A Structural Testing Model Using SDA Algorithm

Saju Sankar S[1(✉)] and Vinod Chandra S S[2(✉)]

[1] Department of Computer Engineering, Government Polytechnic College, Punalur, India
tkmce@rediffmail.com
[2] Department of Computer Science, University of Kerala, Thiruvananthapuram 695581, India
vinod@keralauniversity.ac.in

Abstract. Path testing is the most needed and useful coverage criterion in structural testing. Tracing and obtaining the resultant paths is the main problem in path coverage testing. Evolutionary techniques are adopted in many software product evaluation methods such as generating and selection of input test data. The priority of the feasible paths is also to be determined. In this paper, we proposes an optimization algorithm for identifying the effective test data execution paths in control flow graph for the program module under test and finding the most efficient test paths using modified smell detection agent based optimization algorithm. New innovations are being conducted for bio-motivated algorithmic techniques from the characteristics of animal behavior. Smell detection agent based algorithm helps to identify most feasible paths and it uses sequential search to obtain all paths in a graph. The tester achieves the paths to be tested through a number of smell spot values from the source node to the target node. We will use control flow graph to produce perfect test paths and cyclomatic complexity number for obtaining the number of feasible test paths. The best feasible paths are prioritized using smell detection agent algorithm such that all the paths are thoroughly tested which ensures structural testing. This algorithm generates paths equal to the cyclomatic complexity. It can be illustrated that the proposed approach guarantees full path coverage.

Keywords: Structural testing · Path testing · SDA algorithm · CFG

1 Introduction

Testing is considered as one of the most important process in the life cycle of software development [1]. There are different software testing techniques like structural, functional testing and its hybrid model. The most significant structural testing approach known as Basis Path Testing (BPT) focused into the many ways of evaluating software source code. The emphasis in this method is to develop test data inputs such that it produces all feasible efficient test paths connecting all the nodes and edges of the graph. Path testing has the advantages of thorough testing, more coverage, unit testing, integration testing, maintenance testing, regression testing etc. The main advantage is that the testing effort can be estimated in proportional to the logical complexity of the software.

Y. Tan et al. (Eds.): ICSI 2020, LNCS 12145, pp. 405–412, 2020.
https://doi.org/10.1007/978-3-030-53956-6_36

We used Control Flow Graph (CFG) in BPT and calculated cyclomatic complexity. From that value we will be able to determine all the possible paths from source node to the destination node [2].

McCabe developed the concept of path coverage based basis path testing in the period of 1980's which utilized cyclomatic complexity [3]. In path testing there are different paths from source to destination. Not all paths are feasible while understanding the functionality of the software. It varies depending upon the different types of control statements used in the module and its output boolean values. To prioritize the feasible and infeasible paths, we need a selection procedure. An algorithm can be designed to identify all the basis paths and its priority ranking will help effective testing. In most of the basis path testing techniques, the paths are identified without any prioritizing. An ant colony optimization algorithmic approach was used in identification of paths with its priority [4].

In this paper, we propose Smell Detection Agent (SDA) algorithm that selects all paths and prioritizes the feasible paths. The algorithm is a nature inspired optimization algorithm suitable for identification of optimal paths with its priority in a graph [5, 6].

2 Path Driven Testing

In this approach, the purpose is to test the different paths from the root node to the destination node by which all combinations of various decision or control statements are executed at least once. The technique is based on the logical structure of the program. A graph (CFG) is drawn with all the feasible paths and verified during testing.

Control Flow Graph (CFG): The logical complexity of the program module to be tested is drawn with a CFG. The CGF contains several nodes and edges. The nodes denote executable code lines whereas the edges denote the flow of control between the nodes. All efficient paths are generated with the aid of a CFG diagram.

Cyclomatic Complexity: The maximum number of possible paths in a graph with M predicate nodes is 2^M and if the CFG has any looping statements, then there will be countless number of test paths. The factor of cyclomatic complexity number is an important parameter to minimize the total count of feasible test paths. Cyclomatic complexity number is necessary for the validation of linearly independent test paths in a graph. There are two factors associated with a CFG, one is the cyclomatic number denoted by 'V' in graph theory and the other is the complexity value 'G' as a function of the graph.

The aim of testers is to evaluate all the feasible paths in the CFG. The major challenge in testing is to find the optimal and feasible paths. Hence to find the optimal path, a priority ranking is done for all the feasible paths. The path with highest priority will be initially selected for testing and it continues until the lowest priority path is tested.

Procedure of Basis Path Testing: A software module contains various independent paths to be tested. All these paths should be tested at least once in basis path testing.

Following are the various steps of testing process.

1. Develop the CFG of the program module to be evaluated.

2. Determine the cyclomatic complexity of the CFG, for finding the possible number of linearly independent test paths.
3. Create sets of basis test paths using the baseline method:

   a. Select the first feasible independent path to be tested.
   b. Back trace the independent paths by suddenly moving to each predicate node to create newer paths.

Evaluation of Normal Path Testing: Examine the software program "test" which uses switch case constructs. A CFG is drawn using join (J1, J2) and the graph is depicted using entry and exit criteria as shown in Fig. 1.

```
Program "test".
     1. read(x)
     2. if (x < 0) then
     3. print("negative");
        else
     4. print("positive")
        endif
     5. switch(x)
        case 1:
     6. print("SUN")
        break;
        case 2:
     7. print('MON')
        case 3:
     8. print("TUE")
           break;
      default:
     9. print("OTHER")
        end switch;
```

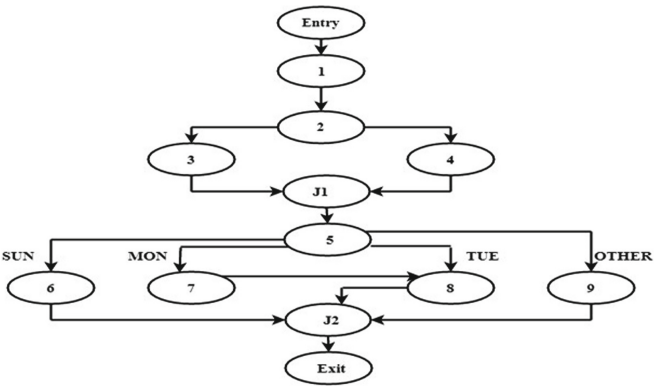

Fig. 1. CFG of program 'test'.

Cyclomatic complexity factor, V (G) = Edges−Nodes + 2, V (G) = 16 − 13 + 2 = 5.

The test paths generated are

Test Path TP1 Entry → 1 → 2 → 3 → J1 → 5 → 6 → J2 → Exit
Test Path TP2 Entry → 1 → 2 → 3 → J1 → 5 → 7 → 8 → J2 → Exit
Test Path TP3 Entry → 1 → 2 → 3 → J1 → 5 → 8 → J2 → Exit
Test Path TP4 Entry → 1 → 2 → 3 → J1 → 5 → 9 → J2 → Exit
Test Path TP5 Entry → 1 → 2 → 4 → J1 → 5 → 6 → J2 → Exit
Test Path TP6 Entry → 1 → 2 → 4 → J1 → 5 → 7 → 8 → J2 → Exit
Test Path TP7 Entry → 1 → 2 → 4 → J1 → 5 → 8 → J2 → Exit
Test Path TP8 Entry → 1 → 2 → 4 → J1 → 5 → 9 → J2 → Exit

In the above example, the number of paths identified is 8 (due to usage of switch case construct) and since the cyclomatic complexity obtained is only 5, we have to weed out infeasible paths. Saurabh et al. [7] proposed an approach using ant colony optimization algorithm, which selects only the feasible paths and prioritized the feasible paths. The algorithm used the factors such as path feasibility, past experience, path visibility and the visited status of path. The model is featured as a directed graph approach and the model also denotes the system to be tested and shows the various test paths of the model during its execution. The best sequence of the path is created automatically after the implementation of the optimization algorithm. The highest priority path is selected first and successively all the other linear paths in the control flow graph can be tested.

Jun yan et al. suggested another suitable and efficient method to create feasible paths for basis path testing [8]. There are two steps in creating feasible paths i) produce a limited set of feasible paths *P* which fulfills the coverage criteria, ii) obtain a minimum subset *p* of set *P* such that *p* fulfills the test coverage. Two conditions should be satisfied by a path when belonging to a basis path set: a) the test paths should be properly feasible and b) the test paths should be linearly independent of all other selected paths.

3 SDA Algorithm for Path Testing

Canines are considered as the earliest animal disciplined by human [5]. They helped men in hunting and its deep odour helped man in finding the exact position of the animal to be hunted. Now also, canines are used by police personnel for tracing the route of culprits from the area of crime. Many problems faced by men cannot be solved conventionally. These problems can be solved with the introduction of new algorithms. In such contexts, solution to the computational problems can be found by the usage of natural phenomena like animal habits or actions. It can be suitably used to solve problems having computational complexity and are asymptotically NP-hard.

Sniffing is used to evaluate priority of different paths which can be implemented by way of an algorithm in the search space [9]. This search space or domain of the problem to be remedied is treated as an area with smell trails and agents motivated from canines are taken to discover all paths, which points to the solution. The concept of canine's path tracing nature helps to develop necessary environments to solve problems regarding path identification. We can draw this nature-motivated technique in co-ordinate geometry. The search space is formed as a Cartesian rectangular plot with specific values mentioning

the space and these values may be changed depending upon the specific constraints of the problem. Every coordinates in the area are not reachable, but some selected arbitrary points that can be visited by the SDAs. The coordinates are denoted as smell spots that help in solving the problem to a subset of points. The values of these smell spots are saved in two parameters. The first is a trailing value of smell from the destination node. The second is a signature value of the SDA that has been denoted as a smell spot.

We propose a SDA algorithm derived from the natural behavior of canines [5]. SDA is a multi-agent algorithm that can be used in any optimum path identification. The algorithm is modified to find the number of feasible paths in a CFG. There are two parameters associated with SDA algorithm i) the assigned signature value that can be used to specify smell spots and ii) the radius value that specifies their olfactory capability. The two parameters are stored in a data structure that is beneficial in the development of the algorithm – data structure ($D_{sda)}$ of the SDA and data structure ($D_{ss)}$ of the smell spot.

Algorithm

Let

N: Count of nodes in the CFG.
N1: Count of SDAs.
N2: Count of smell spots
N3: Count of SDAs that are feasible and reaches the destination

R: radius of the smell spots in the increasing order to the destination
s: the smell value of each node minimum at the source.
P: priority of each path.

1. Assign initially the SDA's with integer values as signature indices and radius values in the increasing direction, thereby the SDA is treated in the progress of traversing in the nodes having highest radius.
2. Select N2 points (nodes) inside the region of the graph as Cartesian values (plots) as smell spots(s).

$$s = 1/(x + y * d)$$

Where 'd' is the Cartesian distance between the smell spots (nodes), the destination and x, y are proportionality constants.

3. Initialize each SDA to the source point.
4. For each SDA from 1 to N1

4.1 Select the unmarked point (within the radius) from the increasing order of smell value.
4.2 Move the SDA forward by earmarking the SDA signature (visiting status).

5. Step 4 is repeated until all SDA's reach the destination in a way all independent paths are traversed at least once.

Path Sequence Generation and Prioritizing Using Modified SDA Algorithm

This algorithm adopted a sequential searching method to obtain test paths in the CFG. The SDA finds a test path from a collection of smell spot values from the root node

to the destination node. For 'n' agents, there will be 'n' paths returned by the algorithm. The feasible paths are prioritized from these 'n' paths. Also the final number of nodes is received. The initial smell value of each node is contained in the node location coordinates. The values get updated while traversing from source to the destination. Identification of the next source and destination nodes will provide the best path. For the calculation of smell value of each node from the destination node, the values of initial smell, decrement count which is inverse of total and effective distance are considered. The values of smell are updated; all the SDA's are initialized with ID value, current node and length. Based on the smell value of each node, each SDA finds a path.

Identification of the path is done by considering the node, which has the highest smell value from the current node. This identification results in assigning highest smell node as current node and this looping process will continue until the destination is reached. The SDA is assigned with a flag 'stop', when the SDA arrives at the destination. The unique paths are identified from the SDA's who have arrived the destination with highest smell value. The optimized path is found by comparing the total number of nodes visited by each SDA. For CFG, the number of nodes, weight assigned to each node, maximum smell value and maximum radius are considered. The weight of each edge is proportional to the maximum number of times; each node is visited by an SDA. The priority is top for the unique path having maximum smell value and depends on the weight assigned to each edge.

4 Results

In the CFG of the example program 'test', the SDA algorithm works as follows. Initially count of nodes N = 13, initial smell value, s = 1, count of SDAs, N1 = 1. The count of smell spots = N2 = N = 13, Radius or distance from source is initially zero. Figure 2 gives the working of our proposed algorithm for the example program 'test' discussed in Sect. 2.

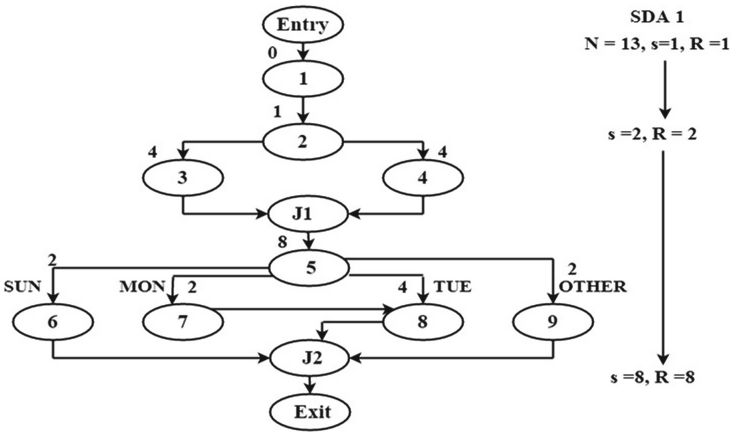

Fig. 2. CFG of program 'test' using modified SDA algorithm.

As per our modified SDA algorithm, the basic paths are traversed by the SDA and fix the priority. In basis path testing, all paths need to be tested, but one test engineer cannot be aware of all the important paths. The SDA algorithm proposed in our model gives all the identified feasible paths in the order of priority. Each edge of the CFG has a weight which depends on the smell spot value. Also, SDAs identify all the test paths in a CFG. In the above program 'test', our algorithm gives a priority wise list of path testing.

Test Path 1 Entry → 1 → 2 → 4 → J1 → 5 → 8 → J2 → Exit
Test Path 2 Entry → 1 → 2 → 4 → J1 → 5 → 7 → 8 → J2 → Exit
Test Path 3 Entry → 1 → 2 → 4 → J1 → 5 → 6 → J2 → Exit
Test Path 4 Entry → 1 → 2 → 4 → J1 → 5 → 9 → J2 → Exit
Test Path 5 Entry → 1 → 2 → 3 → J1 → 5 → 8 → J2 → Exit
Test Path 6 Entry → 1 → 2 → 3 → J1 → 5 → 7 → 8 → J2 → Exit
Test Path 7 Entry → 1 → 2 → 3 → J1 → 5 → 6 → J2 → Exit
Test Path 8 Entry → 1 → 2 → 3 → J1 → 5 → 9 → J2 → Exit

In Ant Colony Optimization (ACO) algorithm, the paths are selected randomly for the path generation from source to destination [10]. In our SDA algorithm, the selection of optimal paths is based on the maximum weight assigned to each most traversed edge of the CFG. In ACO algorithm, the routing is done based of decreasing pheromone value while in SDA algorithm, the olfactory capability is increasing from source to the destination node thereby time complexity is reduced [11, 12] (Table 1).

Table 1. Comparison of ACO and SDA algorithms employed in structural testing.

Algorithm	ACO	SDA
Count of independent paths	All independent paths are identified with some paths have equal priority	All the 8 independent paths are identified with priority from 1 to 8
Swarm communication	Ants communicate called stigmergy	Each Canine or agent has a territory
Algorithm applicability	ACO is suitable to problems where source and destination are predefined	Multiple agents are employed for a faster solution
Problem representation	A construction graph is used to mark ACO's solution space	Cartesian rectangular plot with specific values mentioning the area
Time complexity	$O(n^2)$	$O(\|E\| + \|V\|\log\|V\|)$

5 Conclusion

We have demonstrated the test paths creation methods for basis path testing in this paper and proposed a suitable optimization procedure for feasible test path generation

in structural testing by using SDA optimization algorithm. After implementation of this method, the algorithmic process selects the best test path sequence based on its priority. The highest priority test path is selected first for test execution and in successive steps all the next priority independent paths in the control flow graph can be tested. The SDA Algorithm tends to be more beneficial for better path coverage in basis path testing. The model avoids duplicate paths based on SDA signature and the visited status of the nodes. The model can be modified by an automated method as a future work. The results shows that the SDA algorithm based structural testing can be extended for the generation of optimal and prioritized generation of test paths for multipath software modules.

References

1. Granno, G.: A new dimension of test quality - assessing and generating higher quality unit test cases. In: ISSTA 2019, China, pp. 15–19. Software Evolution and Architecture Lab, University of Zurich, Zurich, Switzerland (2019)
2. Wang, X., et al.: An efficient method for automatic generation of linearly independent paths in white box testing. School of Software and Engineering, University of Electronic Science and Technology of China (2015)
3. McCabe, T.J.: A complexity measure. IEEE Trans. Softw. Eng. **SE-2**(4) (1976)
4. Alshaheen, H.S.: Finding shortest path in routing problem by using ant colony optimization. J. Univ. Thi-Qar **8**(3) (2013)
5. Vinod Chandra, S.S.: Smell detection agent based optimization algorithm. J. Inst. Eng. (India) Ser. B **97**(4), 431–436 (2016). https://doi.org/10.1007/s40031-014-0182-0
6. Ananthalakshmi Ammal, R., Sajimon, P.C., Vinodchandra, S.S.: Application of smell detection agent based algorithm for optimal path identification by SDN controllers. In: Tan, Y., Takagi, H., Shi, Y., Niu, B. (eds.) ICSI 2017. LNCS, vol. 10386, pp. 502–510. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61833-3_53
7. Srivastava, S., Kumar, S., Verma, A.K.: Optimal path sequencing in basis path testing. Int. J. Adv. Comput. Eng. Netw. (2013)
8. Yan, J., Zhang, J.: An efficient method to generate feasible paths for basis path testing. State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China 2008)
9. Salawudeen, A.T., et al.: From smell phenomenon to smell agent optimization (SAO): a feasibility study. In: Proceedings of ICGET (2018)
10. Kaur, S.: Shortest path finding algorithm using ant colony optimization. Int. J. Eng. Res. Technol. (IJERT) **2**(6), 317–326 (2013). ISSN 2278-0181
11. Cherkassky, B.V., Goldberg, A.V., Radzik, T.: Shortest paths algorithms: theory and experimental evaluation. Math. Program. **73**, 129–174 (1996). https://doi.org/10.1007/BF02592101
12. Donald, J.: A note on Dijikstra's shortest path algorithm. J. ACM **20**(3), 385–388 (1973)