An Efficient Technique for Software Coverage using Metaheuristic Algorithm

Nupur bajaj
Student,M.Tech(CSE)
JCDV,Sirsa

Meha Khera
Student,M.Tech(CSE)
TITS,Bhiwani

Vikram
A.P. in IT Deptt.
JCDV,Sirsa

Vijay Chahar
A.P. in CSE Deptt.
JCDV,Sirsa

ABSTRACT

Software Testing is one of the most important parts of the software development lifecycle. Functional and structural testing are the most widely used testing methods to test softwares. Testing effectiveness can be achieved by the State Transition Testing (STT) which is commonly used for carrying out functional testing of software systems. The tester is required to test all the possible transitions in the system under built. Structural testing relies on identifying effective paths of the code. Aim of the current paper is to present a strategy by applying ant colony optimization technique, for the generation of test sequences for state transitions of the system as well as path generation for the Control Flow Graph of the software code using the basic property and behavior of the ants. This Proposed strategy gives maximum software coverage with minimal redundancy.

Keywords

Software Testing, Ant Colony Optimization (ACO), State Transition Testing (STT) Control Flow Graph (CFG)

1 INTRODUCTION

Software Engineering [1] is an engineering discipline which focuses on developing high-quality software systems which are very cost effective. It is a profession wherein designing, implementation, and modification of software are involved. Software Development Life Cycle [1] (SDLC) or systems development life cycle is a process of developing a system which involves different steps like investigation, analysis, design, implementation and maintenance. Software testing is the primary technique, since always, used to gain consumers' confidence in the system. Software testing is a labour intensive and very expensive task. It accounts almost 50 % of software development life cycle [2]. Selection of the test data for testing software is the crucial part of software testing. The appropriate amount of test data can minimize unnecessary execution time of the testing Process.

However, thorough testing is often unfeasible due to the potentially infinite execution space or high cost within tight budget limitations. Many testing tools and methodologies have been emerging in the field of software testing. Code coverage

analysis is one such methodology which helps in discovering the instructions in a software program that have been executed during a test run and also helps in discovering how the testing can be further improved to cover more number of instructions during testing of a software program [2]. Measuring the code coverage is important for testing purpose and also for validation of the code during both the development and porting it to new platforms. Code coverage [3] analysis helps in finding dead code that was written considering it would be useful in near future, but no longer is necessary. When such code is discovered, it has been removed from the software, hence reducing the memory requirements for storing the program and freeing up space for other useful information. The main goal of any Software organization [1] is to provide a good quality software product to the customer within the estimated budget. In order to achieve this goal, the key concept used by any organization is software coverage thus gaining the confidence of the clients.

2 CODE COVERAGE ANALYSIS

Code coverage analysis is one such methodology which helps in discovering the instructions in a software program that have been executed during a test run and also helps in discovering how the testing can be further improved to cover more number of instructions during testing of a software program. Measuring the code coverage is important for testing purpose and also for validation of the code during both the development and porting it to new platforms. Code coverage analysis is the process of finding areas of a program not exercised by a set of test cases, creating additional test cases to increase coverage, and Determining a quantitative measure of code coverage, which is an indirect measure of quality. An optional aspect of code coverage analysis is identifying redundant test cases that do not increase coverage.

A code coverage analyzer automates this process. Coverage analysis is to assure quality of some set of tests, not the quality of the actual product. We do not generally use a coverage analyzer when running the set of tests. Coverage analysis is one of many testing techniques; we should not rely on it alone. Code coverage analysis is sometimes called test coverage analysis. The two terms are synonymous. The academic world more often uses the term "test coverage" while practitioners more often use "code coverage". Likewise, a coverage analyzer is sometimes called a coverage monitor.

3 STRUCTURAL TESTING AND FUNCTIONAL TESTING

Code coverage analysis is a structural testing technique (glass box testing and white box testing). Structural testing compares test program behavior against the apparent intention of the source code. This contrasts with functional testing (black-box testing), which compares test program behavior against a requirements specification. Structural testing examines how the program works, taking into account possible pitfalls in the structure and logic. Functional testing examines what the program accomplishes, without regard to how it works internally.

Structural testing is also called path testing since we choose test cases that cause paths to be taken through the structure of the program. At first glance, structural testing seems unsafe.

Structural testing cannot find errors of omission. However, requirements specifications sometimes do not exist, and are rarely complete. This is especially true near the end of the product development time line when the requirements specification is updated less frequently and the product itself begins to take over the role of the specification. The difference between functional and structural testing blurs near release time.

Code coverage analysis helps in finding dead code that was written considering it would be useful in near future, but no longer is necessary. When such code is discovered, it has been removed from the software, hence reducing the memory requirements for storing the program and freeing up space for other useful information. The main goal of any Software organization is to provide a good quality software product to the customer within the estimated budget. In order to achieve this goal, the key concept used by any organization is software coverage thus gaining the confidence of the clients.

4 COMBINATORIAL OPTIMIZATION ALGORITHMS

4.1 ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) is a paradigm for designing meta-heuristic algorithms for combinatorial optimization problems. A Meta heuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. In other words, a Meta heuristic is a general-purpose algorithmic framework that can be applied to different optimization problems with relatively few modifications. Examples of meta-heuristics include simulated annealing, tabu search, iterated local search, evolutionary computation, and ant colony optimization. Meta heuristic algorithms are algorithms which, in order to escape from local optima, drive some basic heuristic: either a constructive heuristic starting from a null solution and adding elements to build a good complete one, or a local search heuristic starting from a complete solution and iteratively modifying some of its elements in order to achieve a better one. The first algorithm which can be classified within this framework was presented in 1991 by Marco Dorigo with his PHD thesis "Optimization, learning, and Natural Algorithms", modeling the way real ants solve problems using pheromones. The main idea is that the self-organizing principles which allow the highly coordinated behavior of real ants can be exploited to coordinate populations of artificial agents that collaborate to solve computational problems. Several different aspects of the behavior of ant colonies have inspired different kinds of ant algorithms. Examples are foraging, division of labor, brood sorting, and cooperative transport. In all these examples, ants coordinate their activities via stigmergy, a form of indirect communication mediated by modifications of the environment. For example, a for-aging ant deposits a chemical on the ground which increases the probability that other ants will follow the same path. Real ants are capable of finding the shortest path from a food source to their nest. While walking ants deposit pheromone on the ground and follow pheromone previously deposited by other ants, the essential trait of ACO algorithms is the combination of a priori information about the structure of a promising solution with a posteriori information about the structure of previously obtained good solutions. In ACO, a number of artificial ants build solutions to an optimization problem and exchange information on their quality via a communication scheme that is reminiscent of the one adopted by real ants.

4.1.1The ACO system contains two rules:

1. Local pheromone update rule, which applied whilst constructing solutions.

2. Global pheromone updating rule, which applied after all ants construct a solution

Furthermore, an ACO algorithm includes two more mechanisms: trail evaporation and, optionally, daemon actions. Trail evaporation decreases all trail values over time, in order to avoid unlimited accumulation of trails over some component. Daemon actions can be used to implement centralized actions which cannot be performed by single ants, such as the invocation of a local optimization procedure, or the update of global information to be used to decide whether to bias the search process from a non-local perspective.

ACO: Path Construction

• When ant k is located at a node vi the probability pjk of Choosing vj as the next node is:

$$p_j^k = \begin{cases} \dfrac{\tau_{ij}^{\ \alpha} \cdot \eta_{ij}^{\ \beta}}{\sum_{m \in N_i^k} \tau_{im}^{\ \alpha} \cdot \eta_{im}^{\ \beta}} & if\ j \in N_i \\ 0 & if\ j \notin N_i \end{cases}$$

With:
• Ni: the set of nodes that ant k can reach from v (tabu list)
• hij: the heuristic desirability for choosing edge (i,j)
• tij: the amount of pheromone on edge (i,j)
• a and b : relative influence of heuristics vs. pheromone

4.1.2ACO: Pheromone updates (1)

The pheromone on each edge is updated as:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}$$ With:

• r : the evaporation rate of the 'old' pheromone

• Δtij : the 'new' pheromone that is deposited by all ants on edge (i,j) calculated as:

$$\Delta\tau_{ij} = \sum_{k=0}^{m} \Delta\tau_{ij}^{k}$$

ACO: Pheromone updates (2)

The pheromone that is deposited on edge (i,j) by ant k is calculated as:

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k & if\ (i,j) \in T_k \\ 0 & otherwise \end{cases}$$

With:
• Q : a heuristic parameter
• Tk: the path traversed by ant k

Table 1: Comparison of Tools

TOOL	Paths Generated
Old	start,1,2,3,4,end start,1,2,3,4,5,6,7,8,9,end start,1,2,3,4,5,6,10,11,4,5,6,7,8,9,end start,1,2,3,4,5,6,10,12,4,5,6,7,8,9,end
New	start,1,2,3,4,end start,1,2,3,4,5,6,10,11,4,end start,1,2,3,4,5,6,10,12,4,5,6,7,8,9,end

• Lk : the length Tk calculated as the sum of the lengths of all the edges of Tk.

5 DISCUSSION

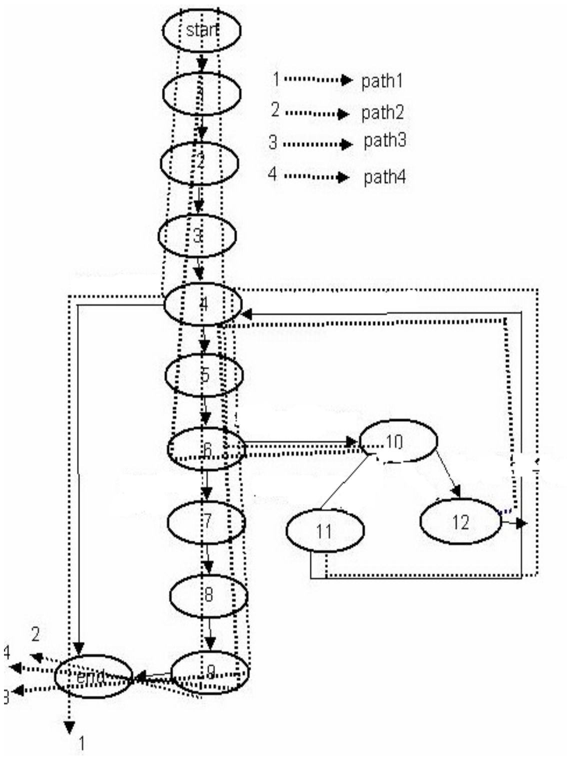

Figure 1: Control Flow Graph

In the above graph different numbers of paths are generated, that shows the code coverage in it. Four different paths are generated when the tool is applied, but from our strategy only 3 different paths are generated giving the total coverage with minimum redundancy.
Comparison is given in Table 2 as follows:

Therefore, it shows the technique provides minimal number of repetitions producing maximum coverage.

6 CONCLUSION

This paper proposes a technique for test sequence generation for state based testing and optimal path generation for structural testing using ant colony optimization. The result that is produced by applying proposed method is very encouraging. To model the system state chart diagram and CFG are taken and the algorithm is applied over them. After successful execution of algorithm, it shows path sequence which gives maximum coverage and minimum redundancy. This may be very useful and helpful for the testers in the software industry. A number of extensions and applications of this model may be possible by using the different meta-heuristic techniques.

7 REFERENCES

[1] Ian Sommerville,” Software Engineering”, eight Edition, Pearson Edition, 2009.
[2] Aditya P. Mathur “Foundation of Software Testing”, First Edition, Pearson Education, 2007.
[3] Myers, G., The Art of Software Testing. 2 edition: John Wiley & Son. Inc. 234 pages, 2004.
[4] Marco Dorigoa and Thomas Stutzle, “Ant colony optimization, The Knowledge Engineering Review", Cambridge University Press New York, NY, USA. , Volume 20, Pp: 92 – 93,2005.
[5] DI´AZ E., TUYA J., BLANCO R.: ‘Automated software testing using a meta-heuristic technique based on tabu search’. ASE-2003.
[6] Marco Dorigo and Thomas Stutzle, “Ant Colony Optimization”, phi publishers, 2005.
[7] S.D.Shtovba,”AntAlgorithms: Theory and Applications”,programming and computing software,vol 31,issue 4,pp167-178, 2005.
[8] Praveen Ranjan Srivastava, K M Baby, “An Approach of Optimal Path Generation using Ant Colony Optimization”, ISBN-978-1-4244-4546-2, pages 1-6, IEEE-TENCON, Singapore, 2009.