ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

IJESRT
INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

SWARM INTELLIGENCE TECHNIQUE AND ARTIFICIAL INTELLIGENCE TECHNIQUE IN SOFTWARE TESTING

(INTEGER TYPE TEST DATA GENERATION)

Dipen Saini, Dr.Parminder Kaur

Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar, India

Abstract

In this research work, we have used genetic algorithm an artificial intelligence technique for automatic generation of test data for integer in case of path testing and for automatic generation of test cases. In case of small problems paths can be easily seen and we can manually select the target path, but in case of large complex problems where control flow graph of a problem is very large like control flow graph of www.irctc.com will be very large and it is very difficult to generate all the paths manually , as there are many paths in the graph we might miss some path, so to generate all the paths automatically, we have used one swarm intelligence technique called intelligent water drop algorithm, which is a graph based algorithm, which select the paths on the basis of probability and generate all the paths in the graph. With an assumption that, most feasible paths are first generated as compare to infeasible paths if present.

Keywords—Software testing, test data, swarm intelligence, IWD algorithm, Artificial intelligence, Genetic algorithm

1.) Introduction

1.1) Overview

Now days, human life is getting very busy and technology is getting advanced day by day. We all want an easy way to do work. For this cause many companies have worked on the development of software's and are also working to develop new software's for people and also for themselves, some software's have been made for the purpose of providing service to the human beings like "Adobe Photoshop, various types of music players etc", this type of software's are called *product* because they are generic. These types of software's are not made for some specific company. There are many websites which are also known as product type software's like irctc.com,passportseva.com.Some software's are made for some specific companies, these types of software's are known as project because they are made for some specific company for example "IBM Rational Robot", it is a testing tool made by IBM for their own testing purpose.

Software development is not an easy process as it seems to be software development doesn't only mean to write a code of programs, it is beyond this, software development is not just writing programs and merging them. Software development is a huge process taking long time and costs very much. There are many phases in the development of software but one phase which is very important from the point of cost and time is testing.

Software testing is the process of finding errors in the software and this phase's cost is nearly 50 to 60% of the total cost of the project. There are various types of testing techniques and strategies present to test software. Testing can be done manually or automatically, in manual testing human do testing themselves, whereas in automatic testing various tools are present like" HP Load runner "etc. The main thing in software testing is test case writing, it can be done in both ways i.e. manual or automatically. Test cases are written by inputting the data in the program and seeing the result. If the result is within limit, it is called positive testing and if result is not within the limit it is called negative testing. The main thing in test case writing is the data which we need to test the program and examine the result.

Data which is used to test a software or program is known as ***test data***. We can have test data in excel sheet which can be entered manually while executing test cases or it can be read automatically from files (XML,

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

Flat Files, Database etc.) by automation tools.
In order to test software, first thing is to generate ***Test Data*** and some test data are better at finding errors than others. Therefore, a systematic testing system has to differentiate good (suitable) test data from bad test (unsuitable) data, and so it should be able to detect good test data if they are generated. Nowadays testing tools can automatically generate test data that will satisfy certain criteria, such as branch testing, path testing, etc. However, these tools have problems, when complicated software is tested. A testing tool should be general, robust and generate the right test data corresponding to the testing criteria for use in the real world of software testing. Therefore, a search algorithm of a tool must decide where the best values (test data) lie and concentrate its search there. It can be difficult to find correct test data because conditions or predicates in the software restrict the input domain that is a set of valid data.
Test data that are good for one program are not necessarily appropriate for another program even if they have the same functionality. Therefore, an adaptive testing tool for the software under test is necessary. Adaptive means that it monitors the effectiveness of the test data to the environment in order to produce new solutions with the attempt to maximize the test effectiveness.
There are number of test-data generation techniques that have been automated earlier.
Generally the test data generators are broadly classified as follows:
· *Random test-data generators* select random inputs for the test data from some distribution.
· *Structured or path oriented test data generators* typically use the program's control-flow graph, select a particular path, and use a technique such as symbolic evaluation to generate test data for that path.
· *Goal-oriented test data generators* select input to execute the selected goal, such as a statement, irrespective of the path taken. Intelligent test data generators often rely on sophisticated analysis of the code, to guide the search for new test data.

1.2) Objective of Research

The main ***objective*** of this research work is to generate all the paths of the graph automatically, because in case of small programs it is very easy to find the paths but in case of long big programs, we might miss some path when selecting path manually. To generate paths we have used a swarm intelligence technique IWD algorithm, this algorithm is based on the probability function, this probability function decides which path will be selected first. Suppose from root node there are 2 paths then on the basis of probability next path will be selected first, this means IWD algorithm first of all selects the feasible paths and then other paths are selected and in the end all paths are selected.
After path is selected, then corresponding to that path a fitness function is generated, which will be used to generate test data such that path can be traversed. Artificial intelligence technique genetic algorithm is used because it is more efficient then random test data generator, because in random test data generator we have no search space, but in genetic algorithm we have search space so test data will be more efficient.

2.) Related work

The research paper from which I have got an idea to work on and, many research papers i have read to complete my thesis work, some of them are as follows:
(Korel, 1990) have worked on the automatic test data generation. In his work, he has used the concept of branch function, that how predicate branch can be converted into branch function which is used in the path testing.
(Michael et al., 1997) have applied genetic algorithm for dynamic test data generation and the results are pretty impressive over random or exhaustive test data generation, but this model was not applicable for boolean and string type of variables.
(Lin et al., 2001) have done on work on path testing, automated test data generation using GA, but in this path selection is manual and not applicable for long programs.
(McMinn, 2004) have provided the technique for full path coverage but failed to provide solution for optimal test data generation.
(Li et al., 2005) have worked on automatic generation of test data but have no work on the path coverage.
(Shah, 2007) have developed the IWD algorithm, swarm based which works on the graph and can be used to find all paths in the graph automatically.
(Srivastava et al., 2008) have also worked on the automatic test data generation but no sufficient work on path coverage.
(Shah, 2008) have improved the IWD algorithm, swarm based which works on the graph and can be used to find all paths in the graph automatically.
(Srivastava et al., 2009) have also worked on the automatic test data generation using ga and have also worked on path coverage.
(Srivastava et al., 2010) have worked on test data using genetic algorithm and hamming distance concept but optimal test data are uncertain, and stuck in the local optima and explore more repetitive paths.
(Sidhu et al., 2011) have explained the various areas where IWD algorithm can be used, they explain the application areas of IWD algorithm.
(Srivastava et al., 2012) have worked on automatic test data generation using IWD,but the result showed by him is not correct .

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

(Srivastava et al., 2012) have worked on integer type of data and said IWD can be used to generate string type of data and genetic algorithm cannot be used to generate test data string type.

(Aggarwal, et al., 2012) worked on the code coverage using IWD algorithm and in this fitness function can further be improved to improve the results.

(Li et al., 2013) proposed a technique using IWD algorithm and Ant colony optimization but this very difficult to implement.

3.) Software testing

3.1) Basic concept

According to different practitioners and researchers, software testing has been defined as given below:

Testing is the process of executing a program with the intent of finding errors. [15]

A successful test is one that uncovers an as-yet-undiscovered error. [15]

Testing can show the presence of bugs but never their absence. [2]

Software testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which it is intended to operate.

From the above definitions, this can be concluded that the software testing is done to enhance the quality of the software under test. The software must perform as per the requirements; in addition to that, it should be free from bugs. Thus software testing can be defined as,

Software testing is a process that detects important bugs with the objective of having better quality. [10]

3.2) White box testing

White Box Testing

White box testing also known as glass box testing .It is a test case design method that uses the control structure of the procedural design to derive test cases. Using white box testing methods, the software engineer can derive test cases that

1. Guarantee that all independent paths within a module have been exercised at least once.
2. Exercise all logical decisions on their true and false sides.
3. Execute all loops at their boundaries and within their operational bounds.
4. Exercise internal data structures to assure their validity.

Thus white box testing is the testing of the underlying implementation of a piece of software (e.g., source code) without regard to the specification (external description) for that piece of software. The goal of white box testing of source code is to identify such items as (unintentional) infinite loops, paths through the code which should be allowed, but which cannot be executed and dead (unreachable) code. [4]

There are several white box (structural) testing criteria:

• ***Statement Testing***: Every statement in the software under test has to be executed at least once during testing. A more extensive and stronger strategy is branch testing.

• ***Branch testing***: Branch coverage is a stronger criterion than statement coverage. It requires every possible outcome of all decisions to be exercised at least once i.e. each possible transfer of control in the program be exercised. This means that all control transfers are executed. It includes statement coverage since every statement is executed if every branch in a program is exercised once. However, some errors can only be detected if the statements and branches are executed in a certain order, which leads to path testing.

• ***Path testing***: In path testing every possible path in the software under test is executed, this increases the probability of error detection and is a stronger method than both statement and branch testing. A path through software can be described as the conjunction of predicates in relation to the software's input variables. However, path testing is generally considered impractical because a program with loop statements can have an infinite number of paths. A path is said to be 'feasible', when there exists an input for which the path is traversed during program execution, otherwise the path is unfeasible.

- ***Steps for basis path testing:***

Following are the steps that should be followed for designing test cases using basis path testing:

ˆ Draw the CFG using the code for which test cases have to be generated.

ˆ Determine the cyclomatic complexity of the graph.

ˆ Cyclomatic complexity provides the number of independent paths. Find a basis set of independent paths through the program control structure.

The basis set is the base for generating the test cases. Based on every independent path, choose the data such that this path is executed.

- ***Control flow graph***

The control flow graph (CFG) is a graphical representation of the control structure of a program. These can be prepared as a directed graph. It consists of a set of vertices V and a set of edges E that are ordered pairs of elements of V.

Following notations are used for a control flow graph:

Node: It represents one or more procedural statements. The nodes are denoted by circles and are either numbered or labeled.

Edges or links: An edge is represented by an arrow, and it must terminate at a node. It represents the flow of control in a program.

Decision node: A node with more than one arrow

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

leaving from it is called a decision node.
Junction node: A node with more than one arrow entering into it is called a junction node.
Region: The area bounded by some edges and nodes is called region.
As a control flow graph is drawn on the basis of the control structure of a program, Figure- 1 shows some of the fundamental graphical notations for basic programming constructs

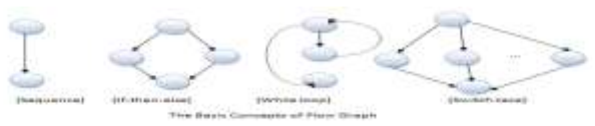


Figure 1 Notation in CFG

- *Cyclomatic complexity*

McCabe introduced the concept of measuring the logical complexity of a program by considering its control flow graph. The cyclomatic complexity also known as structural complexity calculates the number of independent paths through a program. It provides the upper bound of the number of test cases that must be designed, in order to ensure that all statements have been executed at least once and all conditions have been tested. McCabe's cyclomatic metric is very useful in finding the total number of independent paths present in any program.
Cyclomatic complexity can be calculated as shown below:
$V(G) = e - n + 2p$ eq (3.1)
Where,
$V(G)$ is the cyclomatic complexity; p is the number of graphs,
e is the number of edges in the whole graph, and n is the number of nodes in the whole graph.

4.) Swarm intelligence

4.1 Swarm intelligence

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence.
SI systems consist typically of a population of simple agents or boids interacting locally with one another and with their environment. The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. Examples in natural systems of SI include ant colonies, bird flocking, animal herding, bacterial growth, and fish schooling. The definition of swarm intelligence is still not quite clear. In principle, it should be a multi-agent system that has self-organized behavior that shows some intelligent behavior. [7]

4.2 Intelligent water drop algorithm

4.2.1 General introduction

IWD algorithm is a new swarm-based optimization algorithm inspired from natural rivers. In a natural river, water drops move towards center of the earth, due to some gravitational force acting on it. Due to this the water drop follows the straight and the shortest path to its destination. Pictorial representation of basic IWD is shown in Fig 4.1. In ideal conditions it is observed that the optimal path will be obtained. Water drop flowing in the river has some velocity which is affected by another actor, i.e., soil. [25]
Some changes that occurred while transition of water drop from one point to another point are:

1.) Velocity of water drop is increased.
2.) Soil content in the water drop is also increased.
3.) Amount of soil in the riverbed from source to destination gets decreased.

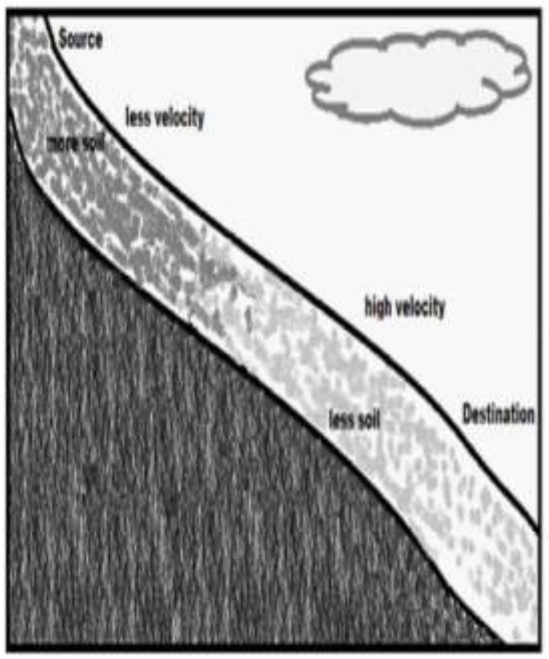


Figure 2 Concept of IWD

Water drop in the river picks up some soil in it when its velocity gets high and it releases the soil content when its velocity is less. Some of the prominent properties of the natural water drop are taken, based on which IWD is suggested. IWD has the two following important properties: [25]

1.) The amount of soil the water drop carries, which is represented by Soil (IWD) (or soil IWD).
2.) The velocity of water drop with which it is moving now, denoted by Velocity (IWD)or(vel IWD).
Value of both the properties may change during the transition. Environment contains lots of paths from source to destination which may be known or unknown .When the destination is known, IWD follows the best path to reach the destination (best is in terms of cost and any other desired measure .)When destination is unknown it finds the optimal destination. From the current location to the next location Velocity (IWD) is increased by an amount, which is nonlinearly proportional to the inverse of the amount of soil between the two locations, referred to as the change in velocity. The Soil (IWD) is also increased by extracting some soil of the path between two locations. The amount of soil added to the IWD is inversely (and nonlinearly) proportional to the time needed for the IWD to pass from its current location to next location. IWD chooses the path with less soil content. In the proposed approach, IWD is applied over the Control Flow Graph (CFG) to obtain the number of paths available in the program .The CFG depicts the logical control flow of the program. All linearly independent paths could be obtained by CFG .Independent path is the path in the program that determines at least one new set of processing statement. In other words it introduces at least one new edge in the graph. [25]

4.2.2 Practical working of IWD

The IWD algorithm as specified by Shah-Hosseini H. is as follows :[18]
1.) Initialization of static parameters.
2.) Initialization of dynamic parameters.
3.) Spread the IWDs randomly on the nodes of the graph.
4.) Update the visited node list of each IWD.
5.) Repeat Steps a to d for those IWDs with partial solutions.
a.) For the IWD residing in node i, choose the next node j, which does not violate any constraints of the problem and is not in the visited node list of the IWD.
b.) For each IWD moving from node i to node j, Update its velocity.
c.) Compute the soil.
d.) Update the soil.
6.) Find the iteration-best solution from all the solutions found by the IWDs.
7.) Update the soils on the paths that form the current iteration best solution.
8.) Update the total best solution by the current iteration best solution.
9.) Increment the iteration number
10.) Stops with the total best solution.

5. Artificial intelligence

Artificial Intelligence (AI) or Soft computing techniques are the science, and engineering of making intelligent machines, especially intelligent computer programs. These techniques have the ability of computer, software and firmware to do those things that we, as humans, recognize as intelligent behavior. A complete description of the AI technique genetic algorithm deployed for generating optimal test data and test case generation is presented in the following sections. [16]

5.1) Introduction

Genetic algorithms searching mechanism starts with a set of solutions called a population. One solution in the population is called a chromosome. The search is guided by a survival of the fittest principle. The search proceeds for a number of generations, for each generation the fitter solutions (based on the fitness function) will be selected to form a new population. During the cycle, there are three main operators namely reproduction, crossover and mutation. The cycle will repeat for a number of generations until certain termination criteria are met. It could terminate after a fixed number of generations, after a chromosome with a certain high fitness value is located or after a certain simulation time.

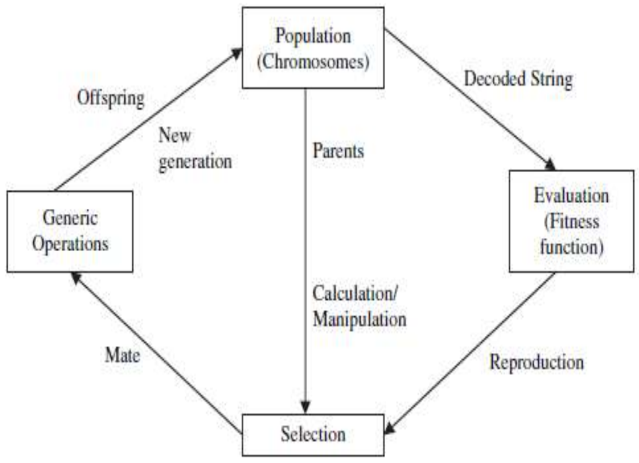


Figure 3 cycle of GA [3]

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

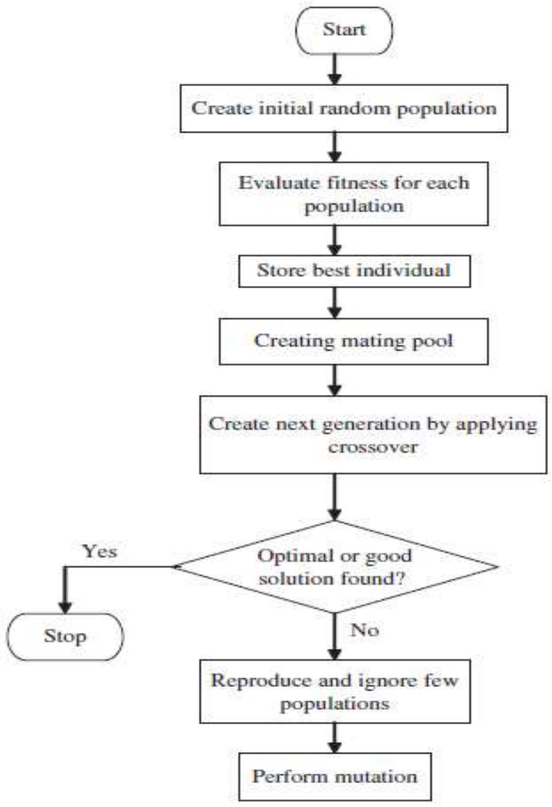

Start
Create initial random population
Evaluate fitness for each population
Store best individual
Creating mating pool
Create next generation by applying crossover
Optimal or good solution found?
Yes
No
Stop
Reproduce and ignore few populations
Perform mutation

Figure 4 Flowchart of GA[8]

5.2) Operators of GA

5.2.1Selection

Roulette wheel selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a roulette wheel where all the chromosomes in the population are placed. The size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome - the bigger the value is, the larger the section is. See figure 5 for an example. A marble is thrown in the roulette wheel and the chromosome where it stops is selected. Clearly, the chromosomes with bigger fitness value will be selected more times. This process can be described by the following algorithm.

1.) [Sum] Calculate the sum of all chromosome fitness's in population - sum S.

1. [Select] Generate random number from the interval (0,S) - r.
2. [Loop] Go through the population and sum the fitness's from 0 - sum s. When the sum s is greater then r, stop and return the chromosome where you are.

Of course, the step 1 is performed only once for each population.

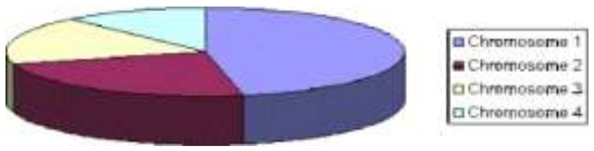


Figure 5 Roulette wheel selections

5.2.2 Crossover

Single point crossover - one crossover point is selected, binary string from the beginning of the chromosome to the crossover point is copied from the first parent, and the rest is copied from the other parent.

Chromosome A= 11110000
Chromosome B= 00001111

After single point crossover,

Chromosome A= 1111000|0
Chromosome B= 0000111|1
Chromosome C= 11110001

Chromosome D=00001110

Two point crossover - two crossover points are selected, binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again.(Robert et al.,1999)

Chromosome A= 1111|000|0
Chromosome B= 0000|111|1
Chromosome C=11111111
Chromosome D=00000000

5.2.3 Mutation

Bit inversion - selected bits are inverted

Chromosome A= 1111000|0
Chromosome B= 11110001

6.) Proposed work

6.1) Working process

Step 1.) Take an algorithm (or source code) of a program.

Step 2.) Convert the algorithm (or source code) of a program into the corresponding control flow graph.

Step 3.) Apply IWD on the control flow graph of a program.

3.1 All the independent paths present in the graph will be generated.

Step 4.) Select the target path.

Step 5.) Then use GA technique to generate the test data as input to the program which will traverse the target path.

6.2) A case study

A case study has been taken, describing a customer's activity of withdrawing money from ATM. The scenario considered here for design of fitness function is that the customer tries to withdraw certain amount from the ATM machine (this withdrawal amount is the initial test data generated randomly, with an assumption that customer entering the withdrawal amount is random).

6.2.1 Pseudo code of ATM Activity

```
Step 1.)  Customer inserts the ATM card in the ATM machine.
Step. 2) Card number is read by the bank database.
Step 3.) Request for the pin number is generated.
Step 4.) Customer enters the pin number.
Step 5.)If pin number is correct
         Then
         Menu appears
         Else
         Enter the pin again
Step 6.) Request for the money withdrawal by the customer is entered in the ATM machine.
Step 7.) Balance request is send from ATM to the bank database.
Step 8.) From bank database, amount request is send to the ATM and asks the customer to enter the withdrawal amount.
Step 9.) Customer enters the withdrawal amount and request is send to the bank database.
Step 10.) Bank database checks whether the enter amount is valid,
          If amount is valid, then debit process takes place
          Else
          No transaction possible.
Step 11.)  If debit process takes place, response from bank database is given to the customer by dispensing cash to the customer.
Step 12.) Customer collects the money.
Step 13.) Request from database to the customer, want a receipt?
Step 14.)  If Customer replies yes to the request.
Step 15.) A printed cash receipt is generated.
```

The ATM system sends the amount and the account number to the bank system. The bank system retrieves the current balance of the corresponding account and compares it with the entered amount for withdrawal. If the current balance amount in the account is found to be greater than the entered withdrawal amount, then the amount can be withdrawn from the ATM and the bank system returns true, after which the customer can withdraw the money, otherwise it checks for the limit if the amount entered for the withdrawal is greater than the total amount (current balance) available in the account, then the amount cannot be withdrawn from the ATM,and bank returns false then, after which the customer cannot withdraw the money. Depending on the return value, the ATM machine dispenses the cash and prints the receipt or displays the failure message.

6.2.2 Algorithm for ATM withdrawal activity

```
1.)  T_ bal = 25000, min_ bal = 1000;
2.)  Am_ left(1: i) = T_ bal – wd_ amt(1:i);
3.)  if wd _ amt(1: i) < T_ bal
4.)  if Am_ left(1: i)  > min_ bal
5.)  success_bal(1:k) = Am_ left(1:i);
    Else
6.)  failure_ bal(1: p) = Am_ left(1: i);
7.)  test data(1: p) = wd_ amt(1: i);
```

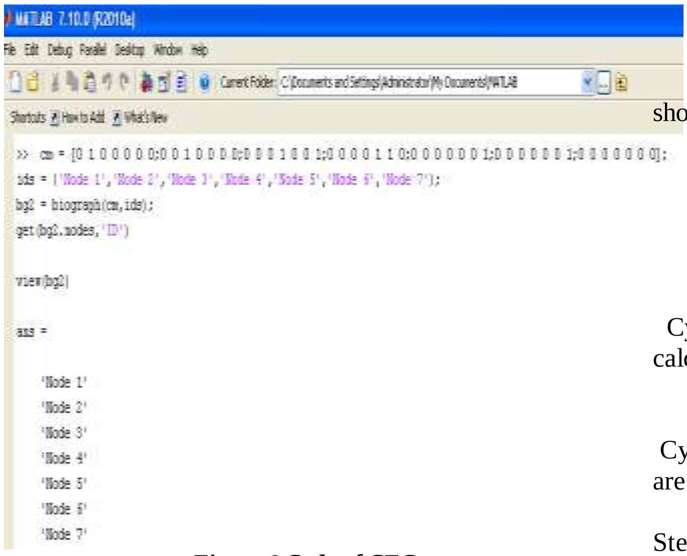
6.2.3 Generation of test data using IWD and GA

Step 1.) Taking an algorithm of an ATM withdrawal activity.

```
T_ bal = 25000, min_ bal = 1000;

1.)  Wd_ amt(1:i)=x(1:i);
2.)  Am_ left(1: i) = T_ bal – wd_ amt(1:i);
3.)  if wd _ amt(1: i) < T_ bal
4.)  if Am_ left(1: i) > min_ bal
5.)  success_ bal(1: p) = Am_ left(1: i);
    Else
6.)  failure _bal(1:k) = Am_ left(1:i);
7.)  test data(1: p) = wd_ amt(1: i);
```

Step 2.) Constructing control flow graph of the given algorithm. Control flow graph can be made manually or automatically .We have made control flow graph manually using c++ programming language and visualization of this control flow graph is given using matlab.


MATLAB 7.10.0 (R2010a)
File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Documents and Settings\Administrator\My Documents\MATLAB
Shortcuts How to Add What's New

```
>> cm = [0 1 0 0 0 0 0;0 0 1 0 0 0 0;0 0 0 1 0 0 1;0 0 0 0 1 1 0;0 0 0 0 0 0 1;0 0 0 0 0 0 1;0 0 0 0 0 0 0];
ids = {'Node 1','Node 2','Node 3','Node 4','Node 5','Node 6','Node 7'};
bg2 = biograph(cm,ids);
get(bg2.nodes,'ID')

view(bg2)

ans =

    'Node 1'
    'Node 2'
    'Node 3'
    'Node 4'
    'Node 5'
    'Node 6'
    'Node 7'
```

Figure 6 Code of CFG

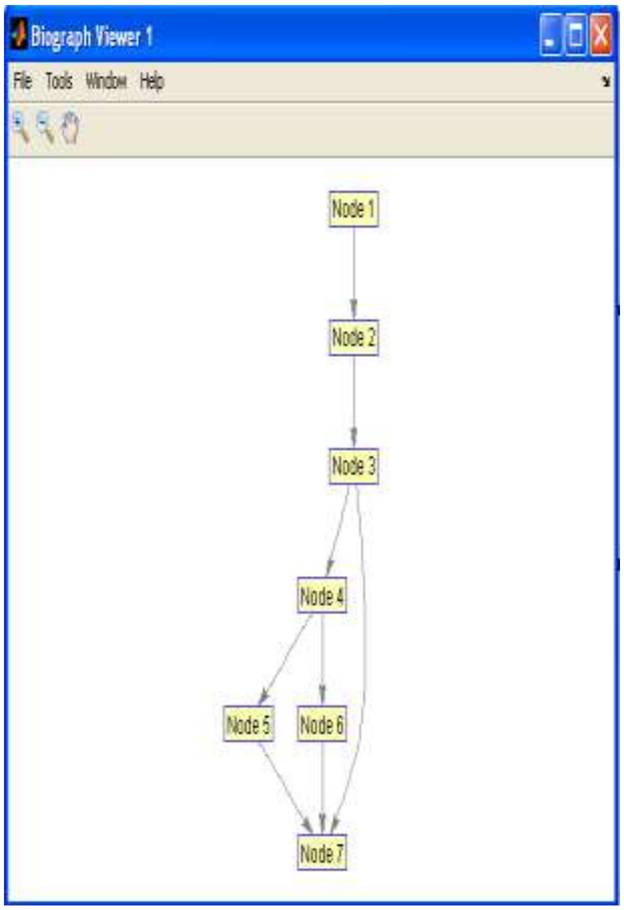


Figure 7 Control flow graph of ATM algorithm

Step 3.) Cyclomatic complexity of graph

Cyclomatic complexity of a graph will give us the number of independent paths.

Cyclomatic complexity can be calculated as shown below:

C (G) = e- n + 2g
Where
e= edges in the graph
n= number of nodes in the graph
g= number of graph

Cyclomatic complexity of the given graph can be calculated as shown below:

C (G) =8-7+2*1
C (G) = 1+2=3

Cyclomatic complexity of the given graph is 3, so there are 3 independent paths in the graph.

Step 4.) Generating all the independent paths present in this graph.

IWD algorithm is used to generate all the paths in this graph.

Algorithm 1.) This algorithm is used to generate cyclomatic complexity of each node

```
Initialization:
Global array visited [N]
/* empty array of size N, where N = number of nodes in the graph */
Input: root node of Graph (N, E)
/* Graph (N, E) = Graph containing N nodes and E edges */
Output: cyclomatic_complexity of each node

Step 1. If node is already visited OR node is an end node then return 1;
Step 2. Add node to visited list;
Step 3. Loop for all branches
Step 3.1 Increment result by cyclomatic complexity of noderesult += cyclomatic_complexity (node);
Goto Step-1
Step 3.2 End loop
Step 4. Return result;
```

Result of algorithm 1

Cyclomatic complexity of node 7 = 0
Cyclomatic complexity of node 6 = 1
Cyclomatic complexity of node 5 = 1
Cylomatic complexity of node 4 = Cyclomatic complexity of node 6+ Cyclomatic complexity

of node 6 =1+1=2
Cyclomatic complexity of node 3= Cyclomatic complexity of node 4+1=3
Cyclomatic complexity of node 2= Cyclomatic complexity of node 3=3
Cyclomatic complexity of node 1= Cyclomatic complexity of node 2=3

Table 1 Cyclomatic complexity of all nodes

Nodes	Cyclomatic complexity
1	3
2	3
3	3
4	2
5	1
6	1
7	0

Algorithm 2.) This algorithm is used to generate the independent paths
Initialization:
Max_Iteration - Maximum time Iteration to be performed
Init_Soil - Initial soil on the path
soil (i,j) - Available amount of soil between node (i) and node (j)
Visited Path - Contains information about node that has been visited
paths (i) - Number of path from node (i) yet to be explored
p (i,j) - Probability of choosing path between node (i) and node (j)
Input: Graph(N,E), cyclomatic_complexity of each node
Output: Path_List = Contains all extracted paths (initially empty list)
Step 1.) Initialize all static parameters (from Step-1.1 to Step-1.8)

Step 1.1 Graph (N, E) having total N nodes
Step 1.2 Max_Iteration = CC which we have found using Algorithm 1
Step 1.3 Velocity updating parameters
av = 1, bv = 0.01, cv = 1 and α= 1
Here, the av, bv, cv and α are user-selected positive parameters.
Step 1.4 Soil updating parameters
as = 1, bs = 0.01 and cs = 1
Here, the as, bs and cs are user-selected positive parameters.
Step 1.5 Initial soil on each edge of graph
Init_Soil = 10000, (It is assigned to each edge, i.e., soil(i,j) = Init_Soil)
Step 1.6 for all IWD, Soil (IWD) = 0
Step 1.7 Initial velocity of each IWD
Init_Vel = 200(It is assigned to each drop)
Step 1.8 Visited_Path = empty list

Step 2. Put IWD on root node of the graph (CFG)
Step 3. Calculate probability for choosing next node (j) from available path of node (i).

Step 3.1 Probability can be find using formulae

$$P(i,j)= (paths(j)/paths(i))+(soil(i,j)/Init\_soil)-((\Sigma soil(i,k)/Init\_Soil)/no.\ of\ k)$$

where, paths (i) = number of path from node (i) yet to be explored (CC(i)),Σsoil(i,k) = Sum of the soil of every path i to k, i≠k. Probability formulae is used for finding probability of a path when there are two nodes are available for moving forward from the current node (i). This function can also tackle the blocked path situation. Along with that for handling the situation of paths having same CC, the concept of soil has also been introduced in the fitness function which is likely to be varied for the different paths.
Step 3.2 using the formulae as mentioned in Step-3.1, find probability for all outgoing paths from the current node (i)
Step 4. Choose next path which is having greatest probability because it is the optimal path where many other paths are yet to be explored, e.g., p(3,4) > p(3,7), then choose path(3,4) and add it to the visited path list.
Step 5. Update the Velocity (IWD) (denoted by velIWD) moving from node (i) to node (j).

$$vel^{iwd}\ (t+1) = vel^{iwd}\ (t)+ a_v /(b_v+(c_v * soil^{2*\alpha}(I,j))$$

where vel^{iwd} (t+1) is the updated velocity of IWD
Step 6. Update time parameter for IWD

$$time\ (i,j;\ vel^{iwd}\ (t+1))=HUD(j)\ /\ vel^{iwd}$$

HUD (j) = CC of CFG × CC at node (j) where, a local heuristic function HUD(j) has been defined for a given problem to measure the undesirability of an IWD to move from one location to the next.
Step 7. For the IWD, moving on path node (i) to node (j), it computes the change of soil.

$$Soil(i,j)= a_v /(b_v+(c_v * \ time^2 (i,j;\ vel^{iwd}\ (t+1)))$$

where, Δsoil(i,j) which IWD loads from path while traversing through that path.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

Step 8. Update the Soil (IWD) (denoted by $soil^{IWD}$) by some amount which it has loaded from path $soil^{IWD} = soil^{IWD} + \Delta soil(i,j)$
Step 9. Update the soil of path between node (i) and node (j).
$soil(i,j) = soil(i,j) - \Delta soil(i,j)$
Step 10. Repeat Step-3 to Step-9 until it encounters the end node or already visited node.
Step 11. Store the whole path as one of the final independent path in Path_ List and decrease CC by 1
Step 12. Repeat Step-2 to Step-11 until, CC (root node)! = 0

Step 5.) Test data generation using Genetic algorithm

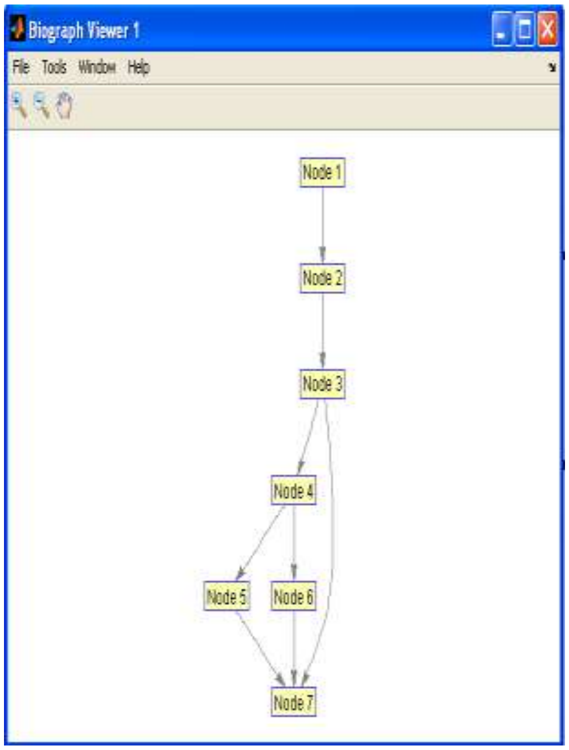


Figure 8 CFG

Node1: wd_ amt (1:i)=x(1:i);

Node2: Am_ left (1: i) = T_ bal – wd_ amt (1:i);

Node3: if wd _ amt (1: i) < T_ bal

Node4: if Am_ left (1: i) ≥ min_ bal

Node5: success_ bal (1: p) = Am_ left (1: i);

Node6:failure_bal(1:k)=Am_left(1:i);

Node7: test data (1: p) = wd_ amt (1: i);

Table 2 Alphabetical notations

Nodes	Corresponding Variable	Alphabetical Notations
1	wd_ amt	A
2	T_ bal	X
3	Am_ left	B
4	min_ bal	C
5	failure_ bal	D
6	success_ bal	E
7	test data	F

After generation of all paths in the graph, a target path is selected as P. The goal of the test data generation problem is to find a program input x on which path P will be traversed.
Path P: 123467
Condition for Path P is A<X, B<C, D=B.
Path P: 123457
Condition for Path P is A<X, B≥C, C=B.
where A=x, B=25000-x, C=1000
Target path=123457
Steps for GA
1.) Intially, we generate a test data (x (i)) randomly as chromosomes in the population.
2.) Initially randomly generated test data is in binary form it is converted into integer form.
3.) Now we have to find the fitness function. Fitness function is made with the help of korel's theory. Without loss of generality, Korel assumed that the branch predicates are simple relational expressions (inequalities and equalities). That is, all branch predicates are of the form: E1 op E2, where E1 and E2 are the arithmetic expressions and op is one of {<, ≤,>, ≥, =, ≠}. In addition, he assumed that predicates do not contain AND or OR or any other boolean operators. Each branch predicates E1 op E2 can be transformed to the equivalent predicate of the form: F rel O (Operator), where F and rel are given in Table-. [11]

Table 3 Korel Distance values

Branch predicate	Branch function F	rel
$E_1 > E_2$	$E_2 - E_1$	<
$E_1 \geq E_2$	$E_2 - E_1$	≤
$E_1 < E_2$	$E_1 – E_2$	<

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 2.114

$E_1 \leq E_2$	$E_1 - E_2$	$\leq$
$E_1 = E_2$	$abs(E_1 - E_2)$	$=$
$E_1 \neq E_2$	$abs(E_1 - E_2)$	$\leq$

F is a real valued function, referred to as branch function, which is:
i. Positive (or zero if rel is <) when a branch predicate is false or
ii.Negative (or zero if rel is = or) when the branch predicate is true.
It is obvious that F is actually a function program input. But this process requires a very large and complex algebraic manipulation. For this reason an alternative approach was used in which the branch function was evaluated. Basis path testing includes both statement testing and branch testing. For example, to test \if a > b then", it has a branch function F, whose value can be computed for a given input by executing the program and evaluating `a - b' expression.
This concept was used in the approach to test the Automatic Teller Machine (ATM) withdrawal task. Test data was generated for a single feasible path in CFG with respect to ATM withdrawal task. The fitness function for the ATM withdrawal scenario was based on the traversal of predicate nodes.
Suppose a situation, withdrawal money is less than the total amount in the account. Now, Am_ left can be less than min_ bal i.e. B < C or Am_ left can be more than min_ bal i.e. B > C or Am_ left can be equal to the min_ bal i.e. B = C. So taking equality condition into consideration, B = C, B- C = 0, abs (B-C). As GA for test data generation is minimization problem, the fitness function `f' is given as 1= abs (B-C). But this functional value will evaluate to infinity when B-C= 0, so to avoid this condition a small delta value (= 0:05) is added to the fitness function. Hence the fitness function in general is given as:

$$f = 1/ ((abs (B-C) + 0.05)^2)$$

More general form of fitness function

$$f = 1/ ((abs (Am\_ left - min\_ bal) + 0.05)^2)$$

$$Am\_ left = T\_ bal - wd\_ amt$$

$$T\_ bal= 25000$$

$$wd\_ bal= x$$

$$Am\_ left = 25000 - x$$

$$min\_ bal=1000$$

Now, the general form of fitness function used in the implementation process is
$f = 1/ ((abs (25000 -x- 1000) + 0.05)^2)$ where $0 \leq x \leq 23900$
Algorithm- 1 shows the approach followed to generate test data for the basis path derived from CFG using GA.
Algorithm 1 - Test Data Generation using GA
Input: Randomly generated numbers based on the target path to be covered.
Output: Test data for the target path.

```
Begin
Gen = 0.
while (Gen < 50) do
Step 1.) Evaluate the fitness value of each chromosome based on the fitness function. Fitness function: $f = 1/ ((abs (25000-x (i)-1000) + 0:05)^2)$ where $0 \leq x \leq 23900$
Step 2.) Use Roulette wheel as selection operator, to select the individuals to enter into the mating pool.
Step 3.) Performed one point and two-point cross over on the individuals in the mating pool, to generate the new population.
Step 4.) Performed bitwise Mutation on chromosomes of the new population.
Gen = Gen + 1;
go to Step 3.
end while
Select the chromosome having the best fitness values which traverse the target path and make it a feasible path, as the desired result (
```

7. Implementation & Results

7.1 Case study 1 (integer type)

Fitness function f(x) =1/ ((abs (25000-x-1000) +0.05) ^2)
In genetic algorithm, we prefer to minimize the fitness function. So, we have to minimize this fitness function.

$f(x)_{x\to\infty} = 1/ ((abs (25000-x-1000) +0.05)$ ^2)

1.) for $0 \leq x \leq 23900$, path 123457 will be traverse and it will become a feasible path.
2.) for $24000 \leq x \leq 24900$, path 123467 will be traverse and it will become a feasible path.
3.) for $25000 \leq x \leq \infty$, path 1237 will be traverse and it will become a feasible path.

Table 4 Experimental setup

S.no	GA parameters	Values
1.)	Population size	50
2.)	Chromosome length	15
3.)	Selection	Roulette wheel
4.)	Crossover	Single point with p_c=0.5
5.)	Mutation	Bitwise with p_m=0.05
6.)	Generations	50

Our target path is P= 123457, so fitness function will be calculated with a limit on x i.e. x will be between o and 23900.
Fitness function $f(x)_{0\leq x \leq 23900} =$ 1/ ((abs (25000-x-1000) +0.05) ^2)

(a) When we consider the case, no money is withdrawn i.e. $0 \leq x \leq 23900$, the objective function value is $0.0002 * 10^{-5}$. (Single point crossover)

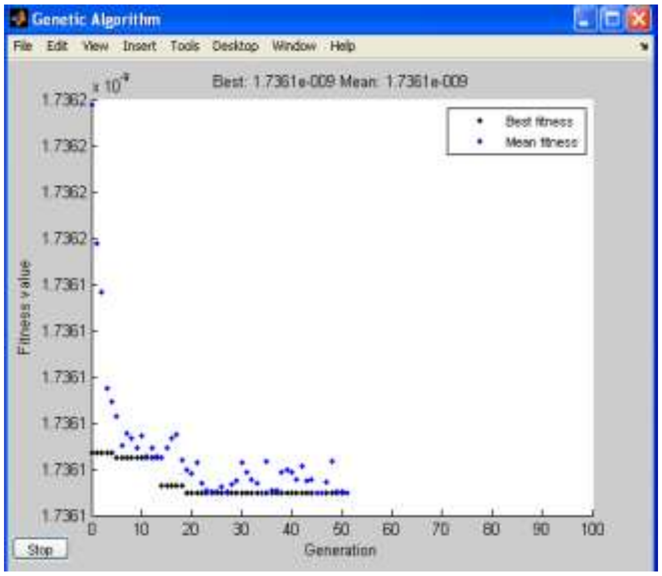


Figure 9 objective function value 1

(b) When we consider the case, no money is withdrawn i.e. $0 \leq x \leq 23900$, the objective function value is $0.0002 * 10^{-5}$. (Two point crossover)

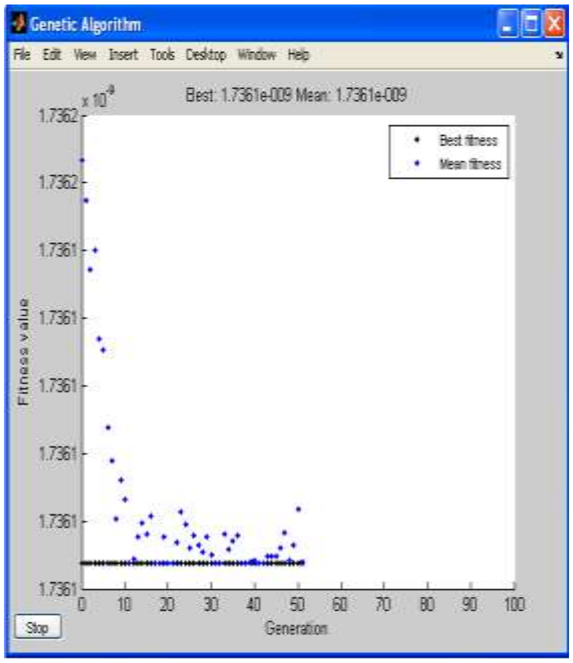


Figure 10 objective function value 2

(c) When we consider the case, money is withdrawn i.e. $100 \leq x \leq 23900$, the objective function value is $0.0002 * 10^{-5}$.

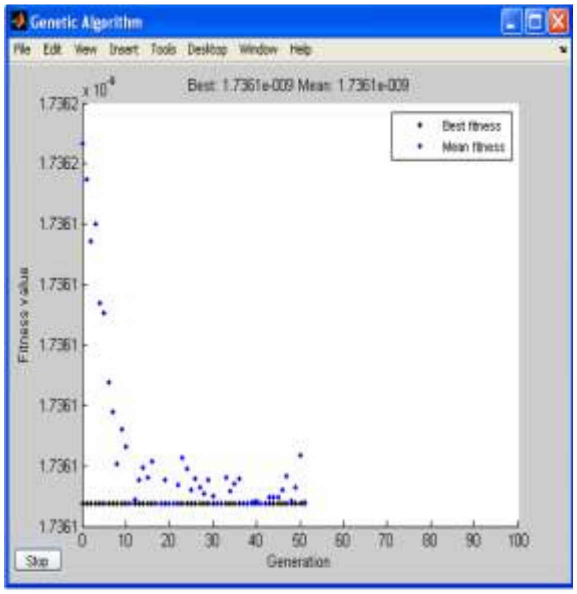


Figure 11 objective function value 3

7.1.1 Results

Table 5 Results of GA

S.no	Input variable(x)	Fitness function value range	Test data %	Path traversed (feasible)
1.)	$0 \leq x \leq 23900$	$(0.0002 – 9.9900) * 10^{-5}$	67%	123457
2.)	$24000 \leq x \leq 24900$	------------------------	3%	123467
3.)	$25000 \leq x \leq \infty$	-------------------------	30%	1237

Conclusion and future work

Software testing is very important from the point of software development process and test data is very important to test the application and write efficient test cases. In this research work, we have shown that how we can automatically generate paths of a control flow graph which will be very useful in case of large applications which have very complex control flow graph for example if we have to make a control flow graph of website www.irctc.com it will be very difficult to manual generate paths from it one by one. In case of test data generation we have shown with the help of results that test data of integer type, generated with the help of genetic algorithm is more optimal than random data generation. In future work can be done to generate conditions on path automatically with the help of parsing technique or a parsing tool and some another swarm or artificial technique can be used to generate data and can be seen whether it is better than genetic algorithm or not.

Acknowledgements

The success of this research depends on the encouragement and guidance I got from many individuals. I take this opportunity to express my gratitude to then in my humble acknowledgement.

First my greatest thanks go to Dr. (Mrs.) Parminder Kaur, Assistant Professor, Department of computer Science and Engineering, Guru Nanak Dev University, Amritsar. She has motivated, encouraged and laughed with me during the year of study. Without the support, valuable feedback and the opportunities created by her, I would not have finished this research paper.

In addition, I want to thank Dr. Gurvinder Singh, Professor and Head, Department of computer Science and Engineering, Guru Nanak Dev University, Amritsar, for his constant encouragement. He is always ready with good and precise feedback, during my study at Guru Nanak Dev University, Amritsar.

I take this opportunity to record my sincere thanks to all the faculty members of the Department of Computer Science and Engineering, for their help and encouragement. I also thank my parents for their encouragement and support.

I also place on record, my sense of gratitude to one and all who, directly or indirectly, have lent helping hand in this venture.

References

[1] Chauhan,Software Testing - Principles and Practices, Oxford UniversityPress,2011

[2] Dahl, Dijkstra, Hoare,Structured programming. Academic Press Ltd., 1972

[3] Deepa,Sivanandam,Introduction to genetic algorithm. Springer Press, 2008

[4] Edvardsson,"A Survey on Automatic Test Data Generation",in Proceedings of the Second Conference on Computer Science and Engineering, (pp. 21-28),2007

[5] Goldberg,Genetic Algorithms. Addison Wesley, 1988

[6] Haibin, Senqi,Xiujuan,"Air robot path planning based on Intelligent Water Drops optimization. IEEE World Congress on Computational Intelligence, pp. 1397 – 140, 2008

[7] Hamed,"Problem solving by intelligent water drops. IEEE Congress on Evolutionary Computation, (pp. 3226-3231), 2008

[8]Haupt, Haupt, Genetic algorithm, Wiley Publications, 2008

[9]Holland.,Adaptation in natural and artificial system, Ann Arbor, The University of Michigan Press,1975

[10]Kaner,Exploratory testing in Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL,2006

[11]Korel,"Automated Software Test Data Generation", IEEE Transactions on Software Engineering, 16(8), 870–879. doi:10.1109/32.57624,1990

[12]Koza,A genetic Programming, Ann Arbor. The University of Michigan Press, 1992

[13]H.Li & Lam,"Software Test Data Generation using Ant Colony Optimization", Journal of World Academy of Science Engineering and Technology, 557-560, 2005

[14]McMinn,"Search-Based Software Test Data Generation: A Survey. *Software Testing", Verification and Reliability*, *14*(3), 212–223, 2005

[15]Myers, C. Sandler, and T. Badgett,The art of software testing. Wiley, 2011

[16]Naaz,"Artificial Intelligence Techniques in Software Engineering",in Proceedings of the International MultiConference of Engineers and Computer Scientists, (pp. 1-3),2009

[17]Pargas,Mary Jean Harrold,Robert Peck," Test-data generation using genetic algorithms", in Software Testing. Verification & Reliability, 9(4):263–282, 1999

[18]Shah-Hosseini,"Problem solving by intelligent water drops", in IEEE Congress on Evolutionary Computation, (pp. 3226-3231), 2007

[19]Shah-Hosseini,"The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm", International Journal of BioInspired Computation, 1(1/2), 71–79. doi:10.1504/IJBIC.2009.022775,2008

[20]Sidhu, Aggarwal, Kundra,Application of intelligent water drop algorithm,2011

[21]Srivastava & Baby, "Automated Software Testing Using Metahurestic Technique Based on an Ant Colony Optimization",International Symposium on Electronic System Design (ISED), (pp. 235 - 240), 2010

[22]Srivastava, Jain, Baheti & Samdani,"Test Case generation using Genetic Algorithm",International journal Information Analysis and Processing (IJIAP), 3 (1), 7-13, 2010

[23]Srivastava, Km,Raghurama,"An Approach of Optimal Path Generation using Ant Colony Optimization", *TENCON 2009 - IEEE Region 10 Conference* , (pp. 1-6),2009

[24]Srivastava, Ramachandran, Kumar.,Talukder, Tiwari & Sharma"Generation of Test Data Using Meta Heuristic Approach",TENCON - IEEE Region 10 Conference, (pp. 1-6),2008

[25]Srivastava,Patel,"Test Data Generation Based on Test Path Discovery Using Intelligent Water Drop", International Journal of Applied Metaheuristic Computing, 3(2), 56-74,2012