A Major Project Report On

COMPARISON OF EVOLUTIONARY ALGORITHMS FOR TEST DATA GENERATION

Submitted in the Partial Fulfilment of the Requirement

For the Award of Degree of

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

By

CHIRAG GOLECHHA

(Roll No. 2K12/SWE/12)

Under the guidance of

DR. RUCHIKA MALHOTRA

Department of Software Engineering

Delhi Technological University, Delhi

Department of Computer Engineering

Delhi Technological University, Delhi

2012-2014

DECLARATION

I hereby declare that the thesis entitled "**Comparison of Evolutionary Algorithms for Test Data Generation"** which is being submitted to the **Delhi Technological University**, in partial fulfillment of the requirements for the award of degree of **Master of Technology in Software Engineering** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any university or institution for the award of any degree.

Chirag Golechha

Department of Computer Engineering

Delhi Technological University,

Delhi.

DELHI TECHNOLOGICAL UNIVERSITY

CERTIFICATE

This is to certify that the project report entitled **"COMPARISON OF EVOLUTIONARY ALGORITHMS FOR TEST DATA GENERATION"** is a bona fide record of work, carried out by **Chiarg Golechha (2K12/SWE/12)** under my guidance and supervision, during the academic session 2012-2014 in partial fulfillment of the requirement for the degree of Master of Technology in Software Engineering from Delhi Technological University, Delhi.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Dr. Ruchika Malhotra

Asst. Professor

Department of Software Engineering

Delhi Technological University

Delhi

DELHI TECHNOLOGICAL UNIVERSITY

ACKNOWLEDGEMENT

With due regards, I hereby take this opportunity to acknowledge a lot of people who have supported me with their words and deeds in completion of my research work as part of this course of Master of Technology in Software Engineering.

To start with I would like to thank the almighty for being with me in each and every step of my life. Next, I thank my parents and family for their encouragement and persistent support.

I would like to express my deepest sense of gratitude and indebtedness to my guide and motivator, **Dr. Ruchika Malhotra**, Assistant Professor, Department of Software Engineering, Delhi Technological University for her valuable guidance and support in all the phases from conceptualization to final completion of the project.

I wish to convey my sincere gratitude to **Prof. Rajeev Kapoor,** Head of Department, and all the faculties and PhD. Scholars of Computer Engineering Department, Delhi Technological University who have enlightened me during my project.

I humbly extend my grateful appreciation to my friends whose moral support made this project possible.

Last but not the least, I would like to thank all the people directly and indirectly involved in successfully completion of this project.

Chirag Golechha

Roll No. 2K12/SWE/12

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Software testing is a very important process and plays a very important and key role in software industry. The cost of testing consumes a significant portion of the total project cost. Exhaustive testing is not possible and hence testing the focus is on testing those portions of the project or program where probability of finding fault is maximum.

Test Data Generation is an important part of testing and it is the process of creating a data set which is then applied on the new or revised software for testing it. Now a day's focus has shifted on automatic generation of test data which saves both time and effort. In the light of generating test data automatically various methods and algorithms are being developed and used so that the problem of generating test data can be solved not only efficiently but also in less time.

In this thesis we have compared three important Test Data Generation Algorithms namely Ant Colony Optimization (ACO), Artificial Bee Colony (ABC) and Genetic Algorithm (GA). These algorithms generate test path from CFG of the program and corresponding to that test data is generated that satisfies that path.

We have developed a tool named "TEST GENERATOR COMPARATOR" that takes input, CFG of 10 C programs and then the tool applies these three algorithms on each program CFG. Test data is generated by each algorithm for each input program, and for each algorithm and each program the tool outputs three parameters namely number of iterations, path coverage and the time taken.

Based on these three parameters the algorithms are compared. The results obtained show that ABC gives better result as compared to other two algorithms and hence is well suited for test data generation.

Keywords: Software Testing, Test Data Generation, Ant Colony Optimization, Artificial Bee Colony, Genetic Algorithm.

Chapter 1: Introduction

1.1. Introduction To Software Testing

Different people understand different definitions of testing. Some of them can be shown as:

1) Process that shows no presence of errors.
2) Checking whether a program performs its functions correctly or not is another purpose of testing.
3) Testing establishes confidence that a program does what it's supposed to do.

All the above definitions unfortunately are incorrect and describe almost opposite of what testing should be viewed:

"To execute a program so that the error in it can be found out" [1]

The goal of testing is to find critical situations of any program. Test cases shall be designed for every possible critical situation in order to make the program fail if such situation arises. In case if it is not possible to remove a fault then proper warning message should be given at proper places in the program.

The aim of the best testing person should be to try to fix all or most of the errors and faults. This is possible only when the intensions are to show that the program does not work as per the specifications, hence the definition given above is most appropriate.

This Chapter proceeds with discussing why software should be tested, who should do the testing and finally Test case generation

Why Should We Test? [2]

Software testing is no doubt very expensive and critical activity but releasing any software without proper testing is definitely more expensive and dangerous. It would be like a car running without a brake which no one would like to do.

The basic problem is that software cannot be released without adequate testing. The results after study may not be applicable universally but they give us some idea about the dept and how serious the problem is. When the software should be released is an important decision.

The cost of fixing errors increases as shown in Fig 1.1 [2].

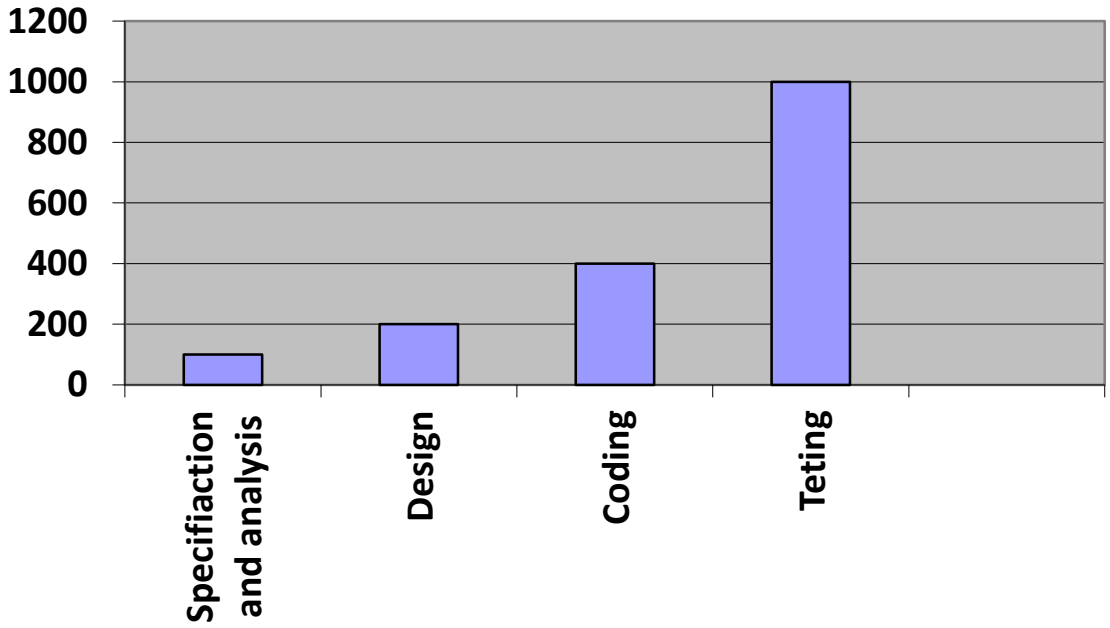


Fig 1.1 Cost of fixing errors with development stages

Testing continues to the point where the cost of testing process significantly outweighs the return.

Who should do the Testing?

Testing system software is not a responsibility of a single person. The role of software developer should be as small as possible in testing.

Since the developers are involved so intimately with the development of the software it becomes difficult for them to point out errors from their own creations.

The testing persons must be cautious and good communicators. One part of their job is to ask those questions that the developers might not be able to ask themselves. The testing people use the software just like an expert on the customer side.

Therefore most of the time testing people are made different from the development people keeping the overall benefit of the system in mind. The developers give guidelines during testing; however the overall responsibility is on the people who are involved in the testing.

Roles of the person involved in the testing during development and testing are given in Table1.1 [2]

Table 1.1 Persons and their roles during development and testing [2]

S.No	Persons	Roles
1.	Customer	Provides funds, provides requirements, approve any change and test the results.
2.	Project Manager	Project is managed and planned by him.
3.	Software Developer	He designs the software, builds it, participates in source review and testing, bugs, defects fixture done by him
4.	Testing coordinator	Test plan are created along with test specifications based on functional requirements and docunments.
5.	Testing Persons	Tests and documents the results.

1.2. Introduction To Test Data Generation

Generation of test data plays an important role in Testing. In this process data is created and then checked on software which can be old or new. Test data generation is done as follows:

1) Control Flow Graph Construction for an input program.
2) Selecting a path.
3) Generation of data for testing.

Generating test data is simple. A path is identified by the path selector. After identifying all the test paths, for every path input data is generated which results in execution of the selected path.

Manual Testing is not possible [2] and consumes a lot of time, automatic generation of test data can, not only reduce the work of testing but can help in the improvement of efficiency. Therefore the focus is on finding those algorithms that can help in automation of test data generation process and can also find faults in the program as early as possible without causing any other side effects.

1.3. Motivation Of the Work

The need of automatic generation of test cases and finding an efficient technique or algorithm for testing has its roots in the fundamental principles of Software Testing. These fundamental principles are as follows:

a. ***Exhaustive Testing is not possible:*** This implies that the entire set of possible test cases cannot be executed. Therefore it is important to find and apply only those test cases which are useful in testing all or almost all paths.

b. ***Early Testing:*** This implies that testing activities should be started early and move parallel with the development of software. Thus, test cases should be generated on the basis of requirement specification.

c. ***Testing Shows presence of errors:*** This implies that one cannot be assured that software is free from errors. It shows errors are present but cannot assure their absence.

d. ***Accumulation of errors:*** It means that, in a test object the errors are distributed unevenly and unequally. All errors may not be localized to same place in code but it is more likely to happen that some errors may be found where one error is found. Process of testing should be flexible and should respond to the behavior mentioned. Thus, all parts of code are not equally error prone. Hence, the need of effective test data generation arises.

e. ***Fading effectiveness:*** This implies that with the advent of time test suites become less effective. Repetition of test cases can't help in finding out new errors. If a function remains untested, it is possible that errors within it may not get discovered. To overcome this, test cases giving 100% coverage should be chosen.

f. ***Testing depends on context***. Every system is different and hence the method for testing them should also be different. For each system different testing intensity must be defined which tells when to stop testing.

g. ***False Conclusion, no errors means usable system:*** If an error is detected and removed, it does not guarantee that the system has become usable and meets the expectations of the users. Integrating units early and rapid prototyping prevents unhappy clients and discussions.

Besides there are several drawbacks in generation of test data:

1) The test data generated randomly may not be sufficient to cover all the paths in a program. The data generated may be able to detect only some portions of the code and the other portion may remain untouched.
2) As discussed in [3] Ant Colony algorithm for test data generation may prove to be ineffective when loops come into picture.
3) For some programs an algorithm may generate test data in small time duration while for some others it may take more time when compared to others.
4) Manual test data generation is a tedious task and hence there is a shift from manual testing to automatic testing of system software.

1.4. Problem Statement

Software industry faces a huge challenge in today's world. Every now and then market is boomed with new technology, new devices and softwares. These softwares are improving human life style and also providing them ease but they can also lead to catastrophic accidents. Most of the reasons which are given for such accidents are due to failure in the software which incurs due to some bug which was present during deployment of the software/system or due to some problem which occurred due to abnormal working of the software.

These are problems occur due to only one most important reason which is lack of proper testing.

Testing should start from the specification phase itself and should go on until the software dies or becomes outdated. Testing should be carried out in different phases of software development. Errors not only can occur in code, it can also be possible that the specifications are misunderstood, documentation is wrong or has wrongly written something, also come error can occur when the software is deployed on the user machine or system.

Hence it becomes necessary that testing should be done in every phase of software development.

Major setback with Testing as outlined above is that exhaustive Testing is not possible. This means that there should be focus on finding a technique that is suitable for testing in any condition and that gives test data that are effective for finding faults in software. Instead of using any random technique the testers now are required to find a method that is best suitable for most of the software for testing and this method should generate test data that satisfies all the constrains. Hence the test data generated should be effective in finding out all the errors that can occur in the software.

Several test data generation Algorithms have been used and are being used testing a software and to improving the quality of automatic generation of test data. A need is there to choose the best among all the algorithms for generation of test data, so that huge amount of time could be saved in testing and the focus could be shifted on finding those faults that result in faults and errors most of the time when software is working.

1.5. Goal of the thesis

The goal of the work in this thesis is summarized below:

a) ***To apply test data generation Algorithms on inputs programs:*** Three test data generation algorithms namely Ant Colony (ACO) [3], Artificial Bee Colony (ABC) [8] and Genetic Algorithm (GA) [20] are applied on some input C programs which are represented in the form of Control Flow Graph.

b) ***To compare the Three Algorithms for test data generation:*** All the three algorithms are applied on the input programs and then the results are obtained for each of the program. On each program these three algorithms ACO,ABC,GA are applied and then they are compared on the basis of the different parameters .The different parameters which are taken into account are number of iterations, total time taken by each algorithm for generating the test data and path coverage.

c) ***To find the most suitable and efficient algorithm for test data generation:*** On applying the algorithms on the input programs we obtain results based on different

parameters. These results are analyzed and then based on which the most suitable and efficient algorithm is decided that will be helpful in test data generation.

The aim at which our main focus is to find out the best among the three test data generation algorithms. The algorithms are compared based on three parameters which are:

1) No of iterations
2) Path Coverage
3) Time Taken

No of iterations measures that how much time the algorithms iterates in finding out the solution or in finding out the test cases which cover the independent paths. Since it is not possible to find and test all the paths in a control flow graph for a corresponding program, we shift our focus on finding independent paths and then generate data corresponding to independent paths. Hence the no of iterations that take place in finding test data which satisfies all the independent paths is used as a parameter to measure how efficient an algorithm is.

Path Coverage is another parameter that measures how many independent paths has been covered by the algorithm. Each algorithm iterates for a maximum number of iterations.

If all the independent paths are covered before that that means that the algorithm has the efficiency to generate test data that can cover all the independent paths.

Time Taken measures the running time of the Algorithm. It shows how much time an Algorithm takes in generating the test data for a given program that is given as input to the algorithm. It accounts for the running time of an algorithm for completely generating the independent paths or the time it takes to cover the paths before maximum possible iterations

It can be observed that our goals are focused on finding out the best algorithm among the three algorithms that are used. We aim to establish that using the most efficient algorithm will not only improve the quality of testing and reduce time but will also guarantee that the test cases which it will generate will be helpful in identify the faults to the most possible extent possible.

1.6. Organization Of Thesis

The organization of thesis is as follows:

Chapter 1: begins with Introduction to the Software testing, and discussing some general concepts like why testing should be done, who should do the testing. Next the concept of test data generation is discussed which is followed by topics like motivation of the thesis, problem statement, Goal of the thesis and in the end organization of thesis.

Chapter 2: discusses the work done by different people in the field of test data generation in the past. This includes the extensive study of various types of testing techniques and various test data generation algorithms applied on different programs using different methods that have been proposed in the literature so far. It also highlights some of the most relevant works in the field of Software testing using different algorithms and methodology.

Chapter 3: focuses on the general terms and concepts which are being involved in the generation of test data and are being used in the algorithms. Then the three algorithms namely Ant Colony Optimization (ACO), Artificial Bee Colony (ABC) and Genetic Algorithm (GA) are being discussed in detail that what are they and how they are used in test data generation followed by the algorithms which describes the steps that should be followed for each algorithm. Finally the framework for our tool "TEST GENERATOE COMPARATOR" has been shown along with the theoretical comparisons of all the three algorithms.

Chapter 4: describes the detailed implementation of the three algorithms. It illustrates the C programs that we have used as an input to the three algorithms. It shows how the algorithms are applied on the input programs. It also shows the snapshot of the tool used for the implementation of each algorithm.

Chapter 5: presents the results that are being obtained after the algorithms are applied on the input programs. The results consist of the graphs of the output, which compares the three algorithms based on the no of iterations, path coverage and total time taken for running the

algorithm. It shows which algorithm is better and efficient than the other two. Along with it a tabular representation of the result is also shown.

Chapter 6: presents the conclusion of the thesis that shows the goal that we have achieved in each of the chapter. It shows what we have discussed and successfully achieved in each of the chapter.

Chapter 2: Literature Survey

Software Testing is dedicated to finding errors in software. Complete testing of the program is neither feasible nor possible in today's scenario. Hence this situation has made the area of testing very challenging, where the question is:-

"How to choose a reasonable number of test cases out of a large pool of test cases" [2]

Researchers are putting their entire effort to provide answers to the above mentioned question in their own ways, however still selection of test cases is not a cakewalk and cannot be solved with a single method. [2]

Hence we may wish to touch a bottom line which may incorporate the following: [2]

1) At least once every statement of the program must be executed.
2) At least once all possible paths of the program must be executed.
3) At least once every exit of the branch statement must be executed.

Testing is divided by different researchers into different categories [2]. That can be shown in Fig 1.3

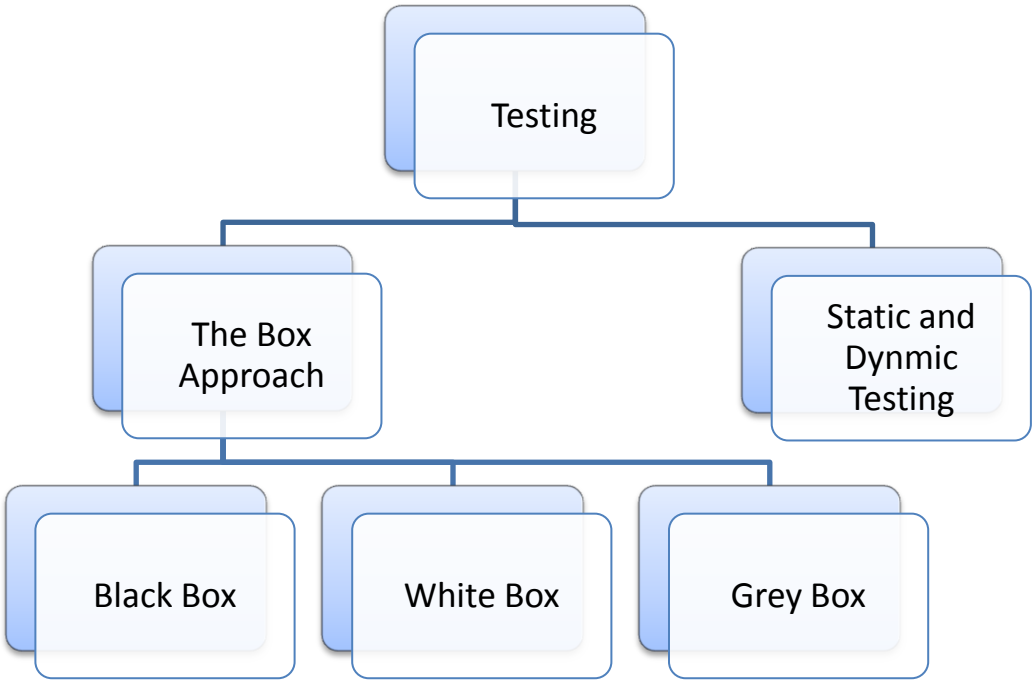


Fig 2.1 Classification of Testing

2.1 Static And Dynamic Testing

Static testing includes **Reviews, walkthroughs or inspections**, on the other hand checking a program code by executing it with given test cases is called dynamic testing.

Reviews:-Code Review is examining a source code systematically. It removes the mistakes that are not taken into account in the initial development and hence improving quality of software and skills of developer. Code reviews generally finds and removes mistakes such as format string, race condition, buffer overflows, thereby improving software's security. Study of types of defects in code reviews has given evidence that a large percentage of defects in code review, affect software [2].

Walkthrough: It is a type of peer review in which a developer guides members of the development team and others, through a product developed by him and then all the members ask questions and gives feedback about errors which can occur and any other problems.

Inspection: It deals with review of any product, by people who are trained in finding defects using well defined process. Inspection comes in reviews which are common in projects. Its aim is that, all the inspectors should agree with work product and give approval for its use in the project.

Static Testing is generally not that important hence can be ignored. Dynamic testing is done on program when it is in use. Dynamic testing may be done even when the program is not yet complete. Validation is done in dynamic testing and verification is done in static testing. [2]

2.2 The Box Approach [2]

The Box approach of testing is divided into white box, black box and grey box. These are discussed below:

2.2.1 **Black Box Testing:** Method of testing that checks what functions are performed by software without looking into its internal structure or working. It finds its use in each and every level of testing including unit, integration, etc. Using this method enables a

tester to know what a software does but gives no idea how. Black Box methods include:

2.2.1.1 **Boundary Value Analysis:** As the name suggests it focus on values which are either on boundary or close to boundary. The number of inputs selected by this technique is 4n+1, 'n' is the number of inputs. The boundary value selects values based on single fault assumption theory [2].

2.2.1.2 **Robustness Testing:** This is extension of boundary value. Here apart from valid inputs, invalid inputs are also selected. The total number of test cases in robustness testing are 6n+1 where 'n' is the number of input values.

2.2.1.3 **Worst Case Testing:** This is a special form of boundary value analysis where we don't consider the single fault assumption theory of reliability. Here those failures are also considered which occur due to more than one fault. Due to which the number of test cases increases from 4n+1 to 5^n test cases, where 'n' is the number of input variables.

2.2.1.4 **Robust Worst Case Testing:** Extension of worst case testing in which we add two more states i.e. just below minimum value and just above maximum value corresponding to invalid inputs. Hence the total number of test cases increases to 7^n test cases.

2.2.1.5 **Equivalence Class Testing:** As the name suggests it divides the data into categories or classes each having equivalent data. It reduces total number of test cases which should be used by using test cases which reveals the classes of errors. It reduces the time for testing.

2.2.1.6 **Decision Table Based Testing:** They provide us with a small and accurate method to present complicated problems and their solutions. Like any other decisional and control flow statements, decision table gives what should be done if a certain situation occurs and this is done in an elegant way

Table 2.1 Decision Table Components

The four quadrants	
Conditions	Condition alternatives
Actions	Action entries

2.2.1.7 **Cause Effect Graphic Technique:** This technique is a popular technique for small programs and takes into account the combinations of various inputs which were not available in earlier mentioned techniques. The Fig 1.3 shows below the step by step process of Cause Effect Graphic Technique.

Fig 2.2 Cause Effect Graph Technique [2]

2.2.2 **White Box Testing:** As the name suggests is a testing in which tester can see through the working of the software hence it is also called clear box testing and since the internal structure are also available for testing it is called structural testing [2] .It tests the working of a software instead of, its functionality. White box test techniques include:

2.2.2.1 **Control Flow Testing:** It is very popular because of its simplicity and effectiveness. Some of the techniques which are part of control flow testing are discussed :

Statement Coverage: Takes care that every statement of the program is executed in order to achieve 100% coverage.

Branch Coverage: Every branch is tested. Does not guarantee 100% path coverage but it guarantee 100% statement coverage.

Path Coverage: Every path is tested of the program. If it is not possible at least all the independent paths of an input program should be executed

2.2.2.2 **Data Flow Testing:** It helps to minimize those mistakes which can occur with Control Flow graph testing. Ii does not involve data flow diagram. It is based on variables, their usage and the definitions of the variables in the program. Its main concern is on where a variable is defined and where it is used. Since a variable if defined is used as long as the program works and hence this technique focuses mainly on where a variable is defined and used

2.2.2.3 **Slice Based Testing:** In this technique various subsets (called slices) of a program are prepared with respect to its variables and their selected location in the program to be tested. Each variable with one of its location gives a program slice.

2.2.2.4 **Mutation Testing:** A popular technique to access the effectiveness of a test suite. We may have a large number of test cases for a program and have neither time nor resources to execute all of them. We may create a test suite selecting some of the test cases and then use mutation testing to assess the effectiveness and quality of the test suite and if found not adequate can also improve the same.

2.2.3 **Grey Box Testing:** It is a combined effect of white box and black box testing. It finds and searches defects if an application is not used properly or it possesses a structure that is not proper. It requires the document which describes what the application does so that its test cases can be defined. It provides the advantage of both black and white box testing. Generally used for testing web based applications.

Grey box testing techniques are as follows:

2.2.3.1 **Matrix Testing:** It provides what is the status of the project.

2.2.3.2 **Regression Testing:** It means that if any changes are made to the software then the testing should be done again on it.

2.2.3.3 **Pattern Testing:** Verifies whether an application is good or not on the basis of its design and its structure.

2.2.3.4 **Orthogonal Array Testing:** Of all the possible combinations how many subsets can be made decides the usage of Orthogonal Array testing.

2.3 Previous Work Done

Testing software is a problem that is not new to the industry. The day when the software was evolved, the problem of testing evolved with it simultaneously. A lot of work has been done from then in order to solve the problem and to come to conclusion in which they can decide how the problem can be best solved. A lot of testing techniques are being discussed and are used in generating test cases. But along with these techniques some algorithms are also used which improves the work and makes it easy. Different testing algorithms are being used in order to solve the problem and each Algorithm uses different techniques.

Algorithms that are inspired from nature have gained attention of the researchers from a long time. They have used them as an example to model many problems that are faced in today's world [22]. The popularity of these algorithms has increased as they are able to solve problem easily and in a short duration of time. Due to these reasons some of the algorithms came into existence namely Ant colony, Evolutionary, Particle Swarm Optimization, Genetic, Bee Colony etc.

Praveen Ranjan Srvivastava et al. [3] have used Ant colony Algorithm for generation of optimal path from given CFG.They have taken input a CFG for a software and then applied ACO on it to calculate various parameters like feasibility of path, heuristic information probability value, pheromone which is then used to calculate the strength of the independent paths found from the CFG and based on which the priority to each path is assigned. Hence they have focused on test case prioritization using Ant Colony Optimization.

Kewen Li,Zilu Zhang Weying Liu[4] have used Ant colony optimization for automatic test case generation and have shown through experiments that it has better performance than use case and genetic approach used for the same purpose. They have used a basic search model based on which they have calculated the shortest path and hence the test cases. They have modified and proposed a new ant density model which is used to update pheromone

$Q= ((M-|f|) / M)*K$

"*M*" is a positive number and has value slightly greater than the sum of every branch functions, *"f"* gives branch function's sum and "*K*" denotes a constant

Huaizhong LI, C. Peng LAM [5] have applied ant colony in state based testing for generation of test sequence automatically. They have made use of UML state chart diagram to implement the algorithm and generate test cases automatically.**Ahmed S.Guiduk [6]** has combined the approach used by Praveen Ranjan Srivastava [3] for test path generation with a slight modification that the pats are generated corresponding to def-use pairs in the control flow graph and approach used by Kewen Li[4] for generation of test data corresponding to the independent paths generated. Here instead of control flow graph they have used def-use graph.

Chengying Mao et al. [7] have used ant colony optimization for generating test data for structural testing. The local transfer rule, global transfer rule and pheromone update rule are re-defined for the Ant Colony Algorithm, to handle the continuous input domain searching. Experimentally it is shown that the algorithm outperforms the existing simulated annealing and genetic algorithm in most cases

Soma Sekhara Babu Lama et al. [8] have used Artificial Bee Colony Algorithm for generating independent paths that are feasible from given CFG and then optimizing the test suite with the help of the approach proposed. They have generated a CFG for the input program and then have applied ABC algorithm to find the fitness values of each node which is then used to generate the independent paths. After these paths are generated, random test data is generated and they are then optimized for the paths generated based on parameter like probability and total fitness. Finally new test cases are generated for the paths which are having probability less than average probability.

AdiSrikanth et al. [9] have proposed a new ABC approach for automatic test case generation and test suite optimization. They have shown that the proposed ABC approach is better than

the previously used approaches for test suite optimization. The approach not only returns feasible paths but also returns paths that are not feasible, if a path cannot be reached using test data generated. **Surender Singh Dahiya et al. [10]** have applied ABC algorithm using symbolic execution method to generate test cases. They have used branch predicate based fitness function. Hence symbolic execution method has been used based on static testing which first selects the target from the CFG of program and then generation of inputs take place which uses the ABC method in which composite predicate corresponding to the target path is satisfied. **D. Karaboga , B. Basturk [11**] have used Artificial Bee Colony for multi dimensional numeric problems which optimizes numerical test functions that are large in number and compares results with that of differential evolution (DE), evolutionary algorithm (EA) and particle swarm optimization (PSO). Experimentally it is shown that ABC is better than the rest of the mentioned approaches.

Adil Baykasolu et al. [12] have discussed the application of ABC algorithm to generalized assignment problem (GAP). They have used the method of neighborhood shifting. Initially each set of agents are assigned with their particular number of tasks. And based on the working calculated the fitness function for the task is calculated. After that the agent-task relationship is rearranged by shifting or double shifting the agents and allocating it to some new set of agents.

D. Jeya Mala and V. Mohan [13] have presented a non-pheromone-based test suite optimization approach. .The employed bee covers all of the coverage and generates the test case along with a happiness function which is given by the heuristic value of each of the test cases. They have shown a comparative analysis of Genetic algorithm (GA) and ABC algorithm in test suite optimization. The framework that is used by them is shown by the Fig 2.3 [13].

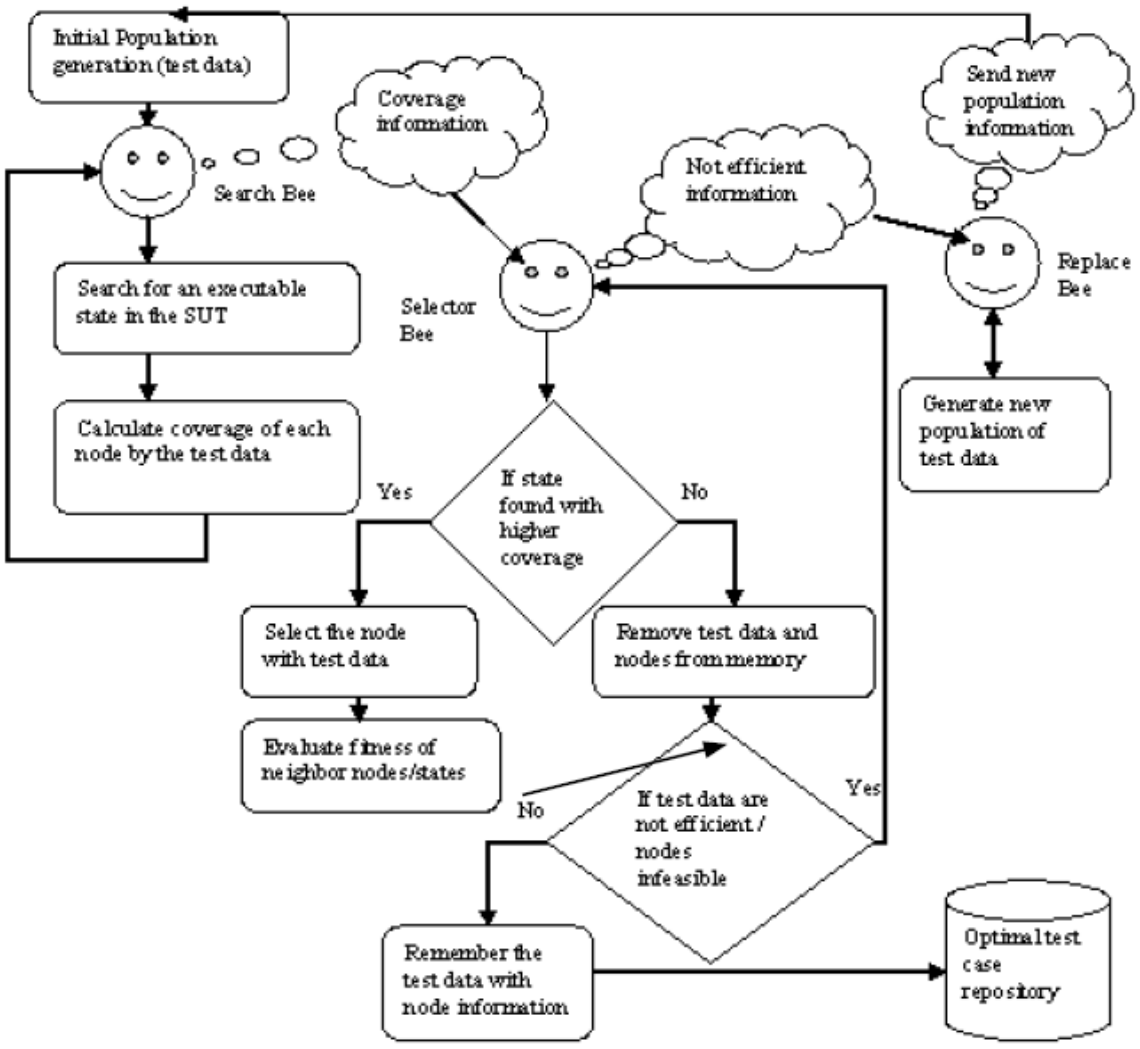


Fig 2.3 Flow Chart of Algorithm proposed in [13]

Dr. Arvinder Kaur,Shivangi Goyal[14] have given a new algorithm that optimizes and maps behavior of natural bees to search and collect food so that it can be used to prioritize test cases of regression test suite. Prioritization of test suite has been done by observing the behavior of scout and forager bees. Test cases denotes food sources, food source quality is denoted as the number of conditions that are detected on executing each test case and number of test cases denotes how many scout bees are there, and the initial population consists of number of forager bees.

The algorithm proposed by them is shown in Fig 2.4:

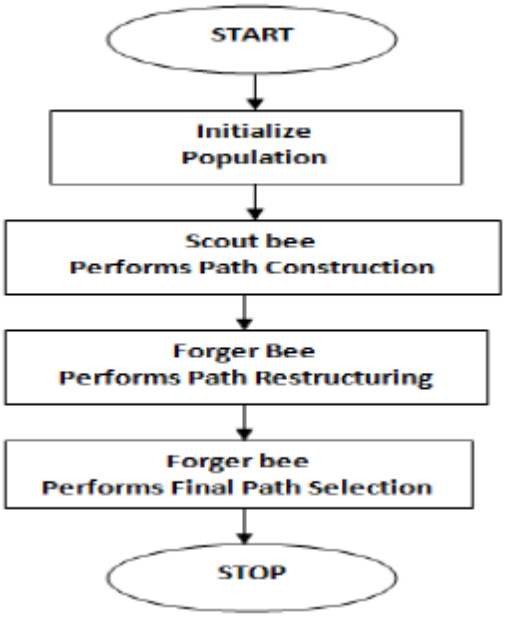


Fig 2.4 Flow Chart of Algorithm proposed in [14]

They have also concluded that the running time of ABC is O (n^2)

Roy P. Paras et al. [15] have presented a technique for generation of test data automatically using Genetic Algorithm. They have implemented a tool that used parallel processing so that the performance for searching can be improved. They have also implemented a random test data generator. They have made use of Control dependence graph(CDS) instead of Control Flow Graph (CFG).They have analyzed the complexity of using CDS for generating test data using Genetic Algorithm. **Li Bin el al [16]** has reduced the problem of generation of test data to a minimizing function. In order to enhance the computational efficiency some improvements have been made to form genetic stimulated annealing algorithm and has compared the Genetic Algorithm, Genetic Stimulated annealing algorithm and random test case generator to assert that stimulated annealing gives better results than the other two**.** **Praveen Ranjan Srivastava et al. [17]** have developed a variable length genetic algorithm. Path clusters that are critical in a program are identified and then those clusters are selected that most critical. **Jin-Cherng et al. [18]** have developed genetic algorithm by making use of normalize extended hamming distance, a metric which selects test cases that should survive so that fitter test cases of next generation can be produced. Based on the metric they have also developed a fitness function named similarity which determines which test cases should

survive.**Moheb R. Girgis [19]** have used genetic algorithm which instruments the program to be tested and then takes it as input, def-use list which shows all the associations that need to be covered, number of variables that need to be input along with their precision and domain. A comparison of roulette wheel and random selection for parent selection is done to find out which one is effective and has experimentally shown that method of random selection is better when compared to roulette wheel method.

Ahmed S. Ghiduk and Moheb R. Girgis [20] have used genetic algorithm for automatic test data generation. In order to reduce the cost of testing, the concept of dominance relation between nodes was introduced by them and hence a new fitness function for evaluation of test data generated was defined. They have compared random testing with the genetic algorithm introduced and have shown that the later gives better result than the former.**Peng NIE [21]** has discussed the particle swarm optimization algorithm with enhanced exploration ability for test case generation to overcome the problems faced by the previous particle swarm optimization algorithms by improving the prematurity and enhancing the efficiency of test case generation.

Chapter 3: Research Methodology

In this chapter we will describe and discuss the machine learning algorithms that are being used by us namely Ant Colony Optimization (ACO), Genetic Algorithm (GA) and Artificial Bee Colony Optimization (ABC). We will describe each algorithm in detail i.e. what does each algorithm does and how it is applied in test data generation. After that we will give a step by step procedure of how the algorithm is applied. Finally the framework that we have proposed for our tool "Test Generator Comparator" and along with a theoretical comparison of the three algorithms has been shown.

3.1 Ant Colony Optimization (ACO)

Ants like humans are animals that are social. They live together and form colonies and the work done by them is directed towards the survival of the entire colony instead of a single ant. Every ant has same ability and no one possess any special abilities. Pheromone, a chemical substance is used by ants to communicate with each other. Communicating in this way helps them to do tasks which are complex in nature such as finding shortest path from their colony to a particular location.

Using this behavior of ants, algorithm for optimization is developed and proposed so that this behavior of ants can be used to solve problems.

While searching for the food, an ant comes across paths that lead to destination and contain pheromones. It then selects that path which contains high concentration of pheromone as compared to other paths with a certain probability. After the path is chosen, an ant deposits certain quantity of pheromone of its own in the path, thereby increasing its concentration.

Always the same path by ants to return to their colony and hence while returning back other portion of pheromone is also deposited. Lets us suppose that the two ants at the same location choose two different paths at the same time. Suppose that two ants choose two different paths at same location and at the same time. The pheromone concentration will increase faster on shortest path as compared to other as the ant choosing this path will returns and hence it will deposit more pheromone in smaller duration.

The concentration of pheromone in the shortest path will be much higher than the concentration of the pheromone in the other paths if the entire colony follows this behavior.

And hence choosing any other path will be less probable and only few ants fail in following the shortest path.

Pheromone concentration involves another phenomenon. The concentration of pheromone on the path becomes less as pheromone being a chemical substance evaporates in air and eventually vanishes as the time passes. Hence the path used less will have lower concentration of pheromone than the path mostly as not only the concentration increases in other paths, but also the concentration of pheromone decreases.

The above process of ant colony can be better understood with the help of the following Fig 3.1 [6]

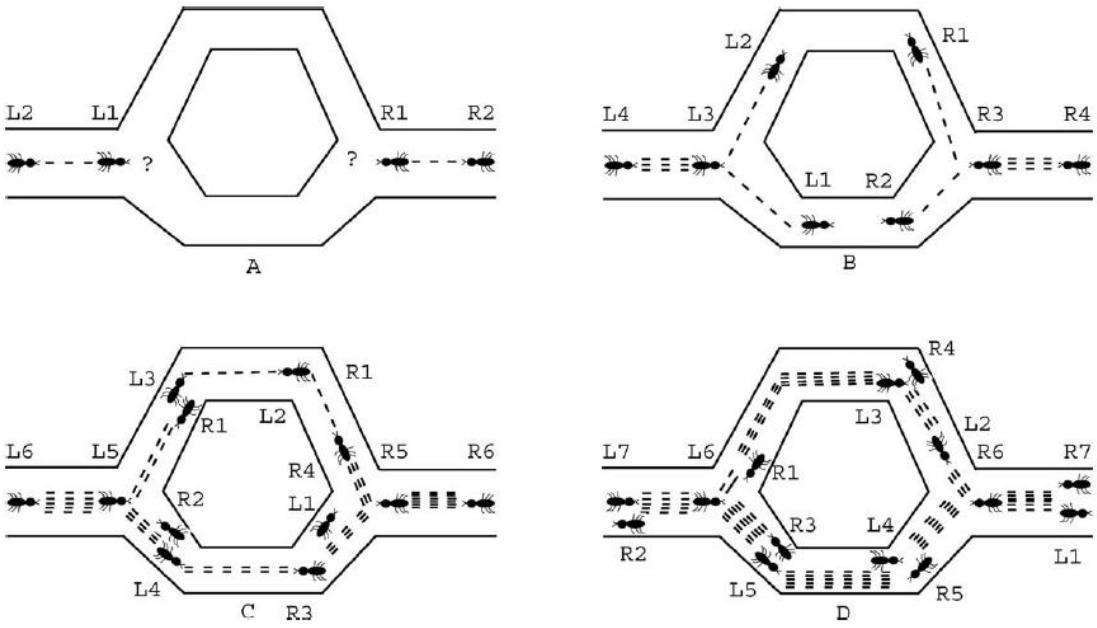


Fig 3.1 Path finding mechanism of Real Ants [6]

The above figure shows the path finding mechanism of real ants:

a) While traveling to a destination, ants arrive at a decision point.
b) Randomly some ants choose lower path, some upper.
c) Ants choosing shorter path tend to reach to the opposite location faster than the one choosing longer path as they all travel at the same speed.
d) On the shorter path, pheromone accumulated at a higher rate. Amount of pheromone deposited by ants is approximately proportional to the number of dashed lines in the above Fig 3.1

3.1.1 Background for the Algorithm [3]

This method generates test data automatically from the Control Flow Graph (CFG). CFG is a type of directed graph G= {V, E}, where vertices V corresponds to states and Edges E corresponds to the paths between the nodes.

All the paths must be covered at least once in the CFG of the program using this algorithm. What path should be selected depends upon its probability. The higher the probability, the higher are the chances that a path will be selected. Probability value of a path depends on: [3]

1. **Feasibility of path (F_{ij}):** denotes that the vertices are connected directly.
2. **Pheromone value (τ_{ij}):** helpful to ants so that in future they can make right decision.
3. **Heuristic information (η_{ij}):** At a current vertex what path is visible by an ant is denoted by heuristic information.

If the selected feasible paths have equal probabilities then follow these steps: [3]

- If next node is end node, than ant will select that node means path form current node to the end node will be selected.
- Visited status parameter's value (V_s) is used to select next node by ant. If there is a connection between two vertices V1 and V2and ant has not visited V1, then V2 will be selected as the next state. The criteria that all states are covered at least once, is fulfilled through this concept.
- Any feasible path is selected randomly if not selection is possible using above conditions.

Proposed algorithm can find out all feasible paths information from its current state. An ant is associated with four other information about: visited states with the help of visited status (V_s), Heuristic information for the paths (η_{ij}), level of pheromone on path (τ_{ij}), and last is probability parameter (P). Pheromone level and heuristic values are updated after selection of a particular path. Increase in Pheromone level depends on last pheromone level and heuristic information on the other hand heuristic information, depends on previous heuristic information.

If a vertex 'i' is connected to vertex 'j' that means that there exist a path from vertex 'i' and 'j' i.e. (i→j). Every path in a graph is associated with five tuples: Fi_j (p), τi_j (p), ηi_j (p), V_s(p) and P_{ij} (p), where (p) shows an ant p and the value of tuple associated with it .These attribute are described below:

1) **Feasible path set F: = Fi_j (p),** connection from vertex 'i' to its next vertex "j" is shown by it. Vertices are adjacent to 'i' if there is a direct connection.
    - If Fij=1 shows that path is feasible between "i" and "j".
    - If Fij=0 shows that path is feasible between "i" and "j".

2) **Pheromone trace set τ: = τ_{ij} (p)** shows that on the path (i→j) which is feasible from one vertex to other vertex what is the level of pheromone. After a path is traversed, the pheromone level is updated. Helpful for ants to make any decision in future.
3) **Heuristic set $\eta = \eta_{ij}$ (p):** From current vertex 'i' to vertex "j", what is the visibility of a path for an ant is shown by it.

4) **Visited status set Vs:** All the states which are already traversed by the ant p, information about this is shown by it. For state 'i' :
    - If V_S (i) =0 indicates that ant p has not visited vertex "i"..
    - Whereas if Vs (i) =1 shows that ant p has already visited vertex "i".

5) **Probability set:** Probabilistic value, value of pheromone τ_{ij} (p), feasibility of path F_{ij} (p) and heuristic information η_{ij} (p) of path, all these are considered by an ant p for selecting a path. Two more parameters α, β are there, which are helpful and used to calculate the probability of a path. α and β are the parameters that control the desirability versus visibility. Pheromone and heuristic value of the paths are associated with α and β respectively.

3.1.2 Algorithm for ACO [3]

Here we will present the algorithm that we have used in path generation from the CFG of the input program and then test data generation for each path. The algorithm is as follows:

Initialize all parameters;

1.1 Set heuristic Value (η): Heuristic value η=2, for every path in the CFG, is initialized.

1.2 Set pheromone level (τ): Pheromone value τ =1, for every path in the CFG, is initialized.

1.3 Set Visited status (Vs): Vs is initialized to 0 for every state showing that no state is visited initially by the ant.

1.4 Set Probability (P): Probability P is initialized to 0, for each path.

1.5 Set α=1 and β=1.

1.6 Count is set equal to Cyclomatic complexity which gives the number of independent paths in the program.

1.7 Key is set equal to end node (end_node).

2 while(count>0)

2.1 Start is initialized to i, sum and visit is initialized to 0.

2.2 Update the track: Visited status is updated for the current vertex 'i', i.e. Vs[i] =1 if (Vs[i] ==0) and visit=visit +1.

2.3 Evaluate Feasible Set: For the current vertex 'i', F(p) is determined and all possible path are evaluated from the current vertex 'i' to all the neighboring vertices with the help of CFG. Go to step 3, if no path is feasible then.

2.4 Sense the trace: Probability for all the non- zero connection in the feasible set for current vertex 'i', is evaluated. For every non-zero element in F(p), probability is calculated using the following formula:

$$Pij = \frac{(\tau ij)^{\alpha} * (\eta ij)^{-\beta}}{\sum_{1}^{k} (\tau ij)^{\alpha} * (\eta ij)^{-\beta}}$$

2.5 Move to next vertex: Using the below mentioned rule move to next vertex:

R1: Select Path (i→j) whose probability (P_{ij}) is maximum.

R2: If more than two paths have equal probability then selection is made path in accordance to the rule below:

R2.1: Entry present in the feasible set is checked whether it is the end node or not. If (feasible set entry ==end_node) then end_node is selected as next node Otherwise follow R 2.2

R2.2: Select the path whose next state is not visited yet ie Vs=0. If same visited status is same for two of more then follow R2.3

R2.3: If Vs[j] ==Vs[k] then random selection of path is done.

2.6 Update the parameter

2.6.1 Update pheromone for path (i→j) according to the following rule:

$$(\tau ij) = (\tau ij)^{\alpha} + (\eta ij)^{-\beta}$$

2.6.2 Update Heuristic

$$\eta ij = 2 * (\eta ij)$$

2.7 Calculate Strength

Sum=Sum+ τ_{ij}

Strength [count] =Sum

Start=next_vertex

2.8 If(start !=end_node)

Then go to step 2.3

Else

If (visit ==0) then discard the path as it is redundant path otherwise add new path

2.9 Update count: decrement count by 1 each time

Count=count-1

3 End

The above algorithm generates only the independent paths along with the strength of each path generated. For test data generation we randomly generate the test data and then check whether these test data satisfies one of the independent paths or not. The following search model is applied for test data generation as shown in Fig 3.2 [4]

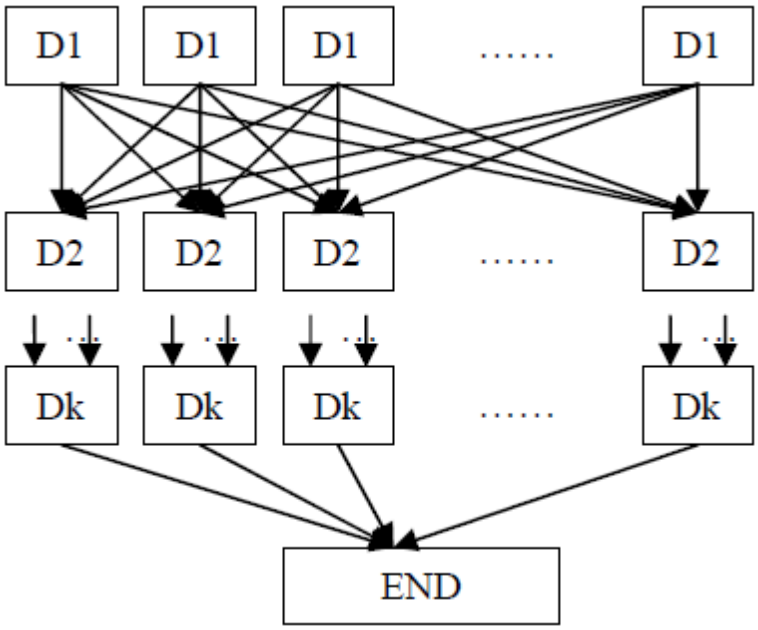


Fig 3.2 ACO Searching model for Test Data Generation [4]

Random values are assigned to input variables from the input domain and then these values are used to evaluate a path. If the data satisfies that path then that value of input data becomes the one of the test case for the program and if not then that value is modified so that new value can be generated. In this way test data is generated to cover the independent paths.

3.2 Artificial Bee Colony Optimization (ABC)

The Artificial Bee Algorithm can be implemented on a wide range of problems including includes global optimization as it is a population-based evolutionary method, it shows similar behavior with swarm intelligence algorithm. As the name suggests it is derived and motivated from the behavior of natural bees. It is all about how honey bees store extra nectar for their survival in the season of winter by distributing their work and collective foraging strategy. They follow a certain protocol for collecting nectar and hence this search behavior and intelligence drives the researchers to use ABC for test case generation and optimization. Bees aim on finding out those sources of food which have high nectar amount. One of the most important things is how they communicate with each other. Bees use waggle dance during food procuring.

The direction of bees indicated the direction of the food source in relation to the sun, while performing waggle dance, how far the food source is indicate by the intensity of the waggle dance indicates and the duration indicates the amount of nectar in the food source.

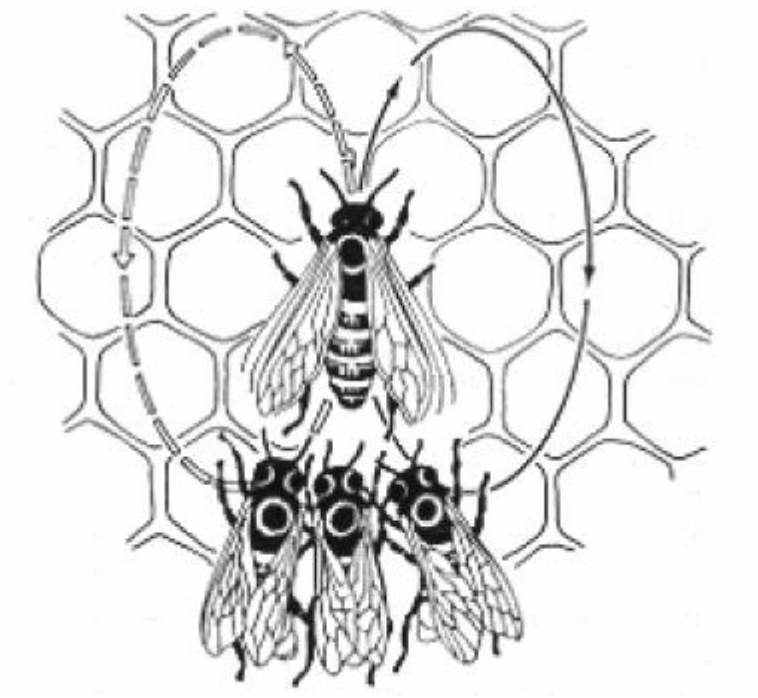


Fig 3.3 Waggle Dance of honey bees [12]

Bee Colony system consists of two essential components:[12]

- **Food Sources:** It depends on different parameters such as how near the food source is, how easily it can be extracted.
- **Foragers:**
  - **Unemployed Foragers:** If the knowledge about the food source is available in the search field, search is initiated by bee as an employed forager. Two possibilities for an unemployed forager are there:
    - Scout Bee (S in Fig 3.4): A scout bee is one which without any knowledge, starts searching spontaneously.
    - Recruit(R in Fig 3.4).If the waggle dance done by some other bee, is attended by the unemployed forager, than using the knowledge of waggle dance it starts searching.

- **Employed Foragers** (EE in Fig 3.4): If the food source is exploited and found out by the recruit, it raises to be an employed forager, whose work is to memorize the location of the food source. Some portion of nectar from the food source is loaded by the employed bee and then when it returns to the hive it unloads the food source to the food area in the hive. There are three options related to residual amount for the foraging bee:
  - The food source is abandoned by foraging bee, if the nectar amount is decreased to a low level or exhausted, and then foraging bee becomes an unemployed bee.
  - It can continue to forage without sharing the food source information with the nest mated, if the amount of nectar is sufficient in the food source.
  - Or it give information about the same food source to other nest mates by performing
- **Experienced Foragers**: Historical memories for the location and quality of food sources are used by them.
  - Can be an inspector by controlling the recent status of food source already discovered.
  - Can be a reactivated forager who makes use of information from waggle dance. (RF in Fig 3.4).
  - Can be a scout bee for searching new food sources if the entire food source gets exhausted. (ES in Fig 3.4).
  - Can be a recruit bee that searches new food source which is shown in the dancing area by another employed bee. (ER in Fig 3.4).

The honey bee foraging behavior discussed above can be depicted in the following Fig 3.4 [12]

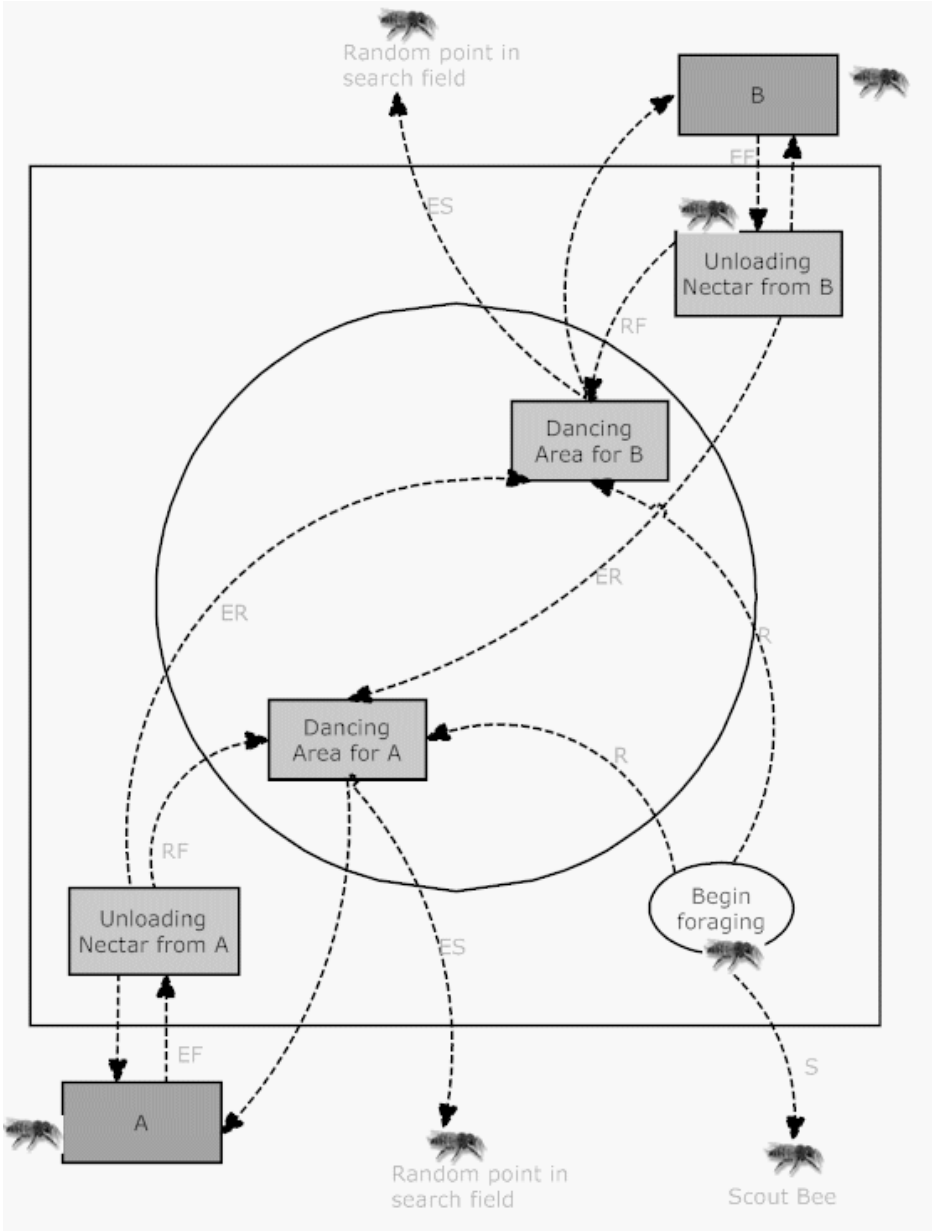


Fig 3.4 Bee foraging behavior [12]

3.2.1 Background for the Algorithm [8]

The approach discussed, generates test path and test data from the CFG diagram automatically. CFG is the form of directed graph G = {V, E}, where V indicates vertices to states, and E shows edges between the nodes in the path.

ABC algorithms is run so that at least once optimal path is covered in the CFG of the software on which testing is done. Fitness of the node of the path decides which path should be

selected. Fitness values which are associated with the nodes are randomly generated and then based on these values the node having minimum fitness value is selected as the next node.

ABC techniques which we will discuss here consist of two strategies that are proposed using Artificial Bee Algorithm [8].

1) First is generation of independent path that are feasible using ABC and

2) And the second one is test suite optimization using ABC

3.2.2 Algorithm for ABC [8]

Feasible Independent Path Generation

1. CFG is constructed for the given code.
2. All nodes of CFG are initialized with random fitness values in small ranges (should be analogous to scout bees).
3. Cyclomatic complexity is calculated using no. of Predicates Nodes + 1, where predicates nodes are the decision nodes in the CFG.
4. Start node is selected as initial node and neighboring nodes are identified (like employed bees).
5. Node with has least fitness value is selected and its fitness value is increased by one unit and that node is added to visited list.
6. If required loop optimization function is applied (as discussed below) for selected node.
7. Selected node is made as initial node and new path is generated using Steps 4-6 until last node is reached.
8. If new path generated from above Step already exists, don't save that path, go to step next step.
9. Steps 4-8 are repeated until total number of iterations are equal to 2*cyclomatic complexity.

Loop Optimization Function

This function checks the node that we have selected already exists in the traversed path or not; if it does, the position 'i' of the selected node from starting node is obtained by this function and then the fitness value of $(i+1)^{th}$ node is increased. In this way in the next iteration, $(i+1)^{th}$ node is not selected hence giving chance that next time new path will be found.

Test Suite Optimization using ABC:

All the independent paths are classified into square root (SQRT) of total number of independent paths sets, based on coverage [8]. Based on the comparison criteria among the independent paths, sets are formed. The algorithm for this can be stated as:

1. Randomly generated test cases (food sources) are initialized, similar to scout bees in Bee Colony.
2. All these test cases (food sources) are evaluated for the given program.
3. Continue to next step if a criterion is not satisfied, else stop (all test cases generated).
4. Using comparison approach, independent paths are grouped into SQRT of total number of independent path sets i.e. independent paths having more number of similar edges are grouped together.
5. Check whether all independent test path coverage criteria is satisfied or not. If it is not satisfied run Bee Thread Algorithm.

Bee Thread Algorithm:

1. Fitness values and probability values are calculated for independent test path.
2. New test cases are generated depending on the number of independent paths having probability value less than minimum qualifying criteria.
3. Test cases that are satisfied and remaining are replaced by its neighbors like employed bees.
4. All these test cases are evaluated for given program.
5. Return to step 5 of above algorithm.

Algorithm for Generation of Test Suite

1. The random population of test data X_{ij} is initialized, within input domain where test case number is denoted by 'i' and index of variable in that test case is denoted by 'j'.
2. Test data generated is evaluated.
3. Check whether all the independent paths are satisfied or not. If not go to next step else exit.
4. Independent paths are divided into sets.
5. Fitness and probability values are calculated for each test data as discussed below:
6. Average of the probability value is calculated and is named as P_{avg}.
7. If value of any path probability is less than P_{avg} then generate new test data in the neighborhood of X_{ij} by making use of the following equation:

   $Y_{ij} = X_{ij} + RAND\ (-1,0,1) * (X_{ij} - X_{kj})$

   Where RAND (-1,0,1) is random number from {-1,0,1} and X_{kj} is neighbor test data, k= (i+1) % number of parameter

8. Populated test data is evaluated.
9. Check whether all independent paths are satisfied or not. If not go to Step-4 else terminate.
10. Repeat step 4-8 until solution for all individual test paths is obtained.

Fitness and Probability Calculation

Test path sequence comparison method is used as the fitness value function so that individual path coverage criteria can be achieved in ABC algorithm. Comparison of each unsolved independent path against each solved independent path takes place, in finding out the fitness value. All solved independent paths are considered to have ideal fitness value and their probabilities value are assumed as 100%, in normalized form these values are taken as 1

1. One solved independent test path is selected, we call it as $SP_{ij.}$
2. Each unsolved test path $UP_{ij\ is}$ compared with the selected solved test path SP_{ij} in such a way that if nodes in both the paths are same then assign value'1' else '0'.
3. All the assigned values in the path UP_{ij} are added to the $FITNESS_i$ (variable used to store fitness).

4. For all unsolved individual test path, repeat step 2 to 3.
5. Maximum fitness value in all the unsolved independent test paths is found out, it is named as MAXFITNESS.
6. Final fitness value corresponding to each solved independent path is computed using

   $FINAL\_FITNESS_i = FINAL\_FITNESS_i + (FITNESS_i)/MAXFITNESS$

7. Repeat step 1-6 for all independent paths that are solved.
   The probability can be calculated using the below formula:
   $PROBABILITY\_VALUE_i = FINAL\_FITESS_i \ / \ \sum FINAL\_FITNESS_i$

3.3 Genetic Algorithm (GA) [20]

Genetic algorithm, now days abbreviated to GA, was first used by John Holland, whose book *"Adaptation in Natural and Artificial Systems"* of 1975 played a key role in creating something which is now a flourishing field of research and application that goes beyond the original GA. In order to cover the developments of the last 10 years, the term evolutionary computing or evolutionary algorithms (EAs) is now used. However, in the context of meta-heuristics, it is probably fair to say that GAs in their original form encapsulate most of what one needs to know.

A variety of problems involving search and optimization are solved by applying GA. Mechanism of evolution and natural selection from the basis of GA search methods. Natural search and selection processes form the source of inspiration for GA, which leads to the survival of the fittest individuals. Mutation and crossover are used for searching and for generating a sequence of population, selection mechanism is used.

In Genetic Algorithm first generation begins with a set of initial individuals, which are selected randomly form the problem domain. The present generation is transformed into a new and fitter generation by performing series of operations using this algorithm.

Evaluation of each individual in each generation may approach the optimal solution. Operations of genetic algorithm are designed in such a way so that solutions produced by them are efficient. These operations are shown below:

1. **Reproduction:** Based on the output of fitness function, this operation assigns each individual reproduction probability. Greater probability for reproduction is given to individuals having higher ranking. Hence the individuals which are fitter stand a better chance of survival from one generation to next.
2. **Crossover:** Descendants that make up the next generation are produced using this operator. Following crossbreeding steps are applied for it:[20]
    i. Two individual are selected randomly as a couple from the parent generation.
    ii. Corresponding to this selected couple, select a position randomly of the genes, as the crossover point.
    iii. First parts of both the genes are exchanged, corresponding to the couples.
    iv. The two resulted individuals are added to the next generation.
3. **Mutation:** A gene is picked at random and according to the mutation probability its state is changed using this operation. To maintain the diversity in a generation, mutation is done so that premature convergence to a local optimal solution can be prevented. As there is no definite way, the mutation probability is determined based on intuition.

After crossover and mutation operations are completed, an original parent and a new offspring population will be there. A fitness function is devised to find out, which of these parents and offspring's can survive into the next generation. These parents and offspring's are filtered to form a new generation, after the fitness function is performed.

Until the desired goal is achieved, perform these operations repeatedly. GA assures with high probability that the quality of the individual will improve over several generations. The structure of simple GA can be shown as Fig 3.5

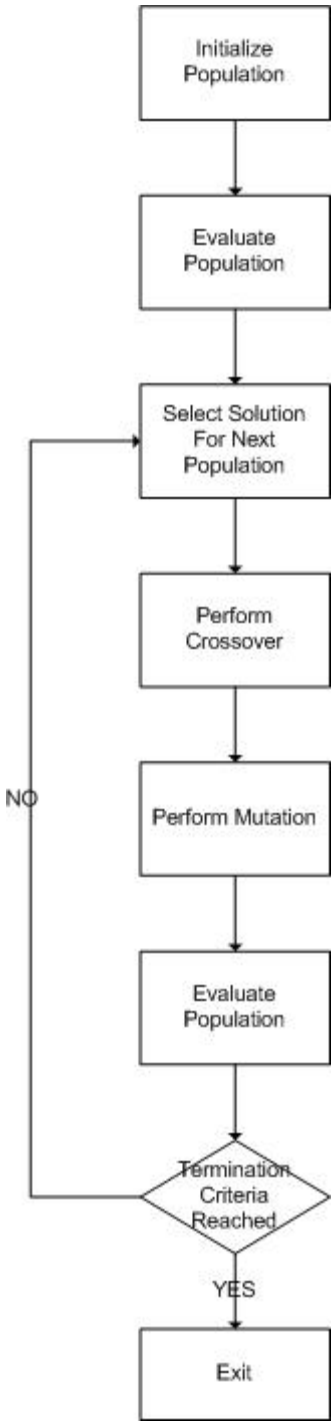


Fig 3.5 Basic Flow of GA

3.3.1 Background for the Algorithm

Control Flow Graph

A control flow graph or CFG is used to analyze the program structure. Control flow graph represents the program graphically. It is a directed graph G= (V, E) where V is the set of vertices and E is the set of edges.

Dominance and Denominator Tree

Let us consider a CFG G= (V, E) with two nodes n_o and n_k. If every path P from the entry node n_o to a node *m* contains node *n*, then *n* dominate *m*. We can obtain a tree rooted at n_o, by applying the dominance relations between the nodes of a CFG g. This tree is called the Dominator Tree DT (G).

A Dominator Tree is a diagraph rooted at node n_o and every node *n* except n_o is head of just one arc and from root n_o to each node 'n', there exist a unique path called dominance path,. This path is denoted by dom(n). Tree nodes having out degree zero are called leaves.

Use of dominator tree is that instead of covering all the nodes of the CFG, only the leaves of the dominator tree are covered. Covering each leaf of dominator tree guarantees that every dominance path is covered. Union of nodes of dominance graph gives set of nodes of the CFG. Thus the cost of testing can be reduced, by applying the above concept [20].

Representation

Program input variables are represented as chromosomes using binary vector. Length of the chromosome depends on the domain length for each input variable and precision required.

Suppose we have a program of k input variables $x_1,x_2,x_3\ldots..x_k$ where each variable x_i can take value from a domain Di= $[a_i ,b_i]$.Now for each input value, the precision of d_i decimal places is required. Each domain Di is divided into $(b_i-a_i) \times 10^{di}$ ranges of equal size, in order to achieve such precision. Let m_i be the smallest integer such that $(b_i-a_i) \times 10^{di} \leq 2^{mi} -1$. Coding of each variable x_i as a binary string$_i$ of length m_i satisfies the precision requirement.

By using the following formula, mapping is done from binary string to a real number:

$x_i = a_i + x_i' * (b_i - a_i) / 2^{mi} -1$

Here the decimal value of the binary string is represented by x_i'.

If precision not required i.e. d_i =0, then the formula would become:

$x_i = a_i + \text{int} (x_i' * (b_i - a_i) / 2^{mi} -1)$

A binary string of length$m = \sum_{i=i}^{k} \text{mi}$, is used to represent each chromosome or a test case. The first group of m_1 bits map into a value from range [a_1, b_1] of variable x_1, the next group of m_2 bits maps into a value from the range [a_2 ,b_2] of variable x_2 and so on.

Initial Population

As mentioned above a binary string of length m, is used to represent each chromosome (as a test case). *pop_size*, m bit string is randomly generated and is used to represent the initial population, where population size is denoted by *pop_size*. *pop_size* is determined experimentally. Using the above mentioned formulas, each chromosome is converted to k decimal numbers which are then used to represent values of k input variables x_1,x_2,x_3…...x_k .

Evaluation Function

For evaluating test data, new evaluation or fitness function is used. Fitness function depends on the concept of the dominance relation between nodes of CFG. This fitness function evaluates by executing the program with each test case and then finding out which nodes are covered by it. This set if traversed path is denoted by *exePath* also dominance path of the target node is found out.

Ratio of the number of nodes covered of the dominance path of the target node to the total number of nodes of the dominance path of the target node. For each chromosome, fitness value is calculated as follows:

1. Find *exePath:* the set of nodes that are covered by a test case, while traversing a program.
2. Find dom(n):dominance path of the target node 'n'.
3. Determine (dom(n) – *exePath*): nodes that are not covered in the dominance path.
4. Determine (dom(n) – *exePath*)**'**: nodes that are covered in the dominance path.
5. Calculate | (dom(n) – *exePath*)**'**| : number of nodes that are covered in the dominance path.
6. Calculate |dom(n)| : number of nodes in the dominance path of the target node n.

Then, ft (v_i) = | (dom(n) – *exePath*)'| / |dom(n)

If the fitness value ft (v_i) of a test case is 1 then it is considered as optimal.

Selection

From all the members of the current population, test cases are selected that will be parents of the new population. The method used for selecting test cases is roulette wheel which is described below:

A roulette wheel with slots sized according to fitness is used, for selecting new population with respect to the probability distribution based on fitness values. Construction of Roulette wheel is as follows:

- Fitness value ft (v_i) for each chromosome v_i (i=1, 2…….*pop_size*) is calculated
- Total fitness of the population $F = \sum_{i=i}^{pop\_size}$ ft (vi) is calculated.
- Relative fitness value rft is calculated for each chromosome, rft (v_i) / F.
- Cumulative fitness value cft is calculated for each chromosome

  If i=1 cft(v_i) =rft(v_i)

  Otherwise

  If i=2, 3……*pop_size* cft(v_i) =cft(v_{i-1}) + rft(v_i)

Spinning the roulette wheel pop_size , is made the basis of selection. A single chromosome is selected, each time for a new population in the following way:

- A random (float) number 'r' is generated from the range [0...1].
- The first chromosome v_1 selected, if r < cft(v_1), otherwise select the i^{th} chromosome v_i ($2 \leq i \leq pop\_size$) such that cft(v_i) $\leq$ r $\leq$ cft(v_{i+1}).

  Some chromosomes would be selected more than once.

Recombination

In this phase two operators are used: crossover and mutation. New individuals are created to form new population from the selected parents.

1. **Crossover:** Substring information is exchanged at a random position in the chromosome of the two parents to produce new strings. Crossover probability is the deciding factor for crossover. The expected numbers of chromosomes (PXOVER x *pop_size*) which will undergo crossover operations are decided on the basis of probability of crossover PXOVER. Followings steps are taken for it:

   For each chromosome in the parent population:

   - A random (float) number 'r' is generated from the range [0…1].
   - Given chromosome is selected for crossover, if 'r' <PXOVER.

   Mating of selected chromosomes is done randomly: Random integer *pos* from the range [1….m-1] (m is the number of bits in a chromosome), for each pair of coupled chromosome is generated. Position of the crossing point is indicated by *pos*.

2. **Mutation:** Performed on a bit by bit basis. Operates after crossover operator and flips with the predetermined probability, each bit of selected chromosome. The expected number of mutated bits (PMUTATION x m x *pop_size*), is given by the probability of mutation (PMUTATION). Every bit has an equal chance to undergo mutation. The followings steps are followed:

   For each chromosome in the current population and for each bit within the chromosome:

   - A random (float) number 'r', is generated from the range [0…1].
   - Mutate the bit, if 'r' < PMUTATION.

Until the test requirement i.e. covering the set of leaves of the denominator tree is achieved, population continues to evolve. When a set of individuals have traversed the dominance path of the test requirement and its fitness value $ft(v_i)$ becomes 1, the evolution stops.

3.3.2 Algorithm for GA [20]

Input:

The program to be tested P;

Number of program input variables;

Domain and precision of input data;

Population size;

Maximum no. of generations (Max_Gen);

Probability of crossover;

Probability of mutation;

Output:

Set of test cases for P, and the set of nodes covered by each test case;

List of uncovered nodes, it any;

Begin

Step 0: Setup (Analyze P to find prerequisites)

1. Classify the program's statements.
2. Build the program's control flow graph CFG.
3. Build the program's dominator tree DT.
4. Find the set of leaves L of the dominator tree.
5. Instrument P to obtain P'.

Step 1: Initialization

Initialize the score board to zero;

nRun ← 0;

Set of test cases for P ← φ;

nCases ← 0;

Step 2: Generate test cases

For each uncovered node and not selected before in the set of nodes to be tested (L)

Begin

nRun ← nRun +1;

Create Initial_Population;

```
Current_population ← Initial_Population;
No_Of_Generation ← 0;
For each member of current population do
Begin
    Current chromosome is converted to corresponding set of decimal values;
    Execute P' with this data set as input;
    Evaluate the current test case;
    If (the current node is covered) then
        Mark the current node as covered;
    End If
End For;
Keep the best member of the current population;
While (current node is not covered and No_Of_Generations ≤ Max_Gen) do
    Begin
        Select set of parents of new population from members of current
        population using roulette wheel method;
        Create New_Population using crossover and mutation operators;
        Current_Population ← New_Population;
        For each member of Current_Population do
        Begin
            Convert current chromosome to the corresponding set of decimal
            Values;
            Execute P' with this data set as input;
            Evaluate the current test case;
            If (the current node is covered) then
                Mark the current node as covered;
            End If
        End For;
        Increment No_Of_Generation;
    End While;
    If (the current node is covered) then
```

nCases←nCases +1;

Add these test cases to set of test cases for P;

Update the score board;

Check all uncovered nodes by this test case

End If

End For;

Step 3: Produce output

Return set of test cases for P, and set of nodes covered by each test case;

Report uncovered node, if any;

End

3.4 Proposed Framework for the Tool "Test Generator Comparator"

Here we are going to give the framework for our tool Test Generator Comparator that we have used for generating test data for some input programs and then applying the above mentioned three algorithms namely Ant Colony (ACO), Artificial Bee Colony (ABC) and Genetic Algorithm (GA) on them. These three algorithms are then compared on the basis of the results that they generate for the input programs.

The framework for the tool can be shown as:

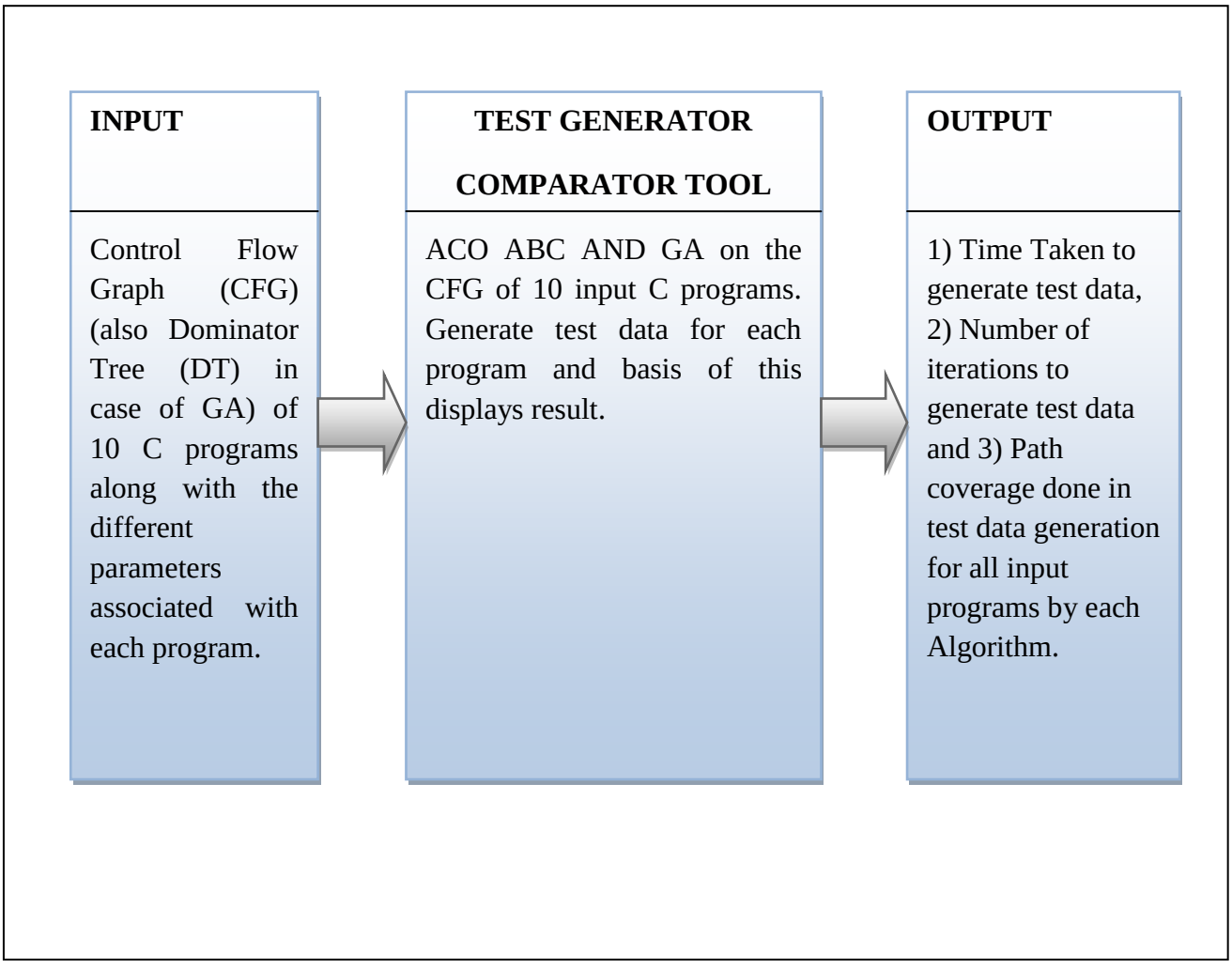


Fig 3.6 Framework for the tool "TEST GENERATOR COMPARATOR"

A theoretical comparison based on different parameters has been performed by us and it can be represented in tabular form as shown below:

Table 3.1 Theoretical Comparison of ACO, ABC and GA

Parameters	Ant Colony Algorithm	Artificial Bee Colony Algorithm	Genetic Algorithm
No Of Cycles	equal to cyclomatic complexity	equal to 2*(cyclomatic complexity)	Prescribed by tester
Type of Algorithm	Pheromone based(overhead)	Non-Pheromone based(no overhead)	Population Based
Communication about selection	Based on Pheromone	Based on waggle dance/fitness function	Status Flag
Technique for test data generation	All ants start the search simultaneously.	Only some bees dedicated for searching (Scout bees).	Crossover and mutation
Type Of Approach	Sequential	Parallel	Parallel
Memory Limitations	yes	No	yes
Computational Overhead	yes	No	yes
Time Taken	Greater than ABC	Less than both ACO and ABC	Greater than ABC
Problem Of Local Optimum	Yes	No	Yes
No Of Iterations To Converge	Greater than ABC	Less than both ACO and GA	Greater than ABC

Chapter 4: Implementation

In this chapter we have shown how the algorithms discussed in above chapter, are applied on 10 input C programs. CFG for these C programs are shown in APPENDIX.

The following are the input C programs that are used:

1) Program for finding whether a number is even or odd.
2) Program for finding whether a given year is leap year.
3) Program for finding the division of a student on the basis of marks obtained.
4) Program for finding maximum of three numbers.
5) Program for finding minimum of three numbers.
6) Program to find whether a point lies inside or outside or on circle.
7) Program to find in which quadrant a given point lies.
8) Program to find the nature of roots of a quadratic equation.
9) Program to check whether a number divides another given number or not.
10) Program to classify the type of triangle.

We have shown the CFG for the input programs and now we will show how theses 3 Algorithms are applied on these Input programs.

We have developed the code for each of the three algorithms in Java. Also the above shown CFG for the input programs are implemented in Java. For each of the program a CFG and a dominator tree (used in GA) is constructed along with this their corresponding input C programs are modified, so that for each test case generated, the path that it satisfies can be found out for a given input program.

USING ANT COLONY ALGORITHM

The Ant Colony algorithm discussed in the above chapter is used for test case generation for a given input C program. The algorithm takes input the CFG of each input program, the number of input variables associated with an input program and the cyclomatic complexity of the CFG. Now starting from the first node all the independent paths are found out based on the algorithm and then these paths are stored. These paths are equal to the cyclomatic complexity

of the CFG. Once the independents paths are found, test data is generated randomly as per the algorithm. These test data are checked as which path they satisfy. This is done by the modified program. If the test data satisfies the path then a new test case is generated and of it does not then this test data is rejected and new test data is generated. For each program, three parameters are given as output, which measures the performance of the algorithm:

1. **No of Iterations:** The number of times the ACO algorithms runs while generating the test data is measured in terms of **iterations.** The maximum limit for the iterations we have set as 51. Hence generating if the test data generated satisfies all the independent paths before reaching the maximum limit then this becomes the number of iterations and if all the paths are not covered and the loop reaches the maximum limit then 51 is the number of iterations. The less the number of iterations, more better the algorithm is.
2. **Path Coverage:** The number of paths which are covered is given by the path coverage. Its maximum value will be equal to the cyclomatic complexity. It shows the number of paths that are covered by the test cases generated by the algorithm.
3. **Time Taken:** The total time taken to run the algorithm on one program is given by time taken. It is calculated as (end_time – start_time).

USING ARTIFICIAL BEE ALGORITHM

The artificial bee algorithm discussed in the above chapter is used for test data generation for a given input C program. The algorithm takes input CFG of an input program. The algorithm loops for (2*cyclomatic complexity) times for generating all the independent paths. The paths are then divided into set based on similarity between the nodes in the path. The test data is generated and then checked with each independent path. If it satisfies the path then the new test data is generated, and if it does not satisfies the path this test data is modified in order to generate new test data. For each program three parameters are given as output:

1. **Number of Iterations:** It gives the number of times the ABC runs while generating the test data. The maximum limit for the iterations we have set as 51. Hence generating if the test data generated satisfies all the independent paths before reaching the maximum limit then this becomes the number of iterations and if all the paths are

not covered and the loop reaches the maximum limit then 51 is the number of iterations. The less the number of iterations, more better the algorithm is.

2. **Path Coverage:** The number of paths which are covered is given by the path coverage. Its maximum value will be equal to the cyclomatic complexity. It shows the number of paths that are covered by the test cases generated by the algorithm.
3. **Time Taken:** It gives the total time taken to run the algorithm on one program. It is calculated as (end_time – start_time).

USING GENETIC ALGORITHM

The genetic algorithm discussed in the above chapter is used for test data generation for a given input C program. The algorithm takes input the Dominator Tree corresponding to the CFG of the program. The population size we have taken as 20, probability of crossover as 0.8 and probability of mutation as 0.15 .Based on the algorithm discussed and the parameters taken, GA is applied on the input denominator tree and test data is generated. If the data generated satisfies any of the independent paths then it is selected, if not then mutation and crossover is applied to generated new test data. For each program three parameters are given as output:

1. **Number of Iterations:** It gives the number of times the GA runs while generating the test data. The maximum limit for the iterations we have set as 51. Hence generating if the test data generated satisfies all the independent paths before reaching the maximum limit then this becomes the number of iterations and if all the paths are not covered and the loop reaches the maximum limit then 51 is the number of iterations. The less the number of iterations, more better the algorithm is.
2. **Path Coverage:** The number of paths which are covered is given by the path coverage. Its maximum value will be equal to the cyclomatic complexity. It shows the number of paths that are covered by the test cases generated by the algorithm.
3. **Time Taken:** It gives the total time taken to run the algorithm on one program. It is calculated as (end_time – start_time).

Snapshot of the Tool "TEST GENERATOR COMPARATOR"

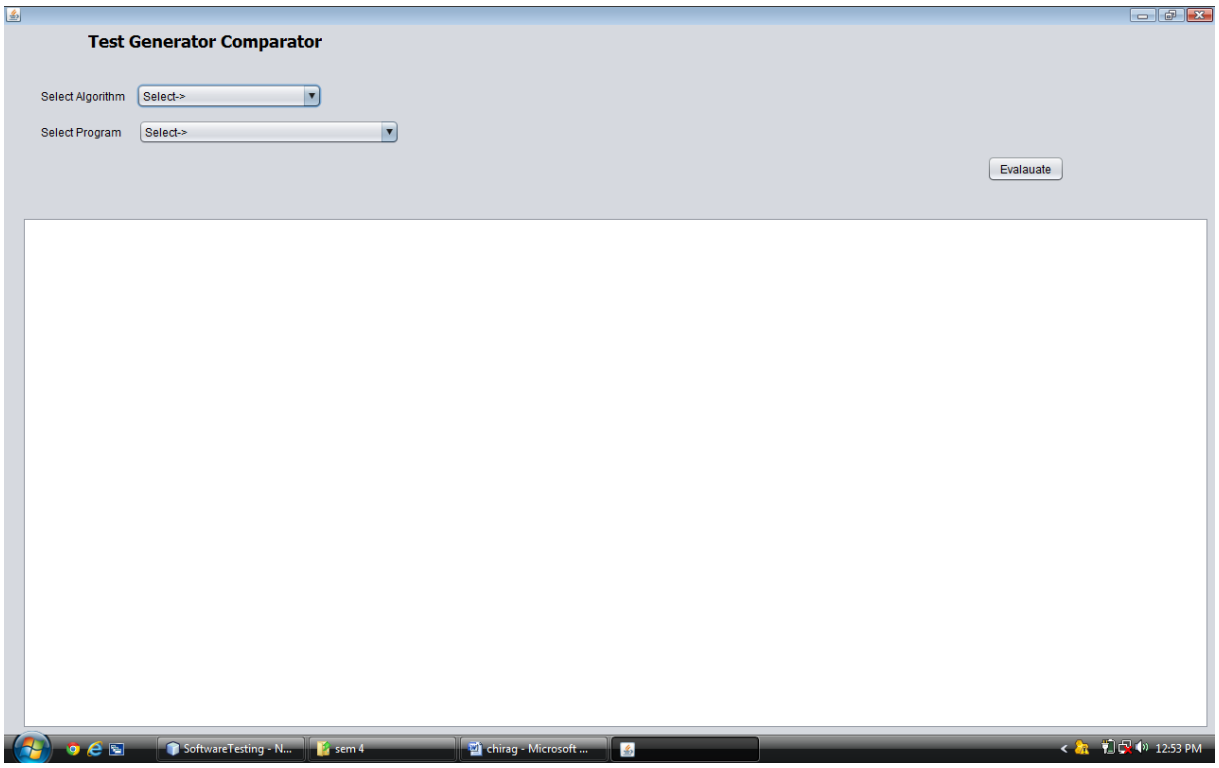


Fig 4.1 Snapshot 1 of the Tool

The above Fig 4.11 shows the snapshot for the tool. We can select the algorithm which we want to apply from the "Select Algorithm" drop down menu and then we can select the program on which the algorithm should be applied from "Select Program" drop down menu. After selecting the algorithm and program, we click on the evaluate button and then in the space below the output is shown which consists of :-

1. No of iterations
2. Path coverage
3. Time taken

The below snapshot in Fig 4.12 shows it:

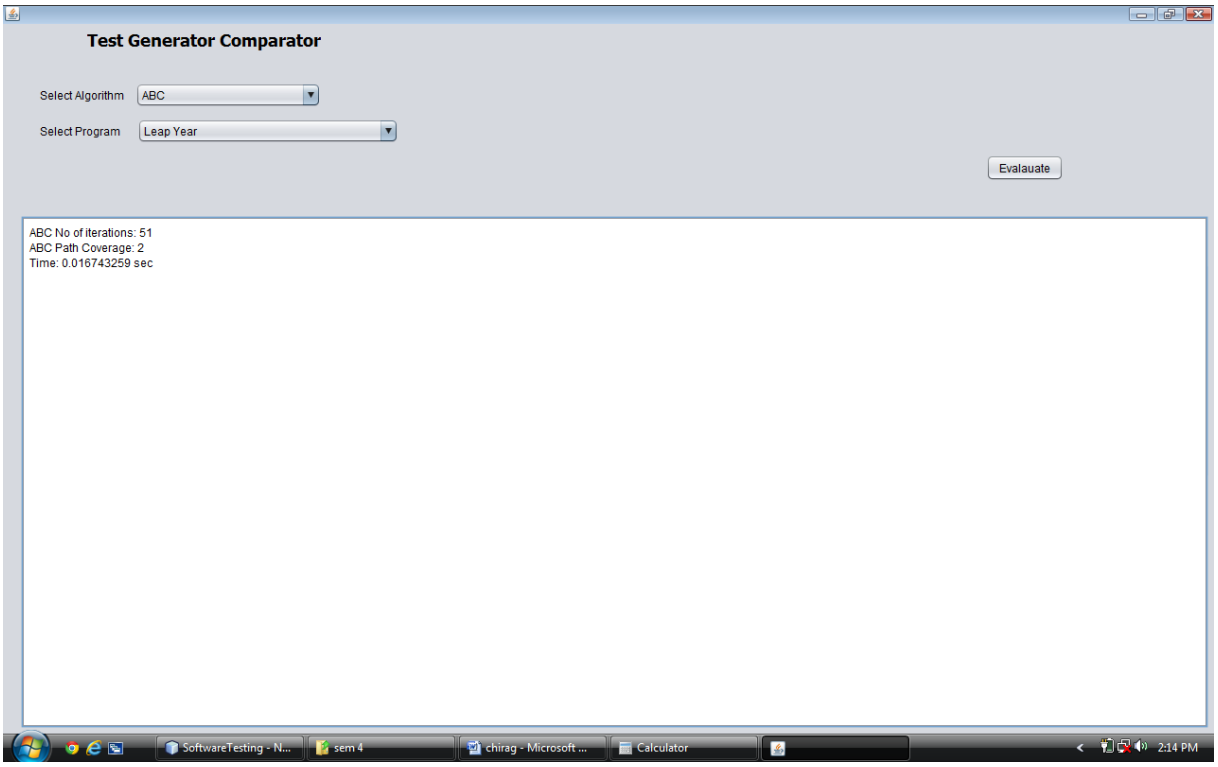


Fig 4.2 Snapshot 2 of the Tool

Hence we evaluate each algorithm and then the output shown by them can be compared.

Chapter 5: Results

Here we have shown the output for each algorithm, when applied on each of the input C program. Each value is obtained by running the algorithms on the input programs for 6 times and then the average of the value is taken as shown.

The output of each algorithm can be shown in the form of table as:

Table 5.1 Output of TEST GENERATOR COMPARATOR for input programs

S.No	Input Program	Inputs	Cyclomaic Complexity	Time by ACO (avg)	Time by ABC (avg)	Time by GA (avg)	Iterations by ACO (avg)	Iterations by ABC (avg)	Iterations by GA (avg)	Path Coverage by ACO (avg)	Path Coverage by ABC (avg)	Path Coverage by GA (avg)
1	Even Odd	1	3	.127	.005	.094	16	13	30	3	3	3
2	Leap Year	1	4	.056	.010	.048	51	51	51	2	2	3
3	Marks	3	5	.014	.012	.012	51	29	51	2	4	2
4	Maximum of three	3	3	.012	.009	.055	16	2	5	3	3	3
5	Min of two	2	2	.011	.007	.009	3	2	3	2	2	2
6	Point Circle	3	3	.010	.013	.009	7	3	3	3	3	3
7	Quadrant	2	4	.009	.006	.011	51	51	51	1	1	1
8	Quadratic Equation	3	4	.009	.015	.016	51	51	51	2	1	3
9	Remainder	2	2	.050	.012	.010	20	18	18	2	2	2
10	Triangle Classifier	3	8	.068	.050	.051	51	47	51	5	6	5

5.1 COMPARISON ON THE BASIS OF TIME TAKEN

Table 5.2 Time Taken by ABC ACO and GA for Input Programs

Programs	Time Taken_GA	Time Taken_ABC	Time Taken_ACO
EO	0.094	0.005	0.127
LEAP	0.048	0.01	0.056
MARKS	0.012	0.012	0.014
MAX	0.055	0.009	0.012
MIN	0.009	0.007	0.011
POINT	0.009	0.013	0.01
QUADRANT	0.011	0.006	0.009
QUADEQ	0.016	0.015	0.009
REMAINDER	0.01	0.012	0.05
TRIANGLE	0.051	0.05	0.068

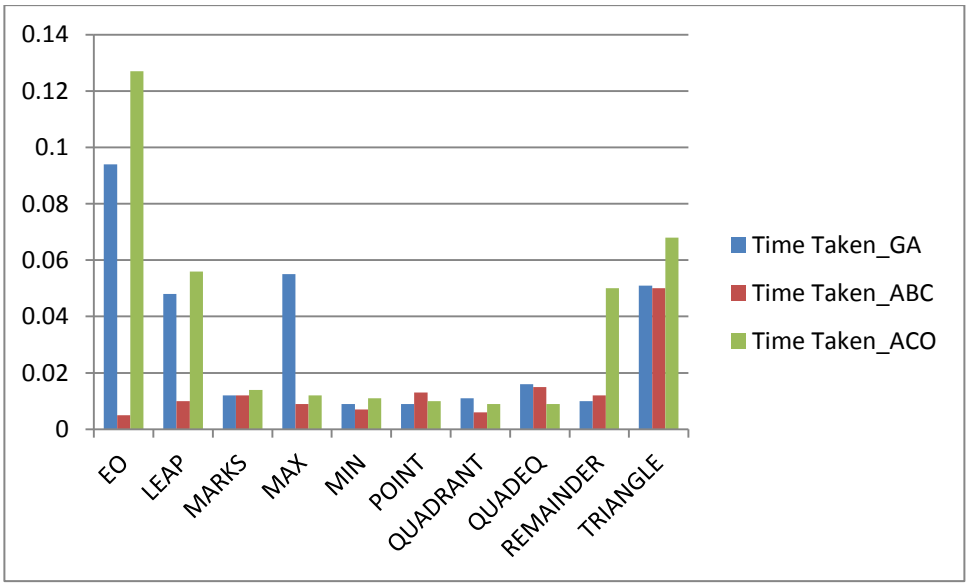


Fig 5.1 Time Taken by ABC ACO and GA for Input Programs

From the table 5.2 and Fig 5.1 we can easily see that the time taken by ABC is less as compared to GA and time taken by GA is less when compared to ACO and hence we can say that the time taken by ABC is smaller as compared to GA and ACO.

5.2 COMPARISON ON THE BASIS OF ITERATIONS

Table 5.3 Iteration done by ABC ACO and GA for Input Programs

Programs	Iterations_GA	Iterations_ABC	Iterations_ACO
EO	30	13	16
LEAP	51	51	51
MARKS	51	29	51
MAX	5	2	16
MIN	3	2	3
POINT	3	3	7
QUADRANT	51	51	51
QUADEQ	51	51	51
REMAINDER	18	18	20
TRIANGLE	51	47	51

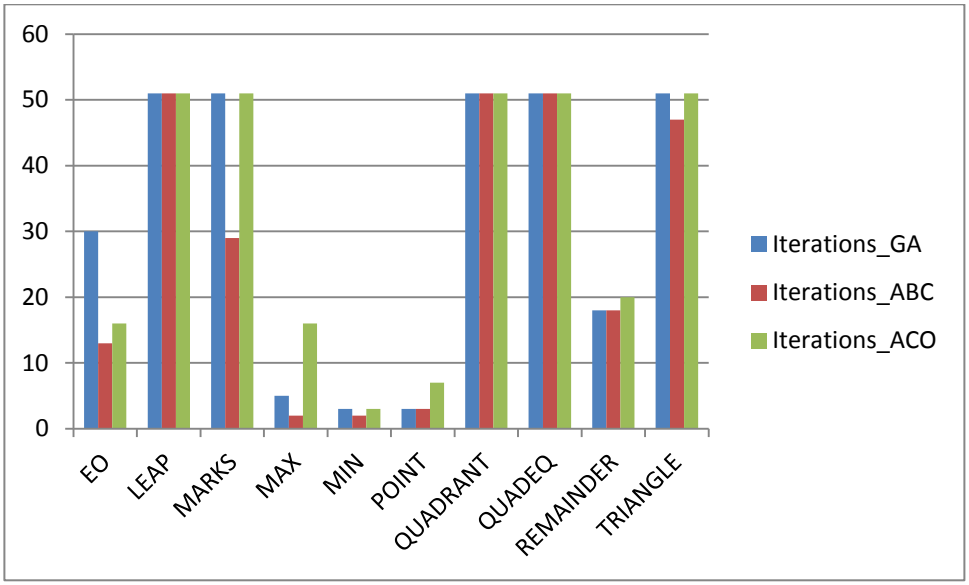


Fig 5.2 Iterations done by ABC ACO and GA for Input Programs

From the Table 5.3 and Fig 5.2 we can deduce that the number of iterations, for all the algorithms is almost same for some of the input programs. But for some of the input programs we can see that the iterations are less for ABC when compared to other algorithms.

This shows that, the iterations taken by ABC to generate test data is less than other two algorithms.

5.3 COMPARISON ON THE BASIS OF PATH COVERAGE

Table 5.4 Path Coverage by ABC ACO and GA for Input Programs

Programs	Path Coverage_GA	Path Coverage_ABC	Path Coverage_ACO
EO	3	3	3
LEAP	3	2	2
MARKS	2	4	2
MAX	3	3	3
MIN	2	2	2
POINT	3	3	3
QUADRANT	1	1	1
QUADEQ	3	1	2
REMAINDER	2	2	2
TRIANGLE	5	6	5

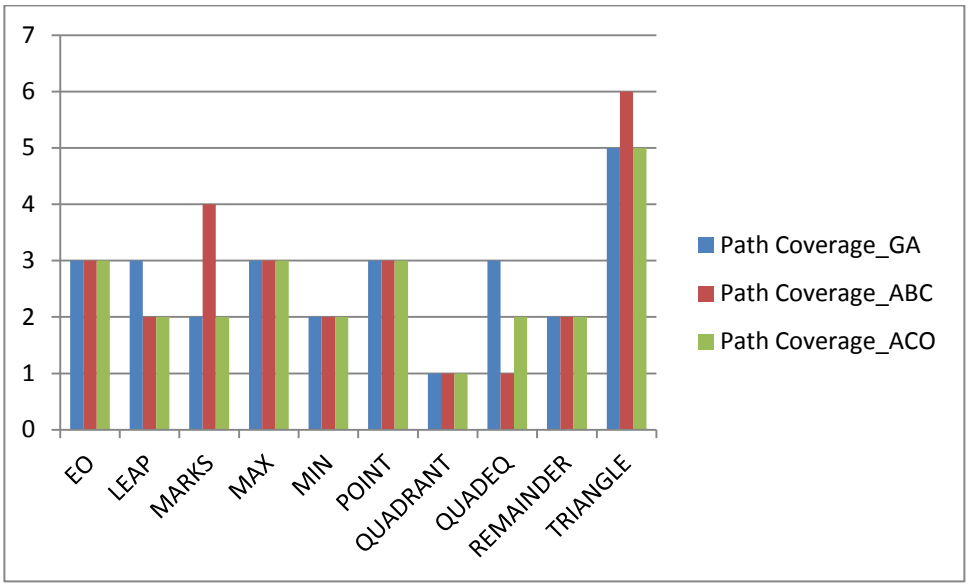


Fig 5.3 Path Coverage by ABC ACO and GA for Input Programs

From Table 5.4 and Fig 5.3 we can see that the path coverage is same for all the three algorithms but in some cases ABC shows better output then the other two algorithms.

Hence the test data generated by ABC covers more paths as compared to other algorithms

Chapter 6: Conclusions

We successfully derived in chapter 1, the concept of software testing, the need for testing and who should do the testing. Along with this we also showed the test data generation, the motivation behind the work that we have done and the significance and importance of this.

In chapter 2, we found out different types and forms of software testing. Also the previous and current work done in the field of software testing by different authors provided us, with different methods and algorithms for machine learning used in test data generation.

In chapter 3, we successfully showed the three algorithms used by us for test data generation. The algorithm along with their background is shown. We also showed the framework for our tool "TEST GENERATOR COMPARAR" developed by us for comparison of three algorithms and showed theoretical comparison that we have performed successfully in tabular form.

In chapter 4, we successfully implemented the three algorithms by showing how they are applied on 10 input C programs. Also the snapshot of the tool developed by us for comparing the three comparisons was displayed successfully.

In chapter 5, we have shown the results that we have obtained by applying the algorithms on the input C programs. The algorithms are compared on the basis of three parameters number of iterations, path coverage and time taken. Results are shown in the form of table and graph and we successfully deduced from the graphs that ABC algorithm gives better performance when compared to the other two algorithms.

In chapter 6 we have concluded the work done in our thesis and showed that we have achieved the aim successfully.

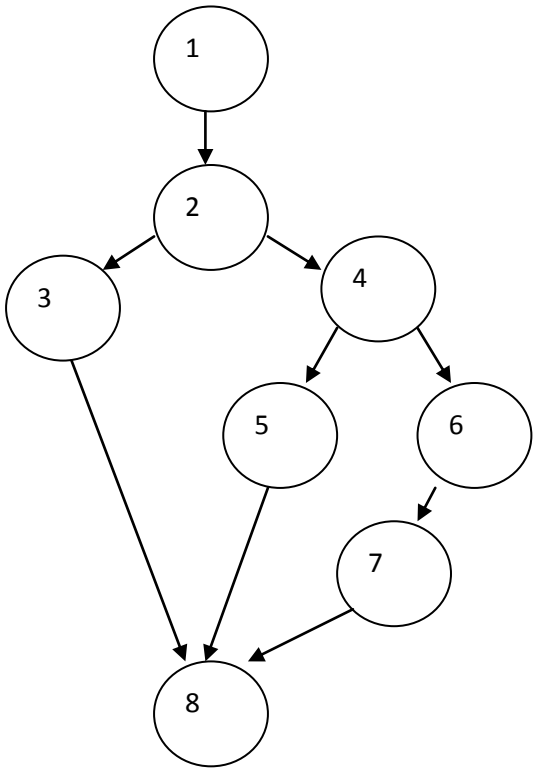
References

1. Aggarwal, K.K, Singh, Y., "Software Engineering", New Age International Publishers, Second ed., 2006.
2. Yogesh Singh,"Software Testing",Cambridge University Press,First,2012.
3. Praveen Ranjan Srivastava, Km Baby, G Raghurama,"An Approach Of Optimal Path Generation Using Ant Colony Optimization", IEEE, 2006.
4. Kewen Li, Zilu Zhang Weying Liu ,"Automatic Test Data Generation Based On Ant Colony Optimization", IEEE, 2009.
5. Huaizhong LI, C. Peng LAM, "Software Test Data Generation using Ant Colony Optimization", World Academy of Science, Engineering and Technology, 2005.
6. Ahmed S. Guiduk ,"A New Software Data-Flow Testing Approach via Ant Colony Algorithms ",Universal Journal of Computer Science and Engineering Technology, 2010.
7. Chengying Mao, Xinxin Yu,, Jifu Chen,"Generating Test Data for Structural Testing Based on Ant Colony Optimization", IEEE, 2012.
8. Soma Shekhar Babu Lama, M L Hari Prasad Rajub, Uday Kiran Mb, Swaraj Chb, Praveen Ranjan Shrivastava, "Automated Generation Of Independent Paths and Test Suite Optimization Using Artificial Bee Colony", Elsevier, 2011.
9. AdiSrikanth, Nandakishore J. Kulkarni, K. Venkat Naveen,PuneetSingh and Praveen Ranjan Shrivastava, "Test Case Optimization Using Artificial Bee Colony Algorithm", Springer, 2011.
10. Surender Singh Dahiya,Jitendar Kumar Chhabra,Shakti Kumar,"Application of Artificial Bee Colony Algorithm to Software Testing",IEEE, 2010.
11. D. Karaboga, B.Basturk,"On the performance of artificial bee colony (ABC) algorithm", Elsevier, 2007.
12. Adil Baykasolu, Lale Özbakır and Pınar Tapkan,"Artificial Bee Colony Algorithm and its Application to Generalized Assignment Problem", Itech Education and Publishing, 2007.
13. D. Jeya Mala and V. Mohan," ABC Tester - Artificial Bee Colony Based Sofware Test Suite Optimization Approach", IJSE, 2009.

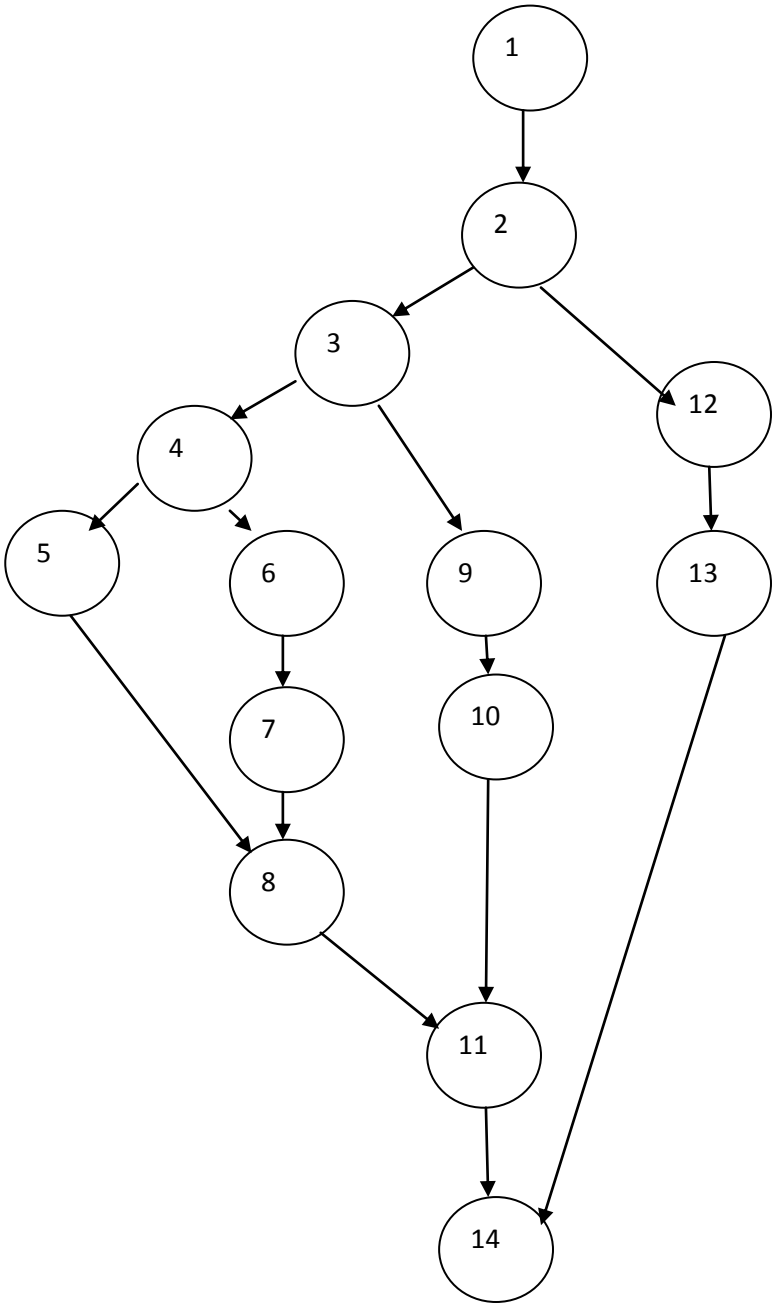
14. Dr. Arvinder Kaur,Shivangi Goyal,"A Bee Colony Optimization Algorithm For Code Coverage Test Suite Prioritization", IJEST, 2011.
15. Roy P. Pargas, Mary Jean Harrold, Robert R. Peck, "Test-Data Generation Using Genetic Algorithms",Journal of Software Testing, Verification and Reliability, 1999
16. Li Bin, Li Zhi-Shu, Chen Yan-Hong, Li Bao-Lin, "Automatic Test Data Generation Tool Based on Genetic Stimulated Annealing Algorithm", International Conference on Computational Intelligence and Security Workshops, 2007
17. Praveen Ranjan Srivastava, Tai-hoon Kim, "Application of Genetic Algorithm in Software Testing", International Journal Of Software Engineering and its Applications Vol. 3, No.4, 2009.
18. Jin-Cherng Lin, Pu-Lin Yeh, "Automatic test data generation for path testing using GAs", Elsevier, 2001.
19. Moheb R. Girgis, "Automatic Test Data Generation for Data Flow Testing Using Genetic Algorithm", Journal of Universal Computer Science, vol. 11,no.6, 2005.
20. Ahmed S. Ghiduk, Moheb R. Girgis, "Using Genetic Algorithms and Dominance Concepts for Generating Reduced Test Data",Informatica 34, 2010.
21. Peng NIE, "A PSO Test Case Generation Algorithm with Enhanced Exploration Ability",Journal of Computational Information Systems,2012.
22. Shivangi Goyal,"The Application Survey:Bee Colony"IRACST-Engineering Science and Technology Journal(ESTIJ), Vol.2, No.2, 2012

APPENDIX:

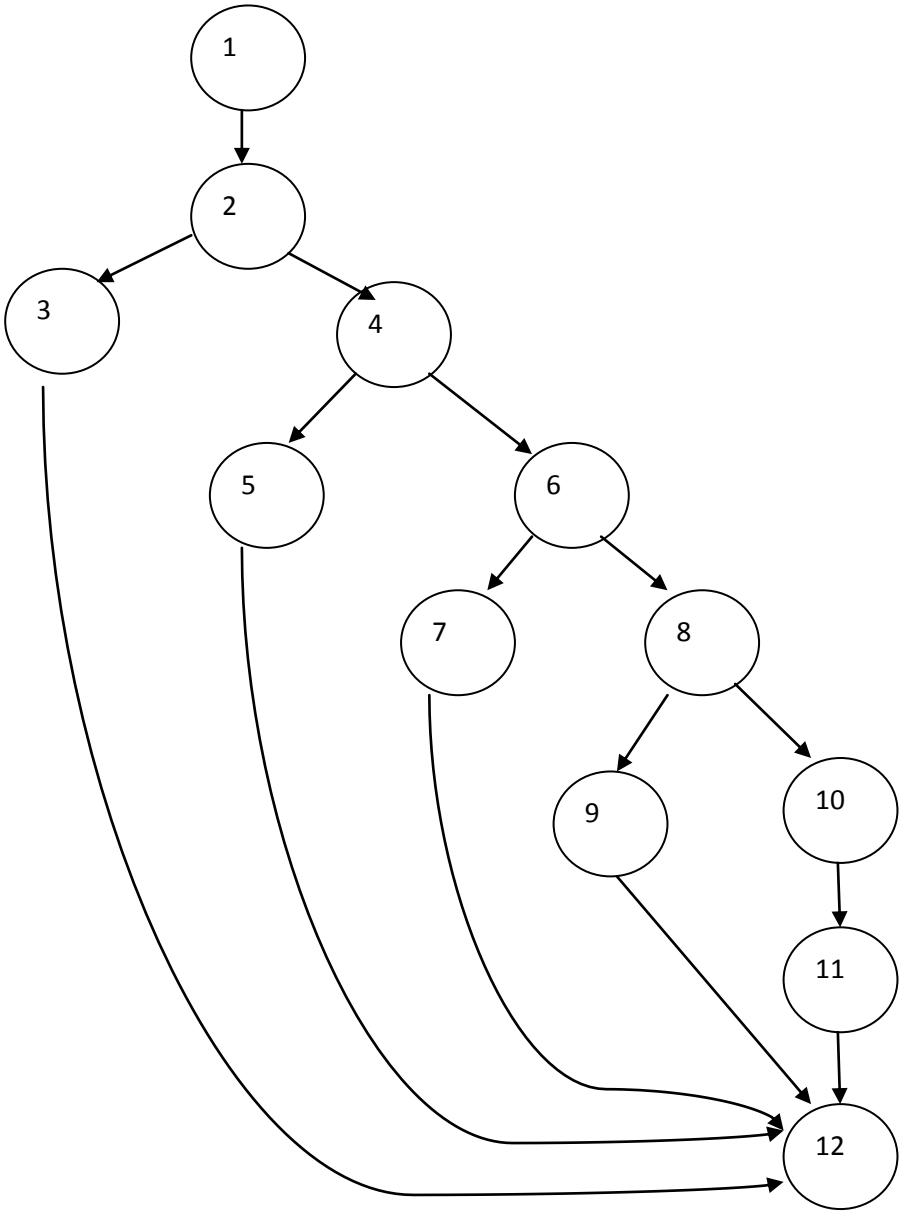
1. CFG of programs for finding whether a number is even or odd.



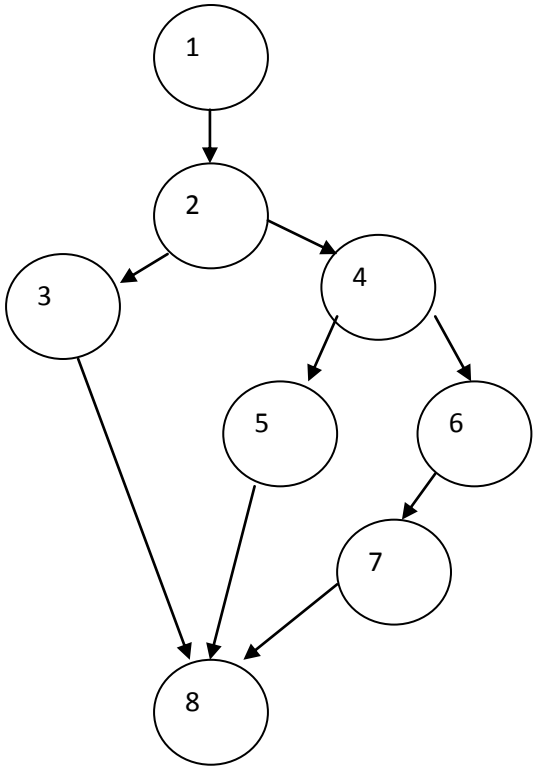
2. CFG of program for finding whether a given year is leap year or not.



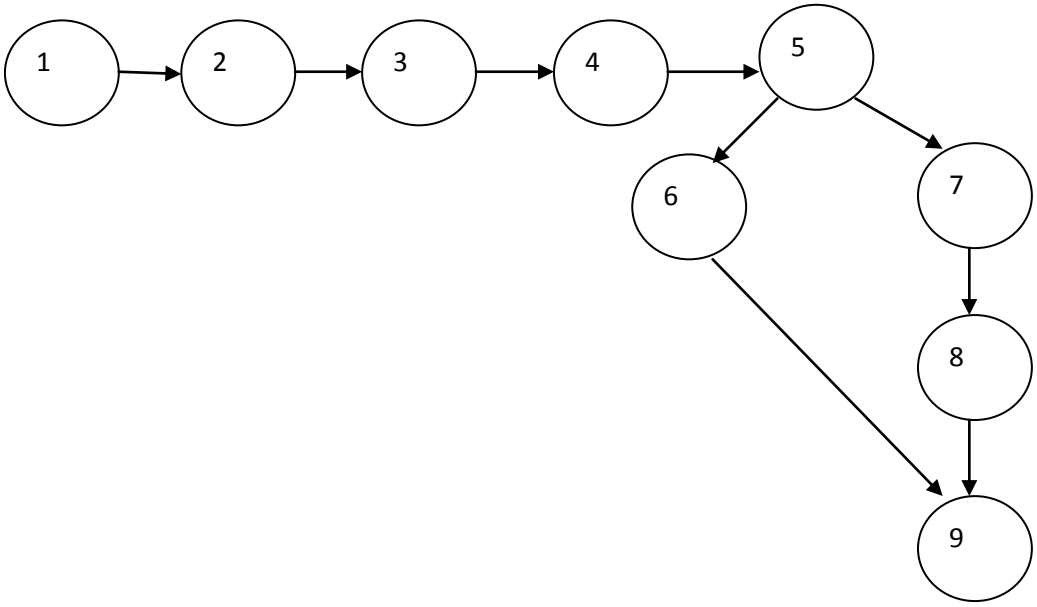
3. CFG of program for finding the division of a student on the basis of marks obtained.



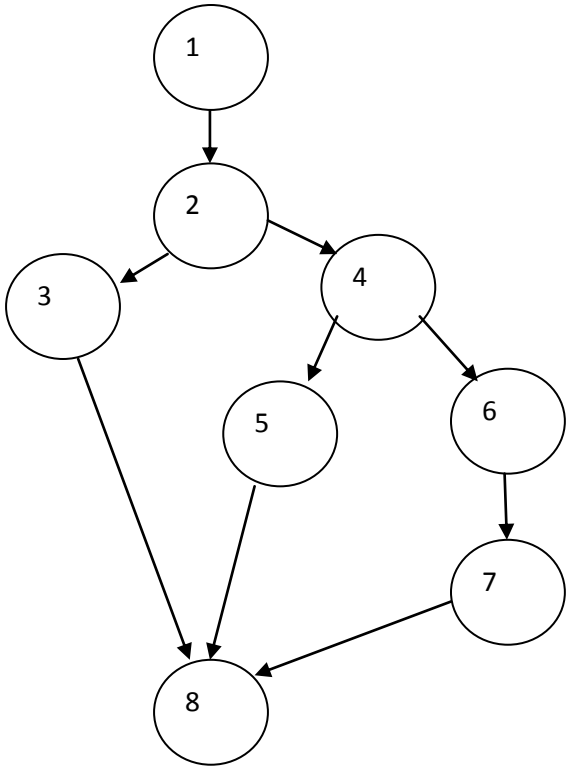
4. CFG of program for finding maximum of three numbers.



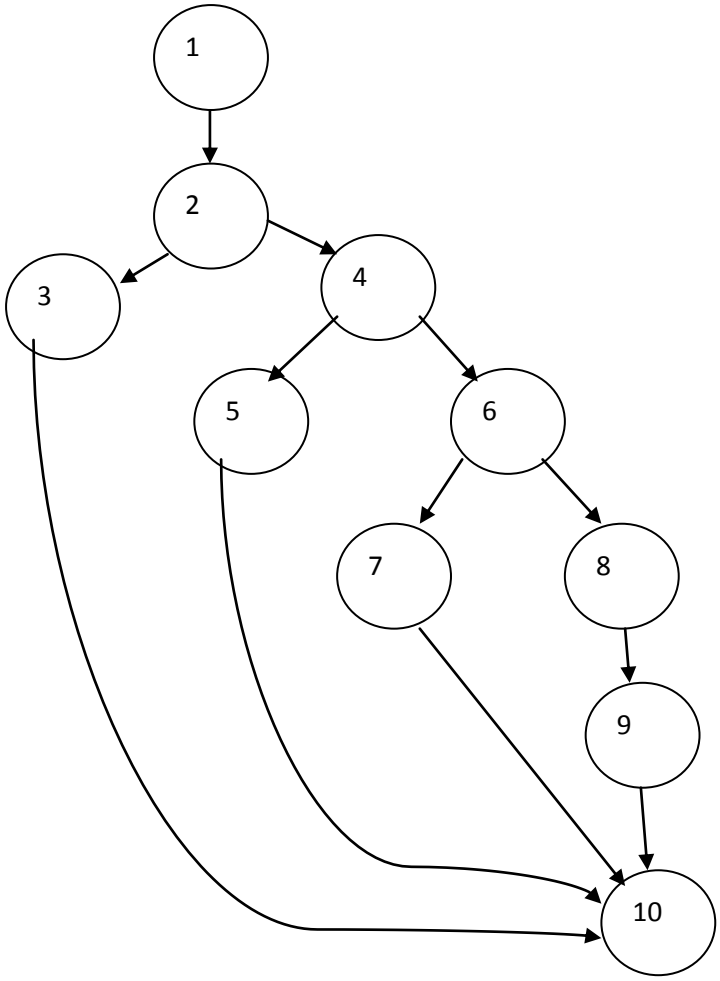
5. CFG of program for finding minimum of three numbers.



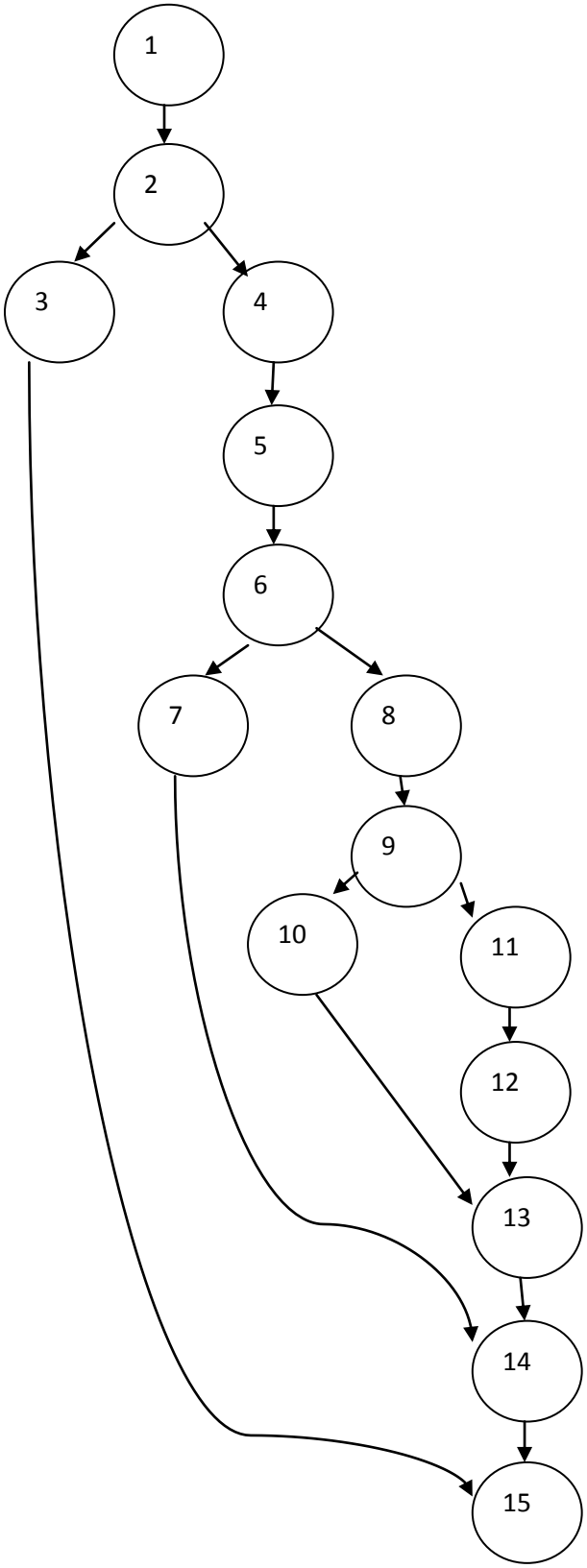
6. CFG of program to find whether a point lies inside or outside or on circle.



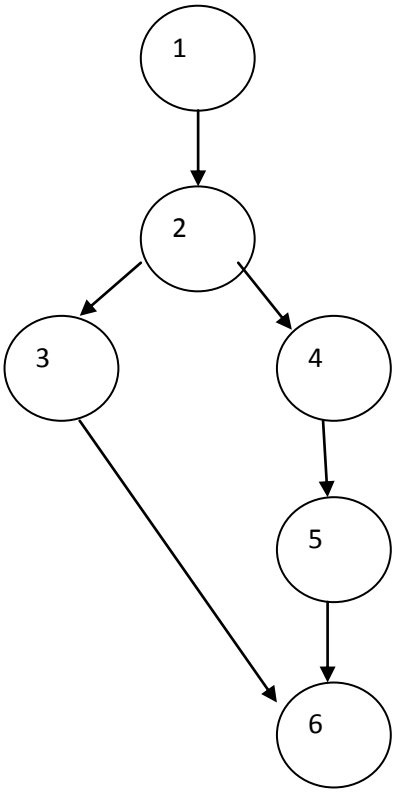
7. CFG of program to find in which quadrant a given point lies.



8. CFG of program to find the nature of roots of a quadratic equation.



9. CFG of program to check whether a number divides another given number or not.



10. CFG of program to classify the type of triangle.