Refinement of Data-Flow Testing using Ant Colony Algorithm

Abhay Kumar Srivastav [1], Supriya N S [2]
[1,2] Assistant Professor
[1,2] Department of MCA,MVJCE Bangalore-560067

Abstract : Search-based optimization techniques have been implemented to a variety of software engineering including test generation and cost estimation. Various search based test generation techniques have been used. These are the techniques used only on finding test data to satisfy control-flow or data-flow testing criteria. Currently, there are so many search-based optimization techniques have been implemented such as Ant Colony Optimization and Bees Colony Optimization. Ant Colony Optimization have been used only in control-flow testing of the programs. The aim of this to applying the Ant Colony Optimization algorithms in software data-flow testing. This technique uses the ant colony optimization to generate test data for satisfying the generated set of paths.

Keywords- *Data-flow testing, path-cover generation, ant colony optimization algorithms*

INTRODUCTION

There are various activities associated with software testing such as 1) finding path to cover criterion 2) test data generation to satisfy the path 3) test execution by using the test data 4) evaluation of test. There are many test-data generation techniques have been implemented.

Random test-data generation techniques is use to select inputs as a random data until useful inputs data are found [1, 2]. This technique some time fail to satisfy the requirements because information about the test requirements is not organised.

Symbolic test-data generation techniques is use to select symbolic values of variables to generate some algebraic equation for the constraints in the program and find the solution for these equation that satisfies a test requirement [3, 4]. Symbolic execution can't determine that symbolic values which are more potential values used for array as A[n] or pointer. Symbolic execution cannot work on floating point value as a inputs because the current constraint can't solve floating point values

Dynamic test-data generation techniques is use to collect data during the execution of the program to determine which test data come nearest to full fill the requirement. Then test inputs are incrementally changed until it is not going to full fill the requirement [5, 6]. Dynamic techniques can work only when if any local minima has occur because it depend on local search techniques such as gradient descent.

Search-based optimization techniques is use to software engineering activities such as cost estimation, next release problem and test-data generation [7]. Several search based test-data generation techniques have been implemented [8, 9, 10, 11, 12, 13]. Some of these techniques had focused on finding test data to satisfy a wide range of control-flow testing criteria (e.g., [8, 10, 11]) and the other techniques had concentrated on generating test-data for covering a number of data-flow testing criteria [12, 13, 9].

GENETIC ALGORITHMS is a employed search-based optimization technique in area of software testing [7].

There are some search-based optimizations techniques have been implemented like Ant Colony Optimization[14, 15], Particle Swarm Optimization [16], Bees Colony Optimization [17], and Artificial Immune System [18]. Ant Colony Optimization has been applied in the area of software testing in 2003 [19, 20]. Boerner and Gutjahr [19] described technique which use Ant Colony Optimization and a software called Markov Software usage for set of test paths for a software system, and McMinn and Holcombe [20] have given idea on the application of ACO as a supplementary optimization stage for finding sequences of transitional statements in generating test data for evolutionary testing. Srivastava and Rai [24] proposed an ant colony optimization based approach to test sequence generation for control-flow based software testing. The data-flow testing is important because it augments control-flow testing criteria and concentrates on how a variable is defined and used in the program, which could lead to more efficient and targeted test suites. The results of using ant colony optimization algorithms in software testing which obtained so far are preliminary and none of the reported results directly addresses the problem of test-data generation or path cover finding for data-flow based software testing.

This paper aims at employing the Ant Colony Optimization algorithms in the issue of software data-flow testing. The paper presents an ant colony optimization based on technique for generating set of optimal paths to cover all definition use associations in the program under test. This technique also uses the ant colony optimization algorithms to generate suite of test data for satisfying the generated set of paths.

II. BACKGROUND

The basic concepts and definitions are as follows

A. *Ant Colony Optimization*

Ant Colony Optimization is a population-based, general search technique for the solution of difficult combinatorial problems, which is inspired by the pheromone trail laying behaviour of real ant colonies. The Ant Colony Optimization technique is also known as Ant System [14] and it was applied to the travelling salesman problem. In Ant Colony Optimization , a set of software agents called artificial ants search for good solutions to a given optimization problem. To apply Ant Colony Optimization , the optimization problem is transformed into the problem of finding the best path on a weighted graph. The artificial ants (hereafter ants) incrementally build solutions by moving on the graph. The solution construction process is stochastic and is biased by a pheromone model, i,e a set of parameters associated with graph components such as nodes or edges whose values are modified at runtime by the ants. Figure 1 shows a generic ant colony algorithm.

Step 1: Initialization
- Initialize the pheromone trail

Step 2: Iteration
- For each Ant Repeat
- Solution construction using the current pheromone trail
- Evaluate the solution constructed
- Update the pheromone trail
- Until stopping criteria

Figure 1. *A generic ant colony algorithm*

The procedure to solve any optimization problem using Ant Colony Optimization is:

1) Represent the problem in the form of sets of components and transitions or by means of a weighted graph that is travelled by the ants to build solutions.

2) Appropriately define the meaning of the pheromone trail, i.e., the type of decision they bias. This is a crucial step in the implementation of an ACO algorithm. A good definition of the pheromone trails is not a trivial task and it typically requires insight into the problem being solved.

3) Appropriately define the heuristic preference to each decision that an ant has to take while constructing a solution, i.e., define the heuristic information associated to each component or transition. Notice that heuristic information is crucial for good performance if local search algorithms are not available or cannot be applied.

4) If possible, implement an efficient local search algorithm for the problem under consideration, because the results of many ACO applications to NP-hard combinatorial optimization problems show that the best performance is achieved when coupling ACO with local optimizers.

5) Choose a specific ACO algorithm and apply it to the problem being solved, taking the previous aspects into
consideration.

6) Tune the parameters of the ACO algorithm. A good starting point for parameter tuning is to use parameter settings that were found to be good when applying the ACO algorithm to similar problems or to a variety of other problems.

It should be clear that the above steps can only give a very rough guide to the implementation of ACO algorithms. In addition, the implementation is often an iterative process, where with some further insight into the problem and the behaviour of the algorithm; some initially taken choices need to be revised. Finally, we want to insist on the fact that probably the most important of these steps are the first four, because a poor choice at this stage typically can not be made up with pure parameter fine-tuning.

An ACO algorithm iteratively performs a loop containing
the following two basic procedures:
1) A procedure for specifying how the ants construct/modify solutions of the problem to be solved;
2) A procedure to update the pheromone trails.

The construction/modification of a solution is erformed in a probabilistic way. The probability of adding a new item to the current partial solution is given by a function that depends on a problem-dependent heuristic and on the amount of pheromone deposited by ants on the trail in the past. The updates in the pheromone trail are implemented as a function that depends on the rate of pheromone evaporation and on the quality of the produced solution.

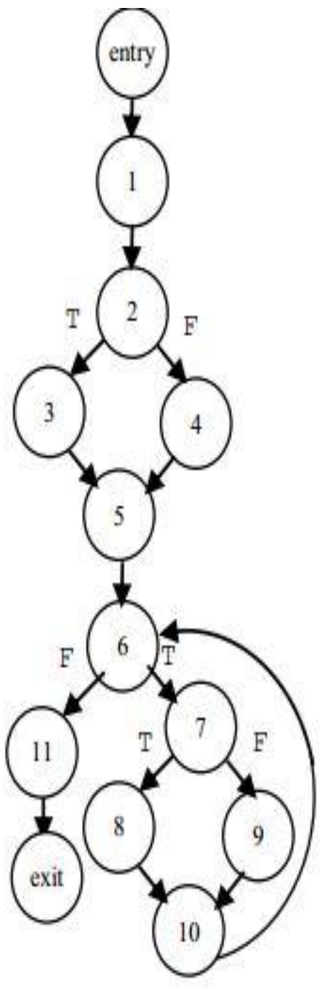
Data-flow analysis and testing

Typically, in structural testing strategies a program‘s structure is analyzed on the program flow-graph, i.e., an annotated directed graph which represents graphically the information needed to select the test cases.

A control flow graph is also a directed graph G=(V,E),with two distinguished nodes— a unique entry n0 and a unique exit n_k .

V is a set of nodes, where each node represents a statement, and E is a set of directed edges such as e = (n,m) is an ordered pair of adjacent nodes, called tail and head of e, respectively. Figure 2(a) gives an example program Program1 and figure 2(b) gives its control-flow graph.

```
#include<iostream.h>
 void main()
{
int i, j, k, n;
1  cin >> i >> j;
2  if(i < 6)
{
3  k = i;
}
else
{
4  k = j;
}
5  n = k;
6  while(n < 8)
{
7  if(j > k)
{
8  k = 2;
}
else
{
9  n = n + k + 7;
}
10  n = n + 1;
}
11 cout << i<< j << k;
}
```



(a) (b)

Figure 2. (a) An example program
(b) it is control-flow graph.

A path p in a control-flow graph is a finite number of nodes in sequence connected through edges e.g., 1→2→3→5 and 2→4.The question in software testing is how to select test cases with the aim of uncovering as many defects as possible. There are many activities normally associated with software testing such as 1) path-cover finding to cover a certain testing criterion 2) test data generation to satisfy the path cover, 3) test execution involving the use of test data and the software under test (SUT) and 4) evaluation of test results. Coverage criteria require that a set of entities of the program control-flow graph to be covered when the tests are executed. A set of complete paths (path cover) satisfy a criterion if it covers the set of entities associated with that criterion. Depending on the criterion selected, the entities to be covered may be derived from the program control flow or form the program data flow. Frankl and Weyuker in [28, 29] defined a family of popular control flow and data flow test coverage criteria.

Data-flow testing considers the possible interactions between definitions and uses of variables.

The variable in a program can be associated with the following events:

- A statement storing a value in a memory creates a definition of the variable.
- A statement accessing a value from the memory location of a variable is a use of the currently active definition of the variable. when the variable appears on the right-hand side of an assignment statement it is called as computational use (c-use), when the variable appears in the predicate of the conditional statement it is called as predicate use (p-use) [29].
- A statement delete the currently active definition of a variable if its value becomes unbound.

A path is definition clear path with respect to a variable if it contains no new definition of that variable. Data flow analysis determines the definitions of every variable in the program and the uses that might be affected by these definitions (i.e. the du-pairs). Such data flow relationships can berepresented by the following two sets:

- dcu(i), the set of all variable definition which have definition clear paths for node i;
- dpu(i, j), the set of all variable definitions for which they have definition clear paths for their p-uses at edge (i,j) [30]

  Using information concerning the location of variable definitions and uses, together with the basic static reach algorithm' [31], the sets dcu(i) and dpu(i, j) can be determined [30]. Tables 1 and 2 show samples of the du-pairs of Program1.

  TABLE V. LIST OF DCU-PAIRS FOR PROGRAM1.

Dcu	Variable	Def-node	Use-node	Killing nodes
1	A	1	3	None
2	C	8	9	3,4

TABLE VI. LIST OF DPU-PAIRS OF PROGRAM1

Dcu	Variable	Def-node	Use-node	Killing nodes
1	A	1	2,3	None
2	n	5	6,7	10

VI. CONCLUSION AND FUTURE WORK

To our knowledge, this paper is the first work using ACO in the issue of data-flow testing. This paper aims at employing the Ant Colony Optimization algorithms in the issue of software data-flow testing. The paper presented an ant colony optimization based approach for generating set of optimal paths to cover all definition-use associations (du-pairs) in the program under test. This approach uses also the ant colony optimization algorithms to generate suite of test-data for satisfying the generated set of paths. The ant colony algorithms are adopted to search the CFG and a model built on the program input domain in order to get the path cover and the test data that satisfies the selected path Our future work will focus on estimates the efficiency of ant colony optimization algorithms against genetic algorithms in this area. In addition, we will concentrate on solving the problem of constructing the searching model for the program with input variable of boolean and character type.

REFERENCES

1] H. D. Mills, M. D. Dyer, and R. C. Linger, ―Cleanroom software engineering,‖ IEEE Software, vol. 4, pp. 19-25, 1987.

[2] J. M. Voas, L. Morell, and K. W. Miller, ―Predicting where faults can hide from testing,‖ IEEE, vol. 8, pp. 41-48, 1991.

[3] W. E. Howden, ―Symbolic testing and the DISSECT symbolic evaluation system,‖ IEEE Transactions on Software Engineering, vol. 3, no. 4, 266-278, 1977.

[4] T. E. Lindquist, and J. R. Jenkins, ―Test-case generation with IOGen, IEEE Software,‖ vol. 5, no. 1, pp. 72-79, 1988.

[5] R. Ferguson and B. Korel, ―The chaining approach for software test data generation,‖ ACM TOSEM, vol. 5, pp. 63-86, 1996.
[6] B. Korel, ―Automated software test data generation,‖ IEEE Trans. on Software Engineering, vol. 16, pp. 870-879, 1990.
[7] M. Harman, "The current state and future of search based software engineering," Proc. of the International Conference on Future of Software Engineering (FOSE‘07), May 2007, pp. 342-357. IEEE Press.
[8] R. P. Pargas, M. J. Harrold, and R. R. Peck, ―Test data generation using genetic algorithms, Journal of Software Testing,‖ Verifications, and Reliability, vol. 9, pp. 263-282, 1999.
[9] A. S. Ghiduk, M. J. Harrold, M. R. Girgis, ―Using genetic algorithms to aid test-data generation for data flow coverage,‖ Proc. of 14th AsiaPacific Software Engineering Conference (APSEC 07), Dec. 2007, pp. 41-48. IEEE Press.
[10] C. C. Michael, G. E. McGraw, M. A. Schatz, ―Generating software test data by evolution,‖ IEEE Transactions on Software Engineering, vol.27, no.12, pp. 1085-1110, 2001.
[11] J. Wegener, A. Baresel, H. Sthamer, ―Evolutionary test environment for automatic structural testing,‖ Journal of Information and Software
Technology, vol. 43, pp. 841-854, 2001.
[12] L. Bottaci, ―A genetic algorithm fitness function for mutation testing,‖
Seminal: Software Engineering Using Metaheuristic Innovative Algorithms, 2001.
[13] M. R. Girgis, ―Automatic test data generation for data flow testing using a genetic algorithm,‖ Journal of Universal computer Science, vol. 11, no. 5, pp. 898-915, 2005.
[14] M. Dorigo, V. Maniezzo, and A. Colorni, ―Ant System: Optimization by a Colony of Cooperating Agents,‖ IEEE Transactions on Systems, Man, and Cybernetics-Part B Cybernetics, vol. 26, no. 1, pp. 29-41, 1996.
[15] C. Blum, ―Ant colony optimization: introduction and hybridizations Proc. of 7th International Conference on Hybrid Intelligent Systems (HIS‘07), Sept. 2007, pp. 24-29. IEEE Press.
[16] X. Zhang, H. Meng, and L. Jiao, ―Intelligent particle swarm optimization in multiobjective optimization, Proc. of the 2005 IEEE Congress on Evolutionary Computation, Vo. 1, pp. 714-719. IEEE Press.
[17] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi ―The bees algorithm – A novel tool for complex optimisation problems Proc. of Innovative Production Machines and Systems Conference (IPROMS‘06), 2006, pp.454-461
[18] A. Bouchachia, ―An immune genetic algorithm for software test data
generation‖ Proc. of 7th International Conference on Hybrid Intelligent
Systems (HIS‘07), Sept. 2007, pp. 84-89. IEEE Press.
[19] Doerner, K., Gutjahr, W. J., ―Extracting Test Sequences from a Markov Software Usage Model by ACO, LNCS, Vol. 2724, pp. 2465-2476,Springer Verlag, 2003.
[20] McMinn, P., Holcombe, M., ―The State Problem for Evolutionary Testing, Proc. GECCO 2003, LNCS Vol. 2724, pp. 2488-2500,Springer Verlag, 2003