International Journal of Engineering & Technology, 7 (4) (2018) 2743-2745

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.17565
Research paper

Ant Colony Optimization Based Test Case Selection for Component Based Software

Palak Palak [1] *, Preeti Gulia [1]

[1] *Department of Computer Science and Applications, MDU, Haryana, India*
**Corresponding author E-mail: palak.aug6@gmail.com*

Abstract

Reusability is one of the prime aspects of high quality software. Based on the concept of reusing the previous effort, Component based software engineering is a widely evolving software development paradigm that sets new challenges for testing team. The third party components need to be selected and assembled in development framework. Components interact with each other for various services and the interface between them can prove as the point of failure. As exhaustive testing of all interaction sequences is not possible, there is need for automated test case reduction and prioritization techniques to increase the efficiency of testing process. Ant Colony Optimization (ACO), a nature inspired optimization technique has wide range of applications in the field of software engineering. This paper presents an ACO based technique for test case selection for interaction testing of reusable software components.

Keywords: *Ant Colony Optimization; Components; Software Testing; Test case selection*.

1. Introduction

Component-based software has potentially attracted many researchers and the interest in this area is growing at fast pace due to its potential in managing the increasing complexity of software systems using a highly modular approach. A marketplace for software components is emerging. The prime idea behind Component Based Software Engineering (CBSE) is reusability of the previous effort done to build components. CBSE helps the developer to reduce overall cost by reusing the basic components developed earlier for similar products or as stand- alone. The development time is also benefitted. The main objectives of using CBSE include productivity increasing and cost saving besides many other inherent benefits [1]. CBSE is an evolving field and accepted by a large number of organizations incrementally. It also helps in achieving project repeatability thus improving overall process maturity. A component is an executable reusable software implementation that is usually black box and interacts with other components using well defined interfaces (Szyperski, 2002). The efficiency of CBSE depends on the successful interactions between different components. A component may require some services from other components and may in turn provide some services too. This forms a highly interactive modular system by assembling various components in common development architecture. If a single interface between components fails, it can lead to complete system failure in long run. This paper focuses on the latest ongoing research in the field of CBSE testing using search based techniques especially Ant Colony Optimization (ACO) based test case selection and test suite reduction. A new model for CBSE test case selection and reduction is proposed that is inspired from behavior of natural ants in the search of food.

2. Related work

Use of ACO in the field of software testing has always attracted researchers which resulted to the rich literature available worldwide. We collected some of the relevant research publications over recent years that shows the impact and efficiency of ACO and other swarm based techniques in the field of software testing for optimizing its various phases. N. Sethi et. al. in [2] utilized ACO for reducing test suite in regression testing. S. Yang et. al. in [3] presented a modified ACO approach for automated software testing. They presented new local pheromone update coefficient and compared the results with traditional random and genetic algorithm based techniques. Improved efficiency and test coverage is claimed in their work. In [4], G. Chen et. al. worked on specification based problems where test cases are inter-dependent and need to be executed in a specified sequence. They converted the problem into sequential ordering problem and applied ACO to solve the same. Besides ACO, some other meta-heuristic, search based approaches are mushrooming over last few years. R. Malhotra et. al. in [5] compared three such approaches namely Genetic Algorithm (GA) and Artificial Bee Colony (ABC) with ACO in context of automated test data generation for C ++ sample programs. They found that ACO suffers from overhead due to pheromone updating process. T. Noguchi et. al. in [6] proposed to prioritize test cases for black box testing on a new product using the test execution history collected from a similar prior product and the ACO. Mukesh Mann et .al. [7] also utilized ACO for optimizing the phases of software testing by applying it to control flow graphs and decision to decision graphs. Authors in [8] proposed an optimized test case prioritization technique using Ant Colony Optimization (ACO) to reduce the cost, effort and time taken to perform regression testing. They used APFD (Average Percentage of Fault Detected) metric to compare their results with non-prioritized test cases and resulted in a significant im-

provement in fault detection. Similar effort is done in [9] by C. Lu et. al. for statement coverage based test case prioritization using tree shaped function for pheromone updates. Authors in [10] used evolutionary strategies for structural test data generation. They gave a new definition to pheromone to increase branch coverage and defined 2 fitness functions i.e. one for branch coverage and second for branch complexity. They claim to have better performance than other ant based testing techniques.

The myriad publication database available worldwide shows the importance of swarm based techniques that too in the field of software testing. The above section only summarizes a few of the important researches over recent years to show the glimpse of problem solving abilities of ACO based techniques.

3. Component Based Software Engineering

Component Based Software Engineering (CBSE) evolved back in late 1980s. Defying the fact to develop everything from scratch, CBSE exploits the previous effort in developing interactive reusable components. Components with specific interfaces can be purchased from various suppliers and used as building blocks to develop complex products. Along with the simple idea there come many software implications to make the components compatible to each other. Components express themselves through interfaces. Interfaces are categorized as provider interfaces (i.e. provides some service to other component) and receiver interfaces (i.e. receives some services from other components). Designing and implementing appropriate and compatible interfaces is the main challenge that is faced by CBSE practitioners. Components are stored in component repository. First of all the component repository is searched for necessary and required components that meets the requirement specification. Domain specific components meeting the requirements and system design should be selected and picked up from component repository [11]. If component functionality is not available, then there may be a need to create a component from scratch. After component selection, it may also be possible that components need to be adapted according to current platform and system requirements. The next step is to deploy the components by assembling them into specified architecture using glue code. Once the product is delivered there may be chances of component replacement at later stages of maintenance. From time to time new versions of components are launched that creates the need to update the application. It may involve component replacement or addition of new component (i.e. new functionality towards the system). Component based applications are highly modular and scalable.

3.1. Factors Influencing Component Performance

Components may be developed at different site and used at other. A component might not perform as expected by the developer side due to various factors influencing its performance. Fig. 1 shows various factors that affects overall component performance. These are: Component implementation, required services, Resource Contention, Usage Profile and Deployment platform (Heiko Koziolek, 2009).

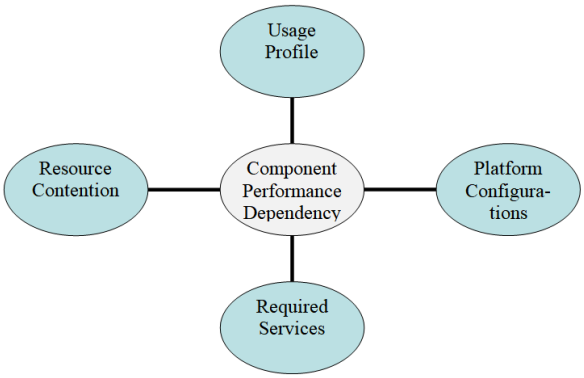

Fig. 1: Factors Influencing Component Performance.

A component provides functionality specified in requirements but can be implemented in different ways in terms of algorithm that influences its performance. Component performance can be greatly affected by parameters or configurations of the platform they are deployed at. The user is not known to the internal implementation as components provide a black box view. The overall execution time of a component is highly influenced by the services it requires from other components. Each component when executed has some resource requirements such as bandwidth, memory etc. which effects the performance of components. The execution time of a service within a component is also dependent on the input parameters. Different components behave differently based on the value and number on parameters passed to it through its receiver interfaces.

3.2. CBSE Testing Challenges

CBSE offers inherent challenges to the application developers due to heterogeneous environment at various parties involved in component development. Testing such complex applications built from components is a cumbersome task due to lack of information about external components but indeed a very important step to deliver highly reliable application. Myriad research is going on for automation of testing to achieve effectiveness. Systematic and planned software testing includes three main activities. These are test data generation, test execution by running test suite from valid and invalid input domain to the system under test and evaluation of test results. Exhaustive testing is not possible due to combinatorial possibilities of test cases. So it becomes an apparent challenge for the testing team to generate smaller test suite that is capable of identifying comparatively more number of application errors. Moreover the test cases need to be prioritized by deciding the preference of one test case over other which reduces overall testing efforts and cost and affects testing effectiveness up to a great extent. Important CBSE testing challenges are listed below:

1) Requirements Confirmation: A component is built by keeping certain set of functionalities to be performed. When it is assembled in a system, it may not confirm to all the requirements that the particular system demands. Moreover the requirements may change from time to time as the software evolves.
2) CBSE Test Case Selection: Effective test suite is selected from all possible input domains with the intent of finding the faults. But in case of CBSE, test case selection becomes a challenging task as different components behave differently in combination with other components.
3) Unavailability of Component Test Driver and Stub: Applications involving components are highly complex and interactive. A need for test drivers arises during unit testing when it requires services from other functional component that is not available at that time (Jhao, 2003).
4) Black Box View: Components represent black box and the internals are not known to the testing team. So coverage testing becomes a challenge.

5) Platform and Application Level Changes: When introducing changes on the platform and application level such as updating of operating system, updating of other components, changes in the application, etc., there is a risk that the change may lead to component incompatibility leading to system failure (I. Crnkovic, 2003).

4. Ant Colony Optimization Revisited

Ant colony optimization (ACO) is a soft computing technique to solve various optimization problems. It is a meta-heuristic technique that can be applied to generate promising sub optimal solutions for the problems having combinatorial possibilities and where exhaustive search is not possible. ACO is inspired by behavior of real ants. Real ants while searching for food travels a path from their nest to food source leaving behind a chemical substance called pheromone trail. The amount of pheromone on a certain path keeps on changing as more ants on that path adds more pheromone and evaporation reduces its amount by time. Other ants communicate by sensing pheromone on their paths and it is likely that the ants will choose the path with maximum pheromone level as shown in figure 2. The ant on the shortest path (i.e. P2 in this case) is likely to return its food nest in less amount of time and subsequently intensify the amount of pheromone during every back and forth motion. This path will further attract more ants due to higher pheromone deposition. Since its inception (M. Dorigo et.al., 1999), ACO has been potentially used in various domains of engineering and for solving complex numerical problems.

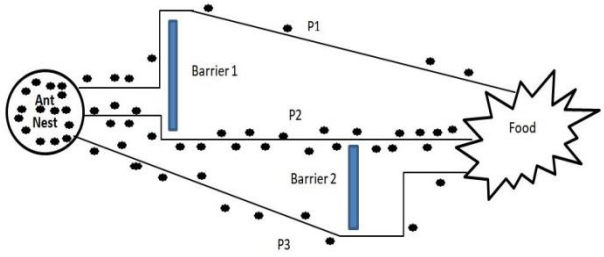

Fig. 2: ACO Example with Three Possible Paths.

To apply ACO, the problem is first converted into a weighted graph to find the shortest path. The shortest path will be discovered through coordination between the ants by sensing pheromone and pheromone evaporation process. Pheromone gets evaporated over time. Pheromone deposition and evaporation rates are important factors in deciding the overall performance of the algorithm. Longer the path, more the pheromone evaporation and lesser the residual pheromones is. As the process converges after much iteration, more and more ants will follow the common shortest path and no more pheromone levels will be updated. The following steps are used for solving the optimization problem using ACO:

1) For applying ACO, convert the given problem as a set of components arranged in a weighted graph showing interactions between them.
2) Initialize the pheromone trail by defining initial weights and define update pheromone function.
3) Define probabilistic next node selection criteria and feed random artificial ants to the system.
4) Each ant propagates and performs two operations: a) calculate probability of next node to be visited; b) update pheromone of the chosen path.
5) Iterate the above steps until stopping criteria is met.

5. Proposed Model

As stated earlier, CBSE consists of highly interactive modular system by assembling various components in common development architecture. If a single interface between components fails, it may lead to complete system failure in long run. This paper aims at utilizing ACO for reducing the test suite for detecting reusable component interaction failures. The algorithm of proposed model is given below:

Input: Initial Pheromone Trail, Fault test case matrix, Component Diagram
Step 1: Convert Component diagram to Control Flow Graph
Step 2: Virtual Ants at each node calculate probability of next node to be visited by using ACO probability function.
Step 3: Select the best next node on the basis of probability calculated in step 2 and Update Pheromone level of nodes.
Step 4: Add that node in Best path.
Step 5: Go to Step 2 until maximum number of iterations are met or length of path exceeds threshold limit.
Output: Best paths containing reduced set of test cases.

6. Conclusion

Complexity and demand of software are increasing day by day. CBSE provides a modular and fast solution to deliver highly complex software at reduced cost and time. But it introduces new testing challenges due to black box view of each component. This paper presents CBSE performance influencing factors; it's testing challenges and a new model for CBSE testing using well known nature inspired technique ACO is proposed. It is found that the interaction among components can be tested by converting component diagram to control flow graphs and utilizing ACO to select promising and reduced set of test cases for detecting faults. In future, the proposed technique will be implemented by taking various CBSE based projects to reduce the interaction failures among components.

References

[1] T. Vale, I. Crnkovic, E. Santana, D. Almeida, P. Anselmo, S. Neto, Y. Cerqueira, S. Romero, and D. L. Meira, "Twenty-eight years of component-based software engineering," *J. Syst. Softw.*, vol. 111, pp. 128–148, 2016. https://doi.org/10.1016/j.jss.2015.09.019.

[2] N. Sethi, S. Rani, and P. Singh, "Ants Optimization for Minimal Test Case Selection and Prioritization as to Reduce the Cost of Regression Testing," *Int. J. Comput. Appl.*, vol. 100, no. 17, pp. 48–54, 2014.

[3] S. Yang, T. Man, and J. Xu, "Improved ant algorithms for software testing cases generation," *Sci. World J.*, vol. 2014, 2014.

[4] G. Y.-H. Chen and P.-Q. Wang, "Test Case Prioritization in a Specification-based Testing Environment," *J. Softw.*, vol. 9, no. 8, pp. 2056–2065, 2014.

[5] R. Malhotra, "Comparison of Search based Techniques for Automated Test Data Generation," *Int. J. Comput. Appl.*, vol. 95, no. 23, pp. 4–8, 2014.

[6] T. Noguchi, H. Washizaki, and Y. Fukazawa, "History-Based Test Case Prioritization for Black Box Testing Using Ant Colony Optimization," *2015 IEEE 8th*, vol. 1, pp. 2–3, 2015.

[7] M. Mann and O. P. Sangwan, "Generating optimization and prioritizing optimal paths using ant colony," vol. 5, no. 1, pp. 1–15, 2015.

[8] A. Ansari, A. Khan, A. Khan, and K. Mukadam, "Optimized Regression Test Using Test Case Prioritization," *Procedia Comput. Sci.*, vol. 79, pp. 152–160, 2016. https://doi.org/10.1016/j.procs.2016.03.020.

[9] C. Lu, J. Zhong, and C. Author, "An Efficient Ant Colony System For Coverage Based Test Case Prioritization," in *GECCO '18 Companion*, 2018, pp. 91–92.

[10] H. Sharifipour, M. Shakeri, and H. Haghighi, "Structural test data generation using a memetic ant colony optimization based on evolution strategies," *Swarm Evol. Comput.*, vol. 40, pp. 76–91, 2018. https://doi.org/10.1016/j.swevo.2017.12.009.

[11] F. Khan, M. Tahir, M. Babar, F. Arif, and S. Khan, "Framework for Better Reusability in Component Based Software Engineering," *J. Appl. Environ. Biol. Sci.*, vol. 6, no. 4S, pp. 77–81, 2016.