APPLY ANT COLONY ALGORITHM TO TEST CASE PRIORITIZATION

Chien-Li Shen* and Eldon Y. Li,
Department of Information Management, College of Commerce
National Chengchi University, Taiwan
E-mail: 99356508@nccu.edu.tw, eli@nccu.edu.tw
*Corresponding author

ABSTRACT
Regression test always take resource constraints into consideration, so how to choose an appropriate set of test cases among all is a crucial issue for regression test planning. This paper aims to utilize Ant Colony Algorithm and design a suitable prioritization process to optimize defect detection rate and performance of regression test under certain defined constraints. In addition, historical testing records and defect severity are also taken into account. Finally, for research validation, Average Percentage of Faults Detected (APFD) metric is employed to compare the result with the one of a set of test cases conducted by experienced testing leaders.

Keywords
Ant colony algorithm, test case prioritization, regression test

1. INTRODUCTION

Regression test conducted in the software development process can be used to ensure no unexpected surprise or side effects after code changes modified by software programming developers. This is, original system behaviors stay the same if no related functionalities are changed. Since incremental development are adopted by more and more software systems or applications, regression test become an important task of feature validation and quality assurance in each iteration of software development cycle, and also a key to software release success. Choosing every test set among all the test cases under different resource constraints and testing purposes is mostly executed by senior quality assurance management according to their working or project experience. Therefore, how to effectively improve efficiency of software testing with an effective method to detect errors on software systems becomes a significant part of software testing.

2. TEST CASE PRIORITIZATION

According to different project goals and performance criteria, different quantity of test cases would be chosen, and to-be-tested test cases would be arranged in different testing order. Rothermel et al. (1999) summarized and classified the testing objectives into the following types in test case prioritization.

- Increase the rate of defect detection: the potential defects can be discovered in the earlier stage of regression test.
- Increase the rate of detection of high-risk defects: the potential high-risk defects can be discovered in the earlier stage of regression test.

- Increase the rate of regression errors discovered: the potential regression errors related to specific code changes can be found in the earlier stage of regression test.
- Increase the rate of the code coverage: the designed test cases can cover as many codes as possible at a faster rate in the system.
- Increase the confidence in the reliability of the system: the confidence in the reliability of the system can increase at a faster rate through the regression test.

Besides those testing objectives, Elbaum et al (2002) proposed the prioritization techniques not only limited to General Prioritization and Statement-level testing, but also advanced to Version-Specific Prioritization and Function-level testing due to cost constraints. In practice, regression test planning always take resource constraints into consideration, such as project or tester resources, testing time, or testing cost. In addition, other conditional restrictions also take into account, such as defect number or defect severity. In addition, regression test is always scheduled to conduct many times during the life cycle of incremental software development, so historical testing records can also be utilized for test case prioritization [7].

The aforementioned scholars perform optimization by using test case prioritization techniques, and in recent years other scholars adopt other algorithms to solve the test case prioritization problems. For example, Li (2007) referenced Greedy Algorithm, Zhang (2009) combined the traditional test case prioritization techniques with Generic Algorithm and compared those two, and Singh (2010) employed Ant Colony Optimization and added time constraint in the Statement-level testing to find the highest defect detection rate.

3. ANT COLONY OPTIMIZATION

Ant Colony Optimization, proposed by Dorigo in 1992, came out by observing foraging behavior of ants in nature. During food hunting, ants leave pheromone on the traveled paths, and the shortest path will be discovered through teamwork and pheromone evaporation process. Assume in the beginning of food foraging, ants choose their paths in random toward the direction of the food source, even when they come to a fork in the road, so any possible path would remain pheromone odor (Figure 1). In the return trip, since the pheromone evaporates in different evaporation rates according to the length of paths, it leads to different amount of residual pheromone. Therefore, the longer the path, the more pheromone evaporation, the less residual pheromones. The other follower ants will choose the shorter path according to the amount of residual pheromones (Figure 2). Through the time evolution of the group cooperation, ants will eventually choose the shortest path.

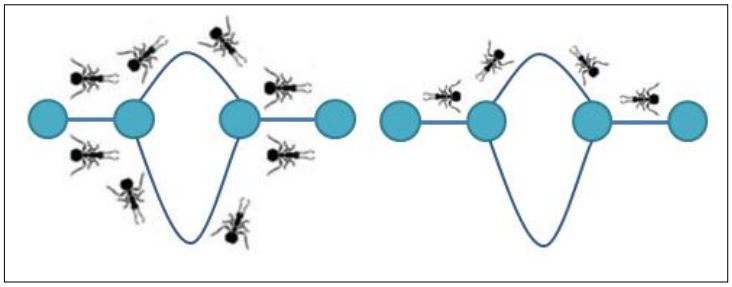

Fig. 1. Random Paths

Fig. 2. A shortest path

In addition, the number of ants also affects the search result of the shortest path. In the initial stage of foraging search, there must be a sufficient number of ants to perform path mining. If the ant colony size

is too small, it would not be able to comprehensively search for all the possible paths and may lead into a local optimization.

Ant System Algorithm has been proved as an effective optimization method, and also widely applied to a variety of industry issues [3], such as quadratic assignment problem [9], job scheduling problem [2], vehicle routing problem [1, 12], shortest common supersequence [10], data mining [11], and test case prioritization [14] mentioned in chapter 2.

4. RESEARCH DESIGN

Software testing in practice is usually managed under the time and cost constraints. After a full run of system test, regression test is scheduled afterwards to verify the revisions with two main purposes. One is to validate all the known bugs or defects have been fixed properly, the other is to ensure there is no unexpected surprise or side effects and all the system functionalities still work normally. Under time and cost constraints, only parts of the designed test cases are chosen to perform in regression test, so how to prioritize the test cases is a crucial issue in this study.

According to project objective and target, the procedure of test case prioritization can design by employing different algorithms and condition variables. In this study we adopt Ant Colony Algorithm and function-level testing method along with historical testing records to assess test cases and defect severity in order to prioritize test cases and assign the potential fail rate to each test case. The optimization target is to find more defects by using fixed number of test cases (with the higher rate of defect detection); this is, conducting the minimum number of test cases can discover the maximum potential defects.

In summary, the key design factors of test case prioritization are:

- Use Function-level test
- Specify specific versions for prioritization
- Utilize historical testing records for test case assessment
- Take defect severity into account
- Use Ant Colony Algorithm for optimization
- Execute the minimum test cases to discover the maximum potential defects

4.1 Define variables

According to system design architecture and Function-level test, a system can be classified into M functions, each function contains N test cases, and each test case belongs to one specific function only. The variables of function and test case for prioritization are defined as below.

- T stands for all defined test cases, T = {t1, t2, ..., tN}, and is defined as (testID, a set of defects, pValue), in where pValue represents a weighted pheromone value calculated according to the number of defects and defect severity as well as the historical testing records. In addition, each test case only contains on fault.
- C stands for all defined system functions, C = {c1, c2, ..., cM}, and is defined as (funID, funName, funTCsNum, sumPValue), in which sumPValue represents the sum of pheromone of all test cases in the cM funtion.
- PT stands for the test suite, which contains a set of test cases to be prioritized.

- F is the target function to convert PT for optimization performance evaluation
- Optimization is to search for T' PT, (T")(T" PT)(T"≠T')[f(T')≧f(T")] [13]
- G(V, E) is the undirected graph used for searching optimization path, in where V (vertices) represents the node set for test cases and E (edge) represents the path set.

The ant colony optimization algorithm used in this study is adjusted as follows [3, 6].

Set parameters for functions and test cases
Initialize pheromone trails
While termination conditions not met do
Construct Ant Solutions (Test Case Suites)
Update Pheromones
End while

4.2 Define pheromone value and evaporation

The ant colony algorithm determines which path each ant would choose when it comes to a fork in the road according to the amount of pheromone concentration (high or low). Since the pheromones left by ants evaporate slowly over time at a fixed evaporation rate, the long path will take the longer time to walk through. Therefore, the formula of the pheromone update and the pheromone evaporation from test case i to test case j is defined as below:

$$T_{ij} \leftarrow (1-\rho)\cdot T_{ij} + \sum_{k=1}^{m} \Delta T_{ij}^{k} \quad (1)$$

$$\Delta T_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if ant k used edge(i, j)in its tour} \\ 0 & \text{other wise} \end{cases} \quad (2)$$

In where:

ρ is the pheromone evaporation rate and this value is set to 10% as a constant [3];

ΔT_{ij}^{k} stands for the residual pheromone per unit length left by the k^{th} ant;

L_k stands for the length of a path traveled by the k^{th} ant;

Q is set as a constant.

4.3 Define path probability

After the pheromone for each path is calculated and assigned to each path in the runtime, this value can be converted to the path probability for the next following ant for path selection. Therefore, the formula of path probability from test case i to test case j is defined as below.

$$p_{ij}^{k} = \begin{cases} \frac{T_{ij}^{\alpha}\cdot n_{ij}^{\beta}}{\sum t\in allowed_k T_{ij}^{\alpha}\cdot n_{ij}^{\beta}} & if\ j \in allowed_k \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$n_{ij} = \frac{1}{d_{ij}} \quad (4)$$

In where:

$allowed_k$ is the list of test cases which the K^{th} ant has not traveled through yet;

α is the variable to control the pheromone concentration;

β is the variable to control the absolute distance;

d_{ij} stands for the distance between test case i and test cases j.

5. VALIDATION DESIGN

5.1 Average Percentage of Faults Detected (APFD)

Many researches associated with test case prioritization employ Average Percentage of Faults Detected (APFD) metric to validate the performance and effectiveness of algorithm and optimization [4, 5, 14, 15]. According to the definition of APFD, the equation is defined as below to fit our research design.

T-> The test suite in a specific version
m -> number of faults contained in a specific version
n -> number of test cases conducted in a specific version
TF_i -> The position of the test case found fault i in T
APFD= 1- (TF_1+TF_2+ ……+ TFm)/(n*m) + 1/(2*n)

5.2 Data Sets and Historical records

To assess the results of our research design, not only APFD is adopted, but also the results from the historical recorders are used to compare the accuracy against the one of the ACO algorithm with the adjusted values of variables. Our data sets are derived from a software development company in Taiwan, and in order to generate historical statistics, there are 35,623 test cases records in total conducted on three test runs by four projects as shown on Table 1. According to the company's design of test cases, each test case contains or represents only one fault and test suite reduction is adopted for each test run. This is, those test cases in each test set are selected according to issue change list, such as new added features, fixed bugs, or related functions associated with code changes.

In our research design, Project A, B, and C are used as a training set to evaluate and adjust the variable values, such as a weighted pheromone value and a weight from historical statistics. In addition, Project D is used as a test set to validate the performance and accuracy assessing by APFD and against historical results respectively. In order to assess the effectiveness of our algorithm design, the APFD value of the best prioritization case is calculated, and the best scenario is to have all faults found in the very first order sequence.

Table 1. Number of test cases conducted by four projects

Project Code	Test code & version	Num of Functions	Num of Test Cases	Faults Found	APFD Best case
A	ST_V0.11	69	5,437	44	99.60%
	RT1_V0.19	69	5,555	44	99.60%
	RT2_V0.24	59	3,620	23	99.68%
B	ST_V0.11	54	4,000	13	99.84%
	RT1_V0.14	28	2,581	7	99.86%
	RT2_V0.21	24	1,090	1	99.95%
C	ST_V0.08	62	1,972	22	99.44%
	RT1_V0.18	52	1,314	10	99.62%
	RT2_V0.19	52	1,033	3	99.85%
D	ST_V0.20	60	3,499	21	99.70%
	RT1_V0.33	55	2,599	10	99.81%
	RT2_V0.39	36	2,923	1	99.98%
Total			35,623	199	

6. CONCLUSION

This paper proposes a research model by utilizing the Ant Colony Algorithm and test case prioritization technique to optimize the test case prioritization. The expected contribution is to help software quality assurance stakeholders identify the higher-risk test cases which enable project managers to control testing execution time and testing budget. To validate the result of prioritization in this model, the result will compare with the one of a set of test cases chosen by experienced test leaders in each run of regression test along with the APFD value. In addition, the parameters in the algorithm will be adjusted in order to dig out the best suitable variable values in this algorithm.

7. REFERENCES

[1] Bullnheimer, B., Hartl, R.F., and Strauss, C. 1999. *An improved Ant System algorithm for the vehicle routing problem*. Annals Of Operations Research, 89 (1999), 312.

[2] Colorni, A., Dorigo,M., Maniezzo, V., and Trubian,M. 1994. *Ant System for Job-Shop Scheduling*. Belgian Journal of Operations Research, Statistics and Computer Science, 34, 1 (1999), 39–53.

[3] Dorigo, M. and Socha, K. 2006. *An Introduction to Ant Colony Optimization*. IRIDIA - Technical Report Series, ISSN 1781-3794.

[4] Elbaum, S., Malishevsky, A.G., and Rothermel, G. 2001. *Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization*. Proceedings of the 23rd International Conference on Software Engineering (May. 2001).

[5] Elbaum, S., Malishevsky, A.G., and Rothermel, G. 2002. *Test case prioritization: a family of empirical studies*. IEEE Transactions on Software Engineering. 28, 2 (Feb. 2002).

[6] Jones, K.O., and Bouffet, A. 2008. *Comparison of Bees Algorithm, Ant Colony Optimization and Particle Swarm Optimization for PID controller tuning*. Proceeding CompSysTech '08 Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing.

[7] Kim, J.M., and Porter, A. 2002. *A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments.* The 24th International Conference on Software Engineering. (May, 2002), 119-129.

[8] Li, Z., Harman, M., and Hierons, R.M. 2007. *Search Algorithms for Regression Test Case Prioritization.* IEEE Transactions on Software Engineering, 33, 4 (April 2007), 225-237.

[9] Maniezzo, V., and Colorni, A. 1999. *The Ant System applied to the quadratic assignment problem*. Knowledge and Data Engineering, IEEE Transactions on, 11, 5 (Sep/Oct 1999), 769-778.

[10] Michel, R., and Middendorf, M. 1999. *An ACO algorithm for the shortest common supersequence problem.* New Ideas in Optimization. McGraw-Hill Ltd., UK Maidenhead, UK, England, 51-62.

[11] Parpinelli, R.S., Lopes, H.S., and Freitas, A.A. 2002. *Data Mining with an Ant Colony Optimization Algorithm*. Evolutionary Computation, IEEE Transactions on, 6, 4 (2002).

[12] Reimann, M. and Ulrich, H. 2006. *Comparing backhauling strategies in vehicle routing using Ant Colony Optimization*. Central European Journal of Operations Research, 14, 2 (Jun 2006), 105.

[13] Rothermel, G., Untch, R.H., Chu, C., and Harrold, M.J. 1999. *Test Case Prioritization: An Empirical Study.* Proceeding of the International Conference on Software Maintenance, Oxford, UK, (September 1999).

[14] Singh, Y., Kaur, A., and Suri, B. 2010. *Test Case Prioritization using Ant Colony Optimization.* ACM SIGSOFT Software Engineering Notes, 35, 4 (July 2010).

[15] Srivastava, P. R. 2005. *Test case prioritization*. Journal of Theoretical and Applied Information Technology, 178-181.

[16] Zhang, L., Hou, S., Guo, C., Xie, T., and Mei, H. 2009. *Time-Aware Test-Case Prioritization using Integer Linear Programming.* Proceedings of the eighteenth international symposium on Software testing and analysis, (July 2009), 213-224.