An Ant Colony Based Algorithm for Test Case Prioritization with Precedence

Camila Loiola Brito Maia[1] , Thiago do Nascimento Ferreira[1], Fabrício Gomes de Freitas[1], Jerffeson Teixeira deSouza[1]

[1] Optimization in Software Engineering Group (GOES.UECE)
State University of Ceará (UECE)
60740-903, Fortaleza-CE, Brazil
camila.maia@gmail.com, thiagonascimento.uece@gmail.com, fabriciogf.uece@gmail.com, jeff@larces.uece.br

Abstract. Test case prioritization is a difficult problem of Software Engineering, since several factors may be considered in order to find the best order for test cases. Search-based techniques have been applied to find solutions for test case prioritization problem. Some of these works apply Ant Colony based algorithms, but the precedence of test cases was not considered. We propose an Ant Colony Optimization based algorithm to prioritize test cases with precedence. Each ant builds a solution, and when it is necessary to choose a new vertex (test case), only allowed test cases are seen by the ant, implementing the precedence constraint of the problem.

Keywords: Ant Colony System, Test Case Prioritization, Precedence

1 Introduction

When a system is maintained, features that had been previously tested may be impacted by the changes made to the system. In this case, in order to assure that such changes did not introduce new faults, regression testing is performed, by rerunning previously executed tests.

In this scenario, the test case prioritization, which means to prioritize test cases according to some criterion [1], becomes very important, since it is usually impractical to perform exhaustive tests [2]. Thus, if one cannot run all test cases, the most significant ones can be executed first.

Like other software engineering problems, the test case prioritization problem has been studied through a search-based perspective, in the research field known as Search-Based Software Engineering [3]. Metaheuristics such as Reactive GRASP [4], Genetic Algorithm [5] and Simulated Annealing [6] have been applied in several approaches to test case prioritization problem.

Additionally, an algorithm based on the Ant Colony System metaheuristic [7] has also been applied to this problem, as in [8] and [9]. In those cases, however, the precedence among test cases was not considered.

Ant Colony is a metaheuristic based on the behavior of ants while seeking food. There is a collective exchange of information, through the pheromone.

This paper proposes an algorithm based on ACO to the test case prioritization problem, considering the precedence of test cases.

The remaining of the paper is organized as follows: Section 2 presents related works, while Section 3 explains the test case prioritization problem as it is tacked in this work. In Section 4, the proposed algorithm is described. Finally, in Section 5, we present the conclusions and future works.

2 Related Works

There are several works that address the test case prioritization problem with search-based algorithms, but there are few studies related to the application of ACO to this problem. Due to space constraints, we will cite only four studies that apply metaheuristics in solving this problem, two of them based on ACO.

The authors of [8] proposed a technique for test case prioritization based on genetic algorithm that reorders the test suite considering two objectives: time constraints for testing and code coverage. The objective function implements Block Average Percentage Coverage (APBC) metric. The proposed technique performed better than other existing techniques for a case study. For another case study, this did not happen because the test cases of this case study were more interchangeable.

In [9], the authors compare five algorithms for the problem of prioritization of test cases: Algorithm Greedy, Additional Greedy, 2-Optimal, Hill Climbing and Genetic Algorithm. They considered the following coverage metrics separately (three single-objective approaches): Average Percentage Block Coverage (APBC), Average Percentage Decision Coverage (APDC) and Average Percentage Coverage Statement (APSC). For small programs, the genetic algorithm was better than the others. For large programs, the two best algorithms were Additional Greedy and 2-Optimal.

In [10], the authors present an approach to the test case prioritization problem based on run time and fault detection rate. In this approach, there is a time constraint, which implies that this approach relates to the test case selection problem, not the test case prioritization problem. The authors consider a set of test cases and a set of known faults, where each test case covers one or more faults. The proposed ACO-based algorithm considers *n* ants, where *n* is the number of test cases, and the edges of the graph to be covered are randomly chosen by the ants among ones having maximum pheromone. The initial vertex is chosen randomly. The results obtained by the proposed approach are similar to the technique called Optimal Fault Coverage, and superior to random order, reverse order and no order techniques.

The approach proposed in [11] also considers the runtime information and fault detection of test cases. The number of ants is the number of test cases, and each ant is placed at a different vertex (test case) at the beginning of the execution. After initialization, the next vertex is chosen probabilistically according to the heuristic function (maximize the number of faults detected by the test case and minimize the execution time) and pheromone previously deposited by ants. The selection process of vertices is performed until all faults are covered by test cases already in the test suite.

After updating the pheromone and choosing the best solution, the algorithm checks the runtime restriction. If the solution is not valid, the whole process is performed again. The proposed approach has reduced by 62.5% the size of the test suites to be performed, because it was considered that a fault can be found by one or more test cases. The proposed approach was not compared with other approaches.

There are two common features to the last two approaches described above: the number of ants and the fact that they do not consider the precedence among the test cases. When considering the number of ants as the number of test cases, the execution of the algorithm may be slowly as the number of test cases increases, since there are more ants sharing pheromone information. Moreover, the approaches do not address the precedence among test cases, which is common in real world applications.

3 Problem Definition

This section formally describes the test case prioritization problem in the way it is considered in this work. Let *R* be the set of requirements for the system. The set *R* contains *N* requirements and each requirement has the following attributes:

• $importance_i$: The importance of the requirement *i*, associated by customers, representing its importance to the business.

• $volatility_i$: Volatility of the requirement *i*, representing the number of times the requirement has been changed.

Also consider *C* as the set of all test cases of the system. *C* has *M* test cases. Each test case has the following attributes:

• $precedence_j$: Test case that should be performed before the test case *j*. This approach considers that a test case has at most one predecessor.

• $coverage_j$: Coverage of test case *j*. The coverage of a test case is the set of requirements that are tested by this test case. In this approach, a requirement can be tested by one or more test cases, but a test case can test only one requirement.

• $executionTime_j$: Estimated time to manually run test case *j*.

• $score_j$: Represents the value of the test case *j*, based on importance and volatility of requirements covered by this test case, given by the following formula:

$$score_j = \sum_{\forall\, requirement\ _i\ \in coverage_j} \left((importance_i * weight1) + (volatility_i * weight2)\right),$$

where *weight1* and *weight2* are inputs and represent, respectively, the importance and volatility weights. Thus, the test case prioritization problem can be mathematically formulated as follows:

$$Maximize \sum_{j=1}^{M} \left(\frac{score_j}{executionTime_j} * P_j\right)$$

Subject to:

$$\forall\, t_{j1}, t_{j2} \in C, \quad \left(precedence_{t_{j2}} = t_{j1}\right) \rightarrow (q_{t_{j1}} < q_{t_{j2}})$$

Where $P_j = M - q_j + 1$, and q_j is the position of test case j in the ordered test suite. The above restriction states that if a test case t_{j1} is precedent of a test case t_{j2}, t_{j1} must be positioned before t_{j2} on the test suite.

4 Proposed Approach

This section describes the proposed algorithm, based on ant colony optimization, for the problem of prioritization of test cases with precedence.

4.1 Overview

To apply the proposed algorithm to the problem of prioritizing test cases with precedence, one must make some modeling.

Each test case of the system represents a vertex of the directed graph to be generated for the problem, i.e., the set V of graph $G = (V, E)$ has M elements, where M is the number of test cases. All vertices are connected with all others, generating the set E of edges. The pheromone information will be updated in graph G.

For this approach, each ant i has the following information associated:

- nextNode: The next node to be visited.
- visitedNodes: Set of vertices (test cases) already visited by the ant.
- allowedNodes: Set of vertices allowed for the next move of the ant. This set will be updated at each movement of the ant. The allowed test cases are represented by test cases that have no precedents or that have had their precedents already added to visitedNodes. Each time the ants reach the next node, its allowedNodes set for next movement is updated.

The heuristic function, used to choose the next vertex of solution is given by the following equation:

$$\frac{score_j}{executionTime_j}$$

Thus, the next vertex is chosen based on its score and execution time.

4.2 Algorithm Description

For the proposed approach, which is described in Fig. 1, each ant will build a complete solution, i.e., a suite of test cases ordered for execution. In each iteration, the ants will perform their activities at the same time, sharing pheromone information.

The algorithm starts with a global initialization, which basically reads precedence information and generates graph G, and ants initialization, with initializes all ants' information, previously described. The function *FIND_INITIAL_NODE()* chooses the initial vertex for the ant, and can be performed in three different ways: random choice, russian roulette or binary tournament. The last two consider the heuristic

function as the basis for the choice of vertices. The algorithm can be set to run with only one of these alternatives.

```
MAIN_INITIALIZATION
ITERATION = 0
Read test case precedence information
Generate directed graph
|Initialize pheromone

ANTS_INITIALIZATION
    FOR ALL ants
        FOR ALL vertices tj ∈ V, visited_j ← False
            ant_k.allowedNodes = UPDATE_ALLOWED_NODES()
            ant_k.nextNode = FIND_INITIAL_NODE()
            ant_k.visitedNode.add(nextNode)

MAIN_LOOP
    MAIN_INITIALIZATION()
    ANTS_INITIALIZATION()
    WHILE (ITERATION < MAX_ITERATIONS)
        FOR EACH ant_k
            WHILE ant_k.allowedNodes.size > 0
                k = ant_k.actualNode
                ant_k.nextNode = ant_k.FIND_NEXT_NODE()
                j = ant_kactualNode
                ant_k.visitedNodes.add(ant_k.nextNode)
                ant_k.allowedNodes = UPDATE_ALLOWED_NODES()
                Update pheromone in edge (k, j),with τ_kj = (1 − φ) · τ_kj + φ · τ_0

    MAIN LOOP FINALIZATION
        currentSolution = EVALUATE_BEST_SOLUTION()
        IF ((bestSolution is null) or (bestSolution < currentSolution)) THEN
            bestSolution = currentSolution
        ITERATION ++

    RETURN bestSolution

ant_i.UPDATE_ALLOWED_NODES()   // Updates the allowedNodes set for ant_i
    FOR ant_i
        ant_i.allowedNodes.clear()      // clear the set of allowed vertices for this ant
        FOR ALL vertices tj ∈ V         // all test cases are verified again
        IF (tj respects the precedence constraint and is not yet in solution)
        allowedNodes.add(tj)  // add the vertice to the allowedNodes set

ant_i.FIND_NEXT_NODE()          // Finds the next node for ant_i to complement its solution
        Move ant k to a vertex tj with probability p_ij^k or considering max(τ_ij · w_j^β),
            considering only nodes in allowedNodes set
```

Fig. 1. ACO Based Algorithm for Test Case Prioritization Problem.

After initialization, the main loop is executed. For each ant, while there are vertices to visit, *FIND_NEXT_NODE()* function is called to seek the next vertex, based on

heuristic and pheromone information, taking into account only test cases presenting in allowedVertices set. After the next vertex is chosen, the function *UPDATE_ALLOWED_VERTICES()* is called to update the allowed vertices for the ant in that moment (test cases that have no precedents or that all precedents are contained in visitedNodes).

The best solution in the execution of iterations is then returned by algorithm and the ant updates pheromone.

5 Conclusions and Future Work

There are few applications of Ant Colony based algorithms in software testing problems, as can be seen in [10] and [11]. Our research has focused on dealing with the test case prioritization problem considering the precedence among test cases. Currently, we are in the process of implementing the ACO-based algorithm. After the implementation is complete we will evaluate it and compare the proposed approach to other search-based algorithms.

References

1. Mathur, A. P.: Foundations of Software Testing. Pearson (2008)
2. Myers, G.: The Art of Software Testing. John Wiley & Sons, Inc, 2nd Edition (2004)
3. Harman, M., Jones, B. F.: Search-based software engineering. Information and Software Technology , vol. 43, n. 14, pp. 833-839 (2001)
4. Maia, C. L. B., Carmo, R. A. F., Freitas, F. G., Campos, G. A. L., Souza, J. T.: Automated Test Case Prioritization with Reactive GRASP. Advances in Software Engineering (2010)
5. Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., Roos, R. S.: Time-Aware Test Suite Prioritization. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 1-12 (2006)
6. Mansour, N., Bahsoon, R., Baradhi, G.: Empirical Comparison of Regression Test Selection Algorithms. Journal of Systems and Software, vol. 57, n. 1, pp. 79-90, Elsevier (2001)
7. Dorigo, M., Stutzle, T.: The ant colony optimization metaheuristic: Algorithms, applications, and advances. Handbook of Metaheuristics. Springer, pp. 250-285 (2003)
8. Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., Roos, R. S.: Time-Aware Test Suite Prioritization. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 1-12 (2006)
9. Li, Z., Harman, M., Hierons, R. M.: Search Algorithms for Regression Test Case Prioritization. IEEE Transactions on Software Engineering, vol. 33, n. 4, pp. 225-237 (2007)
10. Singh, Y., Kaur, A., Suri, B.: Test case prioritization using ant colony optimization. ACM SIGSOFT Software Engineering Notes, vol. 35, n. 4, pp. 1-7 (2010)
11. Suri, B., Singhal, S.: Implementing Ant Colony Optimization for Test Case Selection and Prioritization. International Journal on Computer Science and Engineering, vol. 3, n. 5, pp. 1924-1932 (2011)