History-Based Test Case Prioritization for Black Box Testing on a New Product using Ant Colony Optimization

Tadahiro Noguchi, Hironori Washizaki, Yoshiaki Fukazawa
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
tadahiro93@akane.waseda.jp, {washizaki, fukazawa}@waseda.jp

Atsutoshi Sato, Kenichiro Ota
Software Test Division
SHIFT Inc.
Tokyo, Japan
{sato, kenichiro.ota}@shiftinc.jp

***Abstract*—Test case prioritization is a technique to improve software testing. Although many works have investigated test case prioritization, they focus on white box testing or regression testing. However, software testing is often outsourced to a software testing company that employs black box testing. Herein a framework is 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 Ant Colony Optimization. A simulation using two actual products shows the effectiveness and practicality of our proposed framework.**

Keywords—test case prioritization; black box testing; ant colony optimization

I. INTRODUCTION

Software testing is an essential but expensive verification process. It is commonly outsourced to a software testing company for reverification to reduce time or costs that employs black box testing. Although there is a technique called test case prioritization[1] to improve software testing, it hasn't been applied well to such third-party testing because most previous studies on test case prioritization[1][2][3] involve code coverage[4]. For black box testing, J. M. Kim, et al.[5] and H. Aman, et al.[6] proposed using historical test case performance data on regression testing prioritization. However, the first iteration of black box testing on a new product is difficult to apply due to insufficient historical data on the same test cases and the precedence constraints between test cases.

To employ test history data on a similar prior product in test case prioritization for black box testing on a unit and integration testing of a new product, we propose using test categories in history data collection instead of test cases. Also, to consider the precedence and resource constraints, we use the Ant Colony Optimization (ACO) as a prioritization method and the Average of the Percentage of Faults Detected (APFD) as the test execution order evaluation.

Our main contributions are:

- We propose a framework to apply ACO to history-based test case prioritization for black box testing.
- Our framework is applied actual products, confirming that it improves the effectiveness of black box testing on new products within a practical time.

II. BACKGROUND

A. ACO

ACO is a metaheuristic algorithm based on the behavior of ants seeking food. This algorithm is composed of four steps:

Step 1: Each ant traverses a graph, which is thoroughly connected and contains a set of vertices $\boldsymbol{V}$ and edges $\boldsymbol{E}$. The next vertex is selected according to the probability calculated from a pheromone deposited on each edge and heuristic information to produce the order of vertexes (*path)*. **Step 2**: Evaluate each path. **Step 3**: Terminate the process if the end condition is met. **Step 4**: Otherwise, calculate the amount of pheromone deposited in this iteration, and return to Step 1.

We adopt ACO because the precedence constraints are easily treated by excluding vertexes that violate the constraints when choosing the next vertex in Step 1.

B. Average of the Percentage of Faults Detected (APFD)

In this study, the order of test cases is evaluated by APFD, which measures the weighted average of the fault coverage during software testing. APFD is defined as

$$APFD = 1 - (TF_1 + TF_2 + ... + TF_m) / (nm) + (1/2n) \quad (1)$$

where T represents the test suite, m represents the number of faults, n is the number of test cases, and TF_i is the position of the first test case in T that reveals fault i.

III. PROPOSED FRAMEWORK

Fig. 1(a) overviews our proposed framework.

1) Testers classify test cases of a prior product and a target product into *test categories*. An example of a classification strategy is feature-based classification (e.g., Create New Item**,** Update Item**,** Sort, etc.).

2) Testers collect historical performance data for each category. The data include the number of test cases and the number of detected faults for each severity (e.g., major, normal, minor and trivial).

3) Testers construct a list of precedence constraints between categories (e.g., test cases of Update Items must be executed after those of Create New Item, etc.)

4) Our framework generates a prioritized category order for the prior product using ACO.

5) Our framework then creates a prioritized category list for the target product by comparing the generated list in step 4 and the list of the target product.

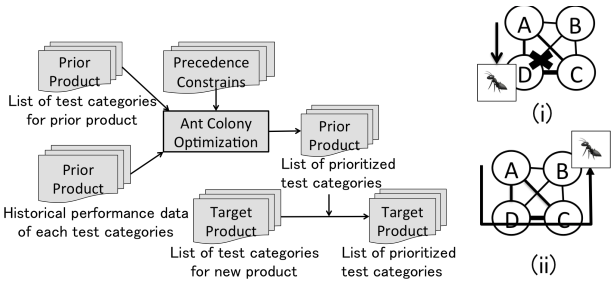

Fig. 1. (a) Overview of the proposed framework, (b) Concept image of ACO

The test history data on a similar prior product cannot be employed directly in the test case prioritization for another product due to the differences between them. Abstracting test cases into test categories makes the historical data reusable on another new product, because both test categories usually roughly correspond each other according to our experience. This shows the limitations of our framework at the same time. Optimizing test categories doesn't always produce optimized test cases. Moreover, the tendency to fault detection on a new product can be different from that on a prior product.

To use ACO, the end condition as well as how to calculate pheromones, heuristic information, and the path evaluation value must be determined. We employs the following settings:

- **End Condition**: Complete after the 10th iteration.
- **Pheromone**: 1 as the initial value on each edge with a 10% evaporation rate and a 100% as deposition rate.
- **Heuristic Info**: Weighted Number of Faults Detected (WNFD), which is defined as

$$WNFD = \alpha M + \beta n + \gamma m + \delta t \quad (2)$$

where *M, n, m,* and *t* represent the number of major, normal, minor, and trivial bugs, respectively. α, β, γ, and δ represent the weighted rate for each severity level (500, 100, 10, and 1 in our research, respectively).

- **Path Evaluation**: APFD.

Fig. 1(b) shows the core idea of Step 4 using four test categories named A, B, C, and D. If an ant visits D after A, and B must be executed after C, the ant should choose C(Fig. 1b(i)). If the best ant visits vertexes in order A-D-C-B, the prioritized category list as output will be A-D-C-B (Fig. 1b(ii)).

IV. Evaluation

We performed a simulation to evaluate (a) whether our framework improves the effectiveness of black box testing on a new product and (b) whether results are obtained in a timely manner. We used two actual products tested by the company that two of the authors work for: medical software, which has about 17000 test cases, 100 test categories, and 1400 faults, and financial software, which has around 3000 test cases, 50 test categories, and 80 faults. All categories in financial software are also appeared in medical software. The medical software was used as the prior product and the financial product as the target product. We implemented the proposed framework 100 times and generated 100 prioritized category lists. As a control, we also created 100 random-ordered category lists with the same precedence restrictions. These lists were evaluated with the APFD on the target product. Fig. 2 shows the results.

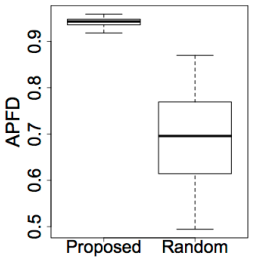

Fig. 2. Box plots of APFD for prioritized and random-ordered lists

The average APFD of the lists that our framework generated is 0.94, while that of the random-ordered lists is 0.69. Our framework took 9.3 seconds on average to finish. These results suggest that our approach successfully improves the effectiveness of black box testing on a new product within a reasonable time.

V. Conclusion and Future Work

We propose a history-based framework to prioritize test cases for black box testing on new products using ACO. A simulation using two real products shows that our framework can improve the effectiveness of black box testing within a practical time. In the future, we plan to conduct simulations on more diverse products to refine our framework. Moreover, we aim to investigate an automated precedence constraint detection algorithm to reduce the human cost.

References

[1] G. Rothermel, et al., "Test case prioritization: an empirical study," ICSM'99, pp.179-188.

[2] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," ACM SIGSOFT ISSTA, 2002, pp.97-106.

[3] Z. Li, M. Harman, and R.M. Hierons, "Search algorithms for regression test case prioritization," IEEE TSE, 33(4), 2007, pp.225-237.

[4] K. Sakamoto, K. Shimojo, R. Takasawa, H. Washizaki and Y. Fukazawa, "OCCF: A Framework for Developing Test Coverage Measurement Tools Supporting Multiple Programming Languages," Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013), pp.422-430, Testing Tools Track, 2013.

[5] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," Proc. 24th Int'l Conf. Software Eng., pp. 119-129, 2002.

[6] H. Aman, et al., "Application of the 0-1 Programming Model for Cost-Effective Regression Test," COMPSAC'13, pp.720-721.