thus, ensuing a new chromosome: 'azdfag'.
In the work done by Dhawan et.al. test data for input set is delimitated in terms of stipulations which illustrate valid and invalid data values. Stipulations are determined from program's specification. Figure 8 shows fitness values for different input variables.

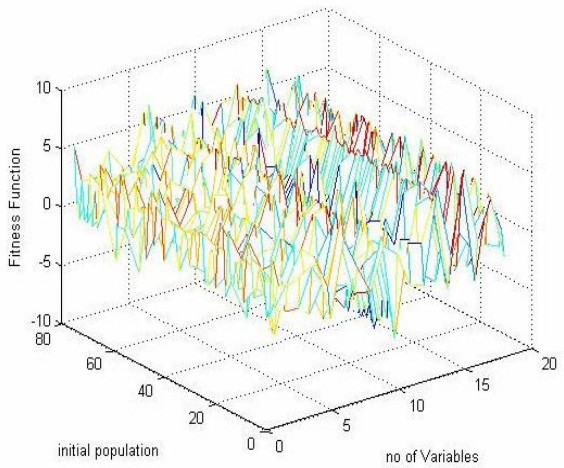



Figure 3 Graph between fitness initial population and input

Figure 3: shows 3-D graph between three variables. X-axis represents "number of variables", y-axis represents "initial population", z-axis represents "fitness function". The graph is depicting that as values of number of variables and initial population are increased, fitness function also increases.
Prerequisite for WBT is availability of source code of SUT. Software testing can be optimized by discovering most critical paths in modules of SUT. Comprehensive testing is very convoluted task. There can be case that part of program which is checked is not error prone also error prone part may be left untested. Also more priority is given to parts of programs which are most critical and they are tested first. Most critical parts of program are branches, loops and predicate nodes. This increases testing efficiency. Srivastava et.al. proposed to take weighted CFG.

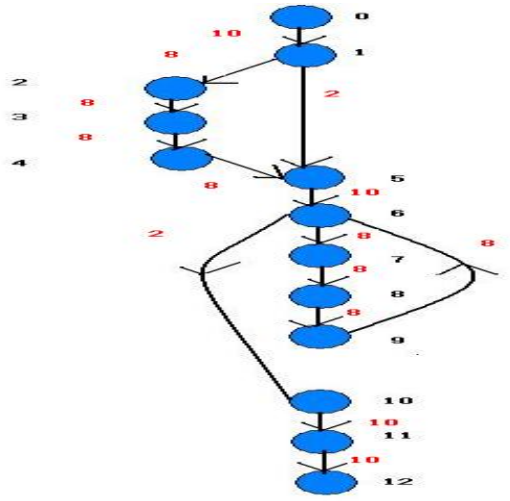
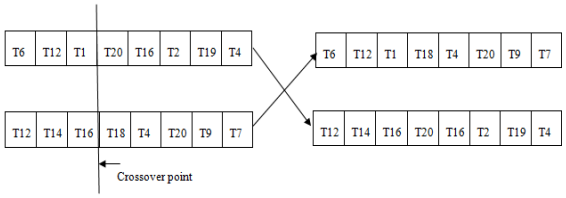


Figure 4: Control Flow Graph with weighted edges

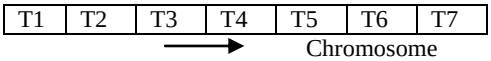
Figure 4 displays weighted CFG. Weighted CFG is CFG whose edges are allotted weight. 80-20 rule is followed to give weights, i.e., 80% of the weight coming from input edge is allotted to loops or branches and 20% is allotted to the sequential statements.
GA is employed to find test data that covers most critical paths in program. Fitness function used is: $F = \Sigma$ wi where wi is edge "i's" weight which is part of path.
Probability of selection of path 'i' is calculated as: $pi=Fi/\Sigma Fi$
Then cumulative probability cj for each path 'j' is calculated by: $cj=\Sigma pi$.
A single point crossover is used. A genetic search technique for random generation of test data has been proposed. GA can be employed to discover most critical paths which can improve testing efficiency. According to Srivastava et.al, GA works better than exhaustive search and local search technique as GA tends to find global optimum solution.
RT is used to validate altered program but it's considered as very costly activity. This lays the significance of TS optimization to have prioritized test cases. Askarunisa et. al. discussed about APFDc which is influenced by APFD for valuating rate of fault detection of optimized TS. Other metrics discussed are APBC, APSC, APCC and APLC. Coverage and cost are driving forces which influence prioritization of test case.

Extent of coverage can be evaluated by metrics APSC, APLC, APBC and APCC.
APSC assesses rate at which prioritized TS covers statement.
APBC assesses rate at which prioritized TS covers branch
APLC assesses rate at which prioritized TS covers loop
APCC assesses rate at which prioritized TS covers condition

According to previous research, for prioritizing test cases, GA is employed and total code coverage is prioritization criteria. It has used APCC metric to show success of GA in optimizing the TS. It has applied GA on path testing to achieve full code coverage. In first step, CFG of module is drawn. Then, different paths in the CFG have been discovered. Test cases are acquainted with paths they cover. Test cases are grouped together to form initial population of chromosomes. So, TS are the chromosomes and test case is a gene and TS needs to be optimized. Fitness function is picking out least count of test cases to address all independents paths. Crossover applied is explained through the figure.



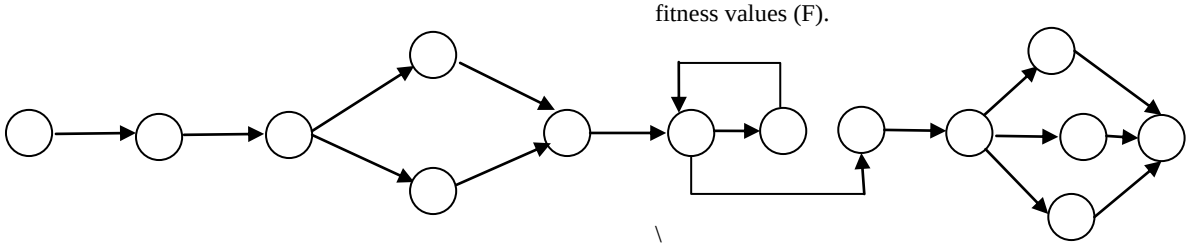

Takagi et. al. discussed about generation of TS from operational profile and optimizing by using GA. Operational profile contain set of all operations that are carried out by SUT and their probability of occurrence. Large amount of test data can be framed using operational profile in order to ascertain consistency of SUT. But because constraints (time & cost), all test cases can't be executed. Usage Distribution Coverage (UDC) has been taken as the criteria to optimize the test suite. UDC shows percentage of software operation executed by TS. Chromosomes are TS and test case represents gene.




Takagi et.al has employed two point crossover. Two TS are

selected randomly and two cut positions are arbitrarily selected on each TS and test cases between cut positions are swapped.

CFG is data structure; graph of program with purpose of representing control flow of program and widely used in software analysis. Directed graph is used to signify CFG. Control flown between code lines are denoted by edges of CFG, i.e., edges represent flow and node in CFG represents code lines without any jumps.



**Sequence** **If** **While** **Case**

Figure 5 The Structured constructs in flow graph form

CFG can be employed to find cyclomatic complexity (CC) of program. CC computes program's complexity. Total linearly independent paths in source code are detected by CFG.

*Formulas for CC:*

E-V+2 where 'E' is edges' count, 'V' is vertices' count

Or No. of regions+1 or No. of predicates+1

The value of CC supplies upper bound for total independent paths and, by inference, upper bound on count of test which ought to be reckoned to assure coverage of all program statements.

GA is used by WBT to search for precise test data which give high coverage of SUT. Fitness function is necessary feature of GA and is engineered on basis of SUT. Objective function is used to construct fitness function which is applied to sequent genetic ops. Intent of GA is to maximize fitness function. If fitness function is modeled well, probability of reaching higher coverage is enhanced considerably. Based on CFG and requisite test aim, test criteria are separated in different classes:

*Node-oriented methods:* It requires traversal of particular nodes in CFG. Statement test and condition test can be categorized in this class. Accomplishment of partial aim of this method isn't reliant on path executed in CFG.

*Path-oriented methods:* It requires traversal of definite path in CFG. This class comprise of every variation of path test. Finding fitness functions for this class of test is less sophisticated compare to node-oriented method.

Other test criteria can be node-path-oriented method and node-node-oriented method.

Baresel et.al separates test into partial objectives and fitness functions are defined for each partial objective, i.e., each statement corresponds partial objective when applying coverage criterion. Ultimate goal of fitness function can be summarized as:

Substantially enhance chance of detecting solution and attain improved coverage of SUT

Reduce count of iterations to achieve optimization.

Intention of Baresel et.al is to better formulation of fitness functions, so that, evolutionary testing could be enhanced by getting prominent coverage. It is difficult to investigate reasons behind unsuccessfulness of optimizations because of large search space and existence of many dimensions.

Execution of GA commence with stochastic population of chromosomes. Fitness function assist evaluation of population and reproductive chances is allowed to population which symbolizes a more adept solution to problem. Chromosomes having superior fitness value are selected by Selection operator. Selection operator is crucial in GA. Jadaan et.al. proposed altered Roulette Wheel Selection(RWS) to reduce incertitude in selection process. RWS probabilistically choose individuals based on their fitness values (F).

Fittest chromosome takes largest share within roulette wheel and chromosome with least fitness value takes smallest share. A random number is generated in interval [0, S] where S is ΣF, chromosome whose segment is closer to random number is picked.

*Ranked based RWS*

Jadaan et. al. modified RWS where individuals are designated fitness values equal to their rank in population; probability of selection of highest ranked chromosome is maximum. Probability is computed as :

P(Ranked based RWS) : 2* Rank/(N_Pop*(N_Pop+1))

RWS is easy to apply. But ranked based RWR is faster than RWR. In RWR, fitness value overlooks other fitness values if good result is disclosed early. This abbreviates diverseness in mating pool and cause GAs to go to awry answers. RRWS overpowers this difficulty and raises diverseness. GA becomes firmer and quicker in getting optimum solutions when RRWS is used instead of RWS.

Reproduction of chromosomes is done by mutation and crossover operator. Travelling Salesman Problem (TSP) and Test suite optimization; both are metaheuristic problems.

One of the first problems encountered by any natural language processing (NLP) task is of ambiguity. The inability to resolve the problem of lexical ambiguity is one explanation for the poor performance of the NLP based systems like information retrieval (IR). The disambiguation process, which helps people identify the correct sense of a word, is not difficult for us. We can perform it easily and accurately. However, this disambiguation process is not so easy for computer applications. Given a text including an ambiguous word like "crane", without performing some sort of disambiguation, it is impossible for a machine to know whether we are talking about the machine that lifts and moves heavy objects or the large long-necked wading bird of marshes and plains in many parts of the world. And, unfortunately, even when after Disambiguating words, machines may not be able to resolve ambiguities. The task of word sense disambiguation is to make machines perform as well as people in identifying the senses of ambiguous words in a context. The ambiguity removal is a difficult task for the computer systems and so it's detection. The query is

ambiguous or not it actually depends upon the results that are generated by the query.

## 3. OBJECTVE

Software testing is a principal technique which is employed for bettering quality attributes of Software Under Test (SUT), particularly reliability and correctness but is also regarded to be tedious. This is also supposed to be intricate work. Software testing suffers from the cognitive biasing of the testers. Automation of testing is a proficient way which can foreshorten time taken and cost incurred in software development. It can also notably better the quality of software.

Intent is to optimize TS which could give 100 % code coverage. This optimization which is grounded on total code coverage needs that inner composition of program is well-known. Inner composition of program can be discovered by Path testing in which a set of test-paths are selected in a program. The different independent paths in the program could be determined through CFG. An independent path is that path in CFG that has one novel set of processing statements or novel conditions. Test cases carrying the information of the path covered by them are grouped together to form initial population of chromosomes and GA is applied. In the end, TS is obtained for each module that gives hundred percent code coverage.

## 4. METHODOLOGY

This section delves into minutia of approach that is complied to reach the motive of optimizing software testing using GA. Generating TS that guarantees full coverage of statements in program, is complex task. There are also odds that more than one test case in TS are checking same path. This redundancy is not appreciated. It is imperative to have optimized test data sets. In this section, GA is employed for optimizing Path testing.

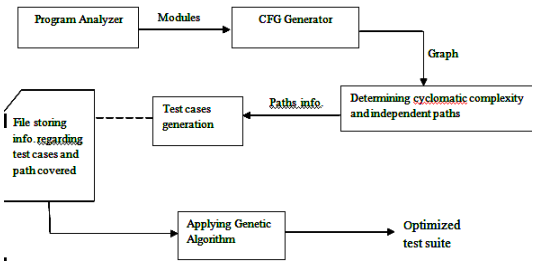



Figure 6: Block diagram of methodology

Figure 6 illustrates approach applied in this work to accomplish the objective. Program analyzer analyzes the java program and discovers all the modules in it. CFG generator generates the CFG for each module. CFG is used to find CC and total independent paths. Test cases are generated and paths followed by them are found. The data regarding test cases and path followed are put in a file. This file is utilized when GA is employed.

WBT is a verification technique, applied by Software Engineers to probe if their codes work as anticipated. Software's structure is tested by codes' execution. In WBT, inner composition of SUT is studied.

In first step, program's analysis is automated by code analyzer; program written in C++ which find all modules in program. The code analyzer parses java programs to be tested. Parsing of program is done to find all modules in it. It is substantive to find all modules as CFG is generated for each module separately. For each module, TS is generated and optimized. Input to the code analyzer is path of a java file.

CFG is graph of program with purpose of representing control flow of program. Directed graph is used to signify CFG. Control flown between statements are denoted by edges of CFG, i.e., edges represent flow and node in CFG represents code lines without any jumps. CFG is used to find CC from which total count of independent paths can be determined. Count of independent paths is used to fix size of chromosomes.

CFG of a module determines its CC. It computes program's complexity. Total linearly independent paths in source code are determined by CC.

K-Shortest Path problem is discovering set of paths(P1,P2,P3…) between given pair of nodes. Here this algorithm is applied to detect all independent paths between start and end nodes. All paths are displayed along with CC. The TS for that module should cover all independent paths. Population of test cases is generated using operational profile. For each test case, corresponding path in CFG is determined.

## 5. CONCLUSION

In this work, optimization of software testing is achieved by employing GA and the process is automated. It will results in formulation of test suite for a module that gives 100 % code coverage, this is our task for next. The process of code analysis to find all modules in a program, generation of CFG, finding cyclomatic complexity, determination of all independent paths and GA steps are automated. GA is employed on a set of different software programs and analyses are done on results obtained which decide performance of GA.

In this future work, test cases will be created manually and paths followed by them will be manually determined. RSW selection operator will be employed for selecting parents and single point crossover will be employed as crossover operator. Also, In future, test case generation from operational profile and path followed by them in CFG can be automated. Other selection operators and crossover operator can be applied and comparison can be drawn between performances of different operators.

In this work very basic fitness function is used. In future, fitness function can be formulated based on APCC and other metrics.

## 6. REFERENCES

[1] Bayliss, D. and Taleb-Bendiab, A.: 'A global optimisation technique for concurrent conceptual design', Proc. Of ACEDC'94, PEDC, University of Plymouth, UK., pp. 179-184, 1994

[2] BCS SIGIST (British Computer Society, Specialist Interest Group in Software Testing): Glossary of terms used in software testing, 1995

[3] DeMillo R. A. and Offutt A. J.: 'Experimental results from an automatic test case generator', ACM Transactions on Software Engineering and Methodology, Vol. 2, No. 2, pp. 109-127, April 1993

[4] Feldman, M. B. and Koffman, E. B.: 'ADA, problem solving and program design', Addison-Wesley Publishing Company, 1993

[5] Frankl P. G. and Weiss S. N.: 'An experimental Comparison of the effectiveness of branch testing and Data Flow Testing', IEEE Transactions on Software Engineering, Vol. 19, No. 8, pp. 774-787, August 1993

[6] Gallagher M. J. and Narasimhan V. L.: 'A software system for the generation of test data for ADA programs', Micro processing and Microprogramming, Vol. 38, pp. 637-644, 1993

[7] Gutjahr W.: 'Automatische Testdatengenerierung zur Unterstuetzung des Software tests', Informatik Forschung und Entwicklung, Vol. 8, Part 3, pp. 128-136, 1993

[8] Hills, W. and Barlow, M. I.: 'The application of simulated annealing within a knowledge-based layout design system', Proc. of ACEDC'94, PEDC, University of Plymouth, UK., pp. 122-127, 1994

[9] Holmes, S. T., Jones, B. F. and Eyres, D. E: 'An improved strategy for automatic generation of test data', Proc. of Software Quality Management '93, pp. 565-77, 1993

[10] Jin L., Zhu H. and Hall P.: 'Testing for quality assurance of hypertext applications', Proceedings of the third Int. Conf. on Software Quality Management SQM 95, Vol. 2, pp. 379-390, April 1995

[11] Korel B.: 'Dynamic method for software test data generation', Software Testing, Verification and Releliability, Vol. 2, pp. 203-213, 1992

[12] Lucasius C. B. and Kateman G.: 'Understanding and using genetic algorithms; Part 1. Concepts, properties and context', Chemometrics and Intelligent Laboratory Systems, Vol. 19, Part 1, pp. 1-33, 1993

[13] Müllerburg, M.: 'Systematic stepwise testing: a method for testing large complex systems', Proceedings of the third Int. Conf. on Software Quality Management SQM 95, Vol. 2, pp. 391-402, April 1995

[14] O'Dare, M. J. and Arslan, T.: ' Generating test patterns for VLSI circuits using a genetic algorithm', Electronics Letters, Vol. 30, No. 10, pp. 778-779, February 1994

[15] Parmee, I. C. and Denham, M. J.: 'The integration of adaptive search techniques with current engineering design practice', Proc. of ACEDC'94, PEDC, University of Plymouth, UK., pp. 1-13, 1994

[16] Parmee I. C., Denham M. J. and Roberts A.: 'evolutionary engineering design using the Genetic Algorithm', International Conference on Design ICED'93 The Hague 17-19, August 1993

[17] Rayward-Smith, V. J. and Debuse, J. C. W.: 'Generalized adaptive search techniques', Proc. of ACEDC'94, PEDC, University of Plymouth, UK., pp. 141-145, 1994

[18] Reeves, C., Steele, N. and Liu, J.: 'Tabu search and genetic algorithms for filter design', Proc. of ACEDC'94, PEDC, University of Plymouth, UK., pp. 117-120, 1994

[19] Roberts, A. and Wade, G.: 'Optimization of finite wordlength Filters using a genetic algorithm', Proc. of ACEDC'94, PEDC, University of Plymouth, UK., pp. 37-43, 1994

[20] Roper, M.: 'Software testing', International software quality assurance Series, 1994

[21] Schultz A. C., Grefenstette J. J. and DeJong K. A.: 'Test and evaluation by Genetic Algorithms', U.S. Naval Res. Lab. Washington D.C. USA, IEEE Expert, Vol. 8, Part 5, pp. 9-14, 1993

[22] Sthamer, H.-H., Jones, B. F. and Eryes, D. E.: 'Generating test data for ADA generic Procedures using Genetic Algorithms', Proc. of ACEDC'94, PEDC, University of Plymouth, UK., pp. 134-140, 1994

[23] Tennant A. and Chambers, B.: 'Adaptive optimization techniques for the design of microwave absorbers', Proc. of ACEDC'94, PEDC, University of Plymouth, UK., pp. 44-49, 1994

[24] Watkins, A. L.: 'The automatic Generation of Test Data using Genetic Algorithms', Conference proceedings Dundee, 1995

[25] Yang, X., Jones, B. F. and Eyres, D.: 'The automatic generation of software test data from Z specifications', Research Project Report III, CS-95-2, February 1995