Path Prioritization using Meta-Heuristic Approach

Himanshi, Nitin Umesh, Saurabh Srivastava
Computer Science and Engineering
India

ABSTRACT
Software testing is one of the most important phase in development life cycle of any software system as testing assures the quality of the software i.e. a software is bug-free can judged using software testing. Although, creating bug-free software is impossible but we can find out most of the bugs and recover them. Software testing can be done in many ways but here we will focus on structural testing. This paper presents an approach which can prioritize the paths among a set of paths such that they can be executed accordingly and comparison between existing methods is done. All results have been produced using a software developed for the purpose.

1. INTRODUCTION
When the computers were first made, they were big-room sized machines which operate on mechanical relays and glowing vacuum tubes. At Harvard University technicians were running the new computer when it suddenly stopped working. The reason was, a moth was stuck between the relay contacts of the computer. It had apparently flown to the system attracted by the light.

From that day computer bug was born. A bug caused the computer to stop working, thus software's need to be tested in order to prevent flaws which may lead to the system failure.

Software testing is now an integral part of software development life cycle (SDLC). Some companies now have an entire different section of testing team in their company for testing the quality of the product. The main goal of performing software testing is to find the bugs in the software before release of product to the end users. Software testing assures the quality of the product and now-a-days most of the expenses are done for performing quality testing such that errors can be removed and we can have bug-free software.

There is a test criteria hierarchy. As we know there are several testing criteria and we need to give preference one over other on some basis. Obviously, the better one which can provide better result will be preferred over others.

Theoretical analysis concerning the hierarchy shows that the most of the testing criteria are incomparable.

There are two major ways of testing any software product.

- Structural testing
- Functional testing

1.1 Structural testing
Structural testing is also known as white box testing or logic-driven testing, it is the process of testing the internal logic for the program. It covers the lines of code written for the software. There are several types of structural testing such as

- Data flow testing
- Control flow testing
- Statement coverage
- Branch coverage
- Path coverage
- Condition coverage
- Basis path testing

These are few important testing techniques in white-box testing. We will keep our focus on basis path testing.

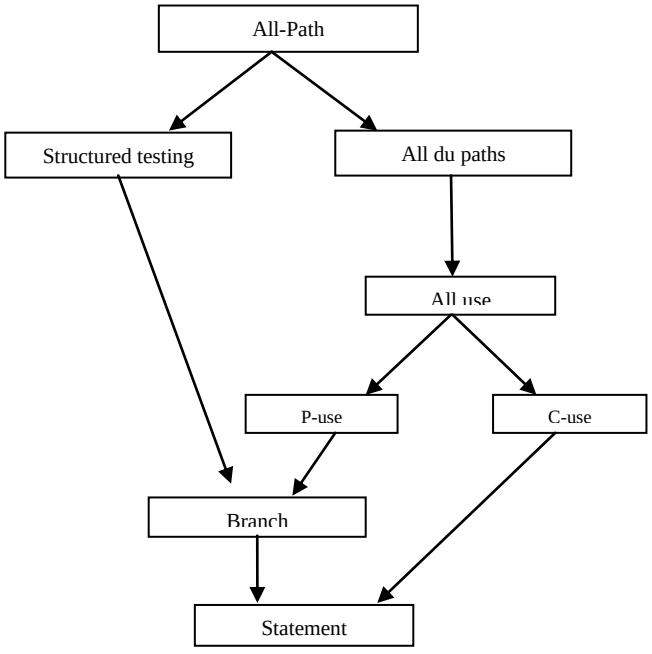

Fig 1.1 Testing criteria hierarchy

1.1.1 Basis path testing

Basis path testing was proposed by Thomas McCabe in 1976. Basis path testing is a kind of structural testing which is based on control flow graph. This method derives a set of feasible independent paths to design a set of test cases through which testing can be performed. In graph based testing there can be infinite no. of paths if there is any loop or cycle in the graph. We can deal with such condition using Cyclomatic complexity which was again given by McCabe.

In Basis Path testing we particularly need to avoid selection of infeasible paths in the graph which can increase the amount of time required for testing, as infeasible paths can't be tested using any set of test cases.

Basis path testing worksheet is shown on next page.

1.1.2 Functional testing

Functional testing which is also known as black-box testing is another very famous technique of software testing which only concerns about the functionality of a software product. In this testing we test software as a whole. We give input to the software and check for the output. We have a reference model to check whether the output is correct or not. If the output is correct we consider the software as correct. The functionality can be tested either by an expert tester or by any other person who may not have the best of knowledge about the software.

One good thing about this testing is that, the tester need not to be an expert programmer or having the in-depth knowledge about the code on which the software is built.

1.2 Cyclomatic complexity

A graph with M predicate nodes, have 2m possible paths, and if the graph contains any loop (one or more), it may have an infinite number of paths; to overcome this situation we use Cyclomatic complexity, as it is an important method to reduce the total number of paths. Cyclomatic complexity is used to generate a number of linearly independent paths in the graph. A path is considered as linearly independent path if it has at least one new node than previous path.

Cyclomatic complexity is also denoted as V(G) while v means the Cyclomatic number in graph theory and G stands for that the complexity is a function of the graph. We have many formulae to calculate the Cyclomatic complexity and one of these is V (G) = e – n + 2, where e represents the number of edges in the CFG and n denotes the no. of nodes. Another formula for calculating V (G) is no. of predicate node + 1. V (G) = P + 1 where P stands for no. of predicate nodes in control flow graph.

1.3 Control Flow Graph

CFG describes the logical structure of the source code or software under test. Every control flow graph consists of various nodes and edges. The nodes in CFG shows computational statements and the edges represent control switching between nodes.

We use Control Flow Graph (CFG) diagrams to generate optimal or efficient path for software under test (SUT).In other words, a control flow graph describes how the control flows throughout the program.

CFG based testing provides all statement coverage, branch nodes coverage, event coverage and provides all path coverage. This is the most effective technique for software testing.

1.3.1 Node

It is expressed as a labeled circle, representing, one or more statement, decision, condition, procedures of a program etc.

1.3.2 Control Flow

It is expressed by an arc or directional edge from one node to another representing statement flow of the program. In CFG a node with condition is known as predicate node i.e. if out-degree of any particular node is more than one then it is said to be a predicate node.

1.3.3 Ant Colony Optimization Algorithm

Ant colony optimization (ACO) is a population-based meta-heuristic that can be used to find estimated solutions to tricky optimization problems.

A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality.

—Tabu Search, Fred Glover and Manuel Laguna, 1998

A predominantly triumphant meta-heuristic is inspired by the behaviour of real ants. Starting with Ant System, a number of algorithmic approaches based on the very same facts were developed and applied with substantial success to a diversity of combinatorial optimization problems from scholastic as well as from real-world applications. The chapter introduces ant colony optimization, a meta-heuristic framework which covers the algorithmic approach mentioned above. The ACO meta-heuristic has been projected as a common frame for the existing applications and algorithmic variants of a diversity of ant algorithms. Algorithms that fit into the ACO meta-heuristic framework will be called in the following ACO algorithms.

In ACO, a set of software agents called mock ants hunt for good quality solutions to a given optimization problem. To apply ACO, the optimization problem is transformed into the problem of judgment of the best path on a weighted graph. The mock ants incrementally construct solutions by affecting on the graph. The solution building process is stochastic and is influenced by a pheromone model, that is, a set of parameters associated with graph components (either nodes or edges) whose values dynamically change at runtime by the ants.

The thought behind ant algorithms is then to use a form of artificial stigmergy to harmonize societies of artificial agents. ACO is motivated by the foraging actions of ant colonies, and targets discrete optimization problems.

Ant colony optimization (ACO) is a very notorious technique which was used for several purposes such as path sequencing or shortest path selection in travelling salesman problem and many more. There are quite a few extension of this algorithm

given by many researchers which were then used in different field of computer science and mathematics.

In computer science and engineering, ant colony is termed as probabilistic technique in order to solve computational problems. Ant colony algorithm uses graphs for finding the superior paths. Ant colony algorithm was used to generate test sequences for state based testing. This algorithm was used to find the shortest path between the start node and any other random intermediary or destination node, this algorithm has loom to cover all the nodes in execution state sequence graph (ESSG) but unsuccessful to do so at higher level or strong level.

Ant optimization technique was majorly applied to the area of testing, where one needs path sequencing in a set of paths to be tested. This algorithm was focused on finding the test data for control flow based testing. A novel approach of testing was given for data flow testing via ant colony optimization algorithm.

There are some common extensions of ACO algorithm but, in this paper, the major prominence is given on the selection of shortest feasible path which needs to be tested first in order to get efficient algorithm. As we have discussed earlier, ACO is based on graph thus, we have nodes and edges collectively forming a graph which then needs to be traversed in order to get path sequence which can then be tested after applying test cases.

In ACO we calculate probability of each path and on the basis of probability the priority is measured.

There are four parameters on which probability depends.

1) Feasibility of path (f_{ij})
2) Pheromone value (τ)
3) Heuristic value (μ)
4) Visited status (V_s).

1.3.4 Feasibility (fij)

It can be defined as the availability of edge from node i to j.

Fij = 1 if possible path exists from i to j

Fij = 0 if possible path does not exist from i to j

1.3.5 Pheromone value (τ)

Pheromone helps ants to make decision in prospect. It keeps a trace from path i to j. The pheromone value is updated after each path is traversed.

1.3.6 Heuristic value (μ)

It indicates the visibility of a path for an ant at current vertex i to j.

1.3.7 Visited status (Vs)

It shows the status of all nodes traversed by any ant p for any state i.

Vs = 0, node is not traversed by ant p.

Vs = 1, node is already traversed by p.

A node can be simply denoted using N and edges can be denoted using E.

Related works:

Zhonglin et al., (2010) put forward an improved approach for basis path testing. This technique combines the baseline method with dependence relation analysis. This method generates a set of linearly independent paths, which we call basis paths. However, when applying these basis paths to designing test cases, we will always find that some of them are infeasible. These infeasible paths are impossible to test using any set of test cases. Thus, we need to avoid selection of infeasible paths using some technique such that an efficient path selection technique can be produced.

Qingfeng et al. (2011) elaborated the work of zhonglin for selection of infeasible paths. In this paper he proposed a new approach for selection of independent paths and at the same time avoiding selection of infeasible paths. He illustrated his work on the program triangle showing the effectiveness of the work.

Kumar et al. (2012) discussed the basis path testing as an imperative testing method in white box testing. As, basis path testing follows internal logic thus it generates a feasible set of independent path present in source code and is known as basis path. Some of these paths may be infeasible.

Balakrishnan et al. (2008) proposed a method to determine the semantically infeasible paths in program using abstract interpretation. Their technique uses path-insensitive forward and backward run sequence of an abstract interpreter to deduce paths in the CFG that cannot be exercised in tangible executions of the program.

Srivastava et al (2009) proposed an approach for optimal path generation using ant colony optimization algorithm. In this paper author presents a simple and novel approach using ACO for optimal path identification using basic property of ants. Let us take an example of CFG for program Product and see the no. of feasible and independent paths in it.

Prog.1

```
1. Begin
2. int no., product;
3. bool done;
4. product=1;
5. input(done);
6. while(!done)
{
7. input(num);
```

8. product= product*no.;

9. input(done);

10. }

11. output(product);

12. end

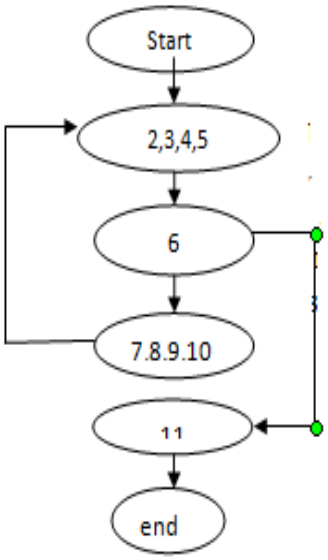

Fig 2.1 CFG for program product

We can now calculate the Cyclomatic complexity and no. of paths in the graph.

Cyclomatic complexity = no. of predicate nodes + 1

1+1 = 2

OR) V (G) = no. of cyclic region = 2

Thus, CC or Cyclomatic complexity or V (G) = 2

Path1: start→2→3→4→5→6→11→end

Path2:
start→2→3→4→5→6→7→8→9→10→2→3→4→5→6→11→end

Now the aim is to determine out of these two paths which path must be selected first for testing.

Proposed work:

The aim of current study focuses on developing a concept of optimizing the way of prioritizing the paths among the set of feasible paths generated from the control flow graph (CFG). There is a great deal of research on path prioritization for path testing or routing. There are several methods of solving such problems. A very renowned travelling salesman problem is one of the best example for which path prioritization becomes a necessity. Although, there are several way of solving travelling salesman problem but there are many other problems which needs to be solved.

One such problem is path prioritization in basis path testing. In order to test the path in efficisent and fast manner we need a prioritization approach which can easily solve our problem and prioritize the path from the set of path such that they can be tested in same order using path prioritization technique.

Basis path testing uses CFG and Cyclomatic complexity to carry out the process. But there are several limitations and drawbacks of control flow graph. A control flow graph may or may not be providing feasible set of path. We need to take care of that thing. A feasible set of path must be generated in order to obtain a correct testing path. In CFG there are many paths which cannot be called in any condition.

The main objective of the work is to produce an algorithm which can prioritize the shortest path first to longest path last among the set of paths first.

- To develop software which gives higher priority to the paths with shorter length i.e. if the path is shorter then it can be given higher priority on the basis of probability such that it can be executed before other longer paths.

Formulae:

- *Pcum = Pij/Lk*
- Pcum is average probability of any path
- $P_{ij} = (\tau_{ij}).^{\alpha} * (\mu_{ij}).^{\beta} / \Sigma.(\tau_{ij}).^{\alpha} * (\mu_{ij}).^{\beta}$
- Tij = (1-p)Tij + Σ delTij
- or
- Tij = (Tij)^x+ (Nij)^y
- Tij is pheromone value
- Nij = 1/Dij
- Nij is heuristic value
- delTij = Q/Lk
- del Tij is diff. in pheromone
- Lk is length of path
- Q is random no.

Steps of iteration:

For both ants, Lk = 10 for ant 1 and 5 for ant 2

Except the length parameter, all other parameters are taken same as we need to show that shorter paths can have greater probability thus priority increases.

Q is a random value which should be same for every ant

Q for ant 1 = Q for ant 2 = 200

Δ Txy = Q/Lk , ΔTxy = 200/10 for ant 1, 200/5 for ant 2

Pheromone evaporation constant (PEC) must lie between 0 and 0.5

Here value of PEC is taken as .1

Calculate Txy = (1-P)Txy + Δ Txy

Calculated Txy for ant 1 and ant 2 is, 20.9 and 40.9 respectively.

In order to calculate Nxy we need value of Dxy

Nxy = 1/Dxy

Value for Nxy for ant 1 and 2 is calculated: 0.5 for both the ants.

Finally we need to calculate Pxy which requires value of N

N is the no. of nodes connected to parent node in any CFG.

Pxy is calculated using ant colony optimization formula

$$P_{ij} = (\tau_{ij}).^{\alpha} * (\mu_{ij}).^{\beta} / \sum.(\tau_{ij}).^{\alpha} * (\mu_{ij}).^{\beta}$$

Here Pij = Pxy , Τij = sTxy, μij = Nxy

Pxy for ant 1 and ant 2 is calculated at value of N = 2: 0.5 and 0.5

The input will be taken again and same procedure will be followed.

Lk = 24 (ant 1) , Lk = 12 (ant 2)

Q = 120

Δ Txy = 5 and 10

PEC = .2

Txy = 21.72 and 42.72

Dxy = 1 and 1

Nxy = 1 and 1

N = 4

Pxy = 0.25 and 0.25

In the third iteration we will take all the values same but for showing the effectiveness of algorithm we will, this time, take value of ant 1 less than the value of ant 2 unlike previous 2 iterations.

Lk = 5 (ant 1) and 10 (ant 2)

Q = 20, 20

Δ Txy = 4 and 2

PEC = 0.3, 0.3

Txy = 19.204, 31.904

Dxy = 3, 3

Nxy = .33, .33

N = 2, 4

Pxy = 0.5, 0.25

Pxy final = 1.25 and 1

Path length final = 39 and 27

Pcum = Pxy final /Path length final

Pcum (ant 1) = 1.25/39 = 0.032

Pcum (ant 2) = 1/27 = 0.037

Pcum for ant 1 < Pcum for ant 2

2. CONCLUSION

Path prioritization is major necessity to efficiently test all the paths involved in CFG so we can prioritize the paths for testing using ant colony optimization algorithm by prioritizing the paths by calculating probability of selection of each node. In path testing we start from the shortest path first. The proposed approach allows tester to find out the priority for each path among the set of paths and put them in ascending order on the basis of path length.

Thus, the proposed approach allows tester to find out the probability for each path and priority of the shortest path comes out to be maximum i.e 1(first). For this approach a software is developed which is helpful is finding out the probability under given path length for different ants, and then sequence them on the basis of higher to lower priority and in ascending order of path length.

3. REFERENCES

[1] Baby, K. M. (2009). An Approach of Optimal Path Generation using Ant colony optimization. *IEEE* , 1-6.

[2] Bhuvnesh Sharma, I. G. (2011). Software Coverage : A Testing Approach through Ant Colony Optimization. *Swarm, Evolutionary, and Memetic Computing - Second International Conference, SEMCCO 2011* (pp. 618-625). vishakhapatnam: Springer-Verlag Berlin Heidelberg 2011.

[3] Christian Blum, M. B. (2005). Combining Ant Colony Optimization with Dynamic Programming for Solving the k-Cardinality Tree Problem. *Computational Intelligence and Bioinspired Systems* , 25-30.

[4] Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms.* italy: the Milano polytechnic.

[5] Ghiduk, A. S. (2010). A New Software Data-Flow Testing Approach via Ant Colony Algorithms. *Universal Journal of Computer Science and Engineering Technology* , 64-72.

[6] Gogul Balakrishnan, S. S. (2008). SLR: Path-Sensitive Analysis through Infeasible-Path Detection and Syntactic Language Refinement. *springer verlag* , 1-16.

[7] Greco, F. (2008). *travelling salesman problem.* croatia: in-teh.

[8] Hunt, t. (2002). *Advanced Topics in Computer Science: Testing.* wales: swansea univesity.

[9] Mathur, a. (2007). *foundation of software testing.* new delhi: pearson education.

[10] McCabe, t. J. (1976). A Complexity Measure. *IEEE transactions on software engineering* , 308-320.

[11] Mousavi. (2012). Path Testing. *Eindhoven University of Technology, The Netherlands* , 1-7.

[12] Qingfeng, D. (2009). An improved algortihm for basis path testing. *IEEE* (pp. 175-178). hefei: IEEE.

[13] Rai. (2009). An Ant Colony Optimization Approach to Test Sequence Generation for Control Flow based Software Testing. *ICISTM' 09* (pp. 345-356). berlin: springr.

[14] Roggenbach, H. S. (2002). *Topics in Computer Science: Testing, Path Testing".* wales: swansea university.

[15] Sommerville, i. (2009). *software engineering.* london: pearson edition.

[16] Srivastava, p. r. (2010). Automated Software Testing Using Metahurestic Technique Based on An Ant Colony Optimization. electronic system design .

[17] Stutzle, t. (2004). *ant colony optimzation.* london: MIT press.

[18] T. Bharat Kumar, N. H. (2012). An Catholic and Enhanced Study on Basis Path Testing to Avoid Infeasible Paths in CFG. *Global Trends in Information Systems and Software Applications* , 386-395.

[19] Zhang Zhonglin, M. L. (2010). An Improved Method of Acquiring Basis Path for software testing. *ICCSE'10* (pp. 1891-1894). hefei: IEEE.

[20] Zhao, R. (2012). A Path-oriented Automatic Random Testing based on Double Constraint Propagation. *IJSEA* , 1-11.