Adaptive Random Testing for Web Application

Liao Chunhao , Kang Yuhui , Yang Wenhanfu
Department of Computer Science and Engineering
Southern University of Science and Technology
11910506@mail.sustech.edu.cn, 11910211@mail.sustech.edu.cn, 11910222@mail.sustech.edu.cn

***Abstract*—In these days, web testing has been recognized as a notoriously difficult task and still heavily relies on manual efforts while automated web testing is far from achieving human-level performance. Key challenges in web testing include dynamic content update and deep bugs hiding under complicated user interactions and specific input values, which can only be triggered by certain action sequences in the huge search space. In this paper, we propose using Adaptive Random Testing algorithms based on coverage to achieve automated web application testing with fewer test cases in needed and better performance achieved.**

1. Introduction

1.1. Basic Concept and introduction to Adaptive Random Testing

As an improvement on random testing (RT) [1], adaptive random testing (ART) [2] introduces adaptive selection of random test cases. Since the program inputs that trigger failures tend to cluster into contiguous regions, ART is considered to be necessary to achieve an even spread of (random) test cases over the input domain. It defines the distance between test cases and tries to select test cases that are farther apart for testing. Compared with RT, ART achieves higher testing efficiency, better failure detection capability and higher code coverage with fewer test cases.

1.2. The usage scenario

Consider a scenario where a network application is developed and ready for deployment. Engineers want to continually add and test new features in subsequent use. He also wants to be able to test as many code as possible with as few test cases at a time. The core of this tool is to record the coverage rate and area of each test case, and use ART to select the best few test cases to achieve the testing goal.

2. ART test case generation

2.1. Test case generation process

The core of ART is to select the best test case among a set of candidate test cases. The following figure shows the flow chart of ART test case generation process of the test framework.

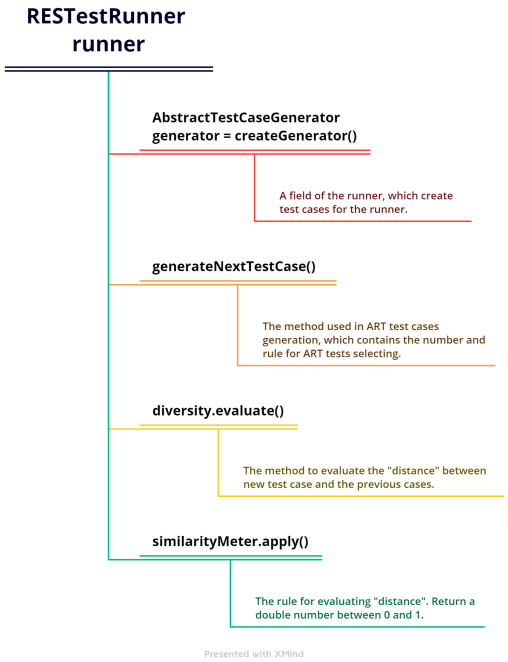

Figure 1. ART test cases generation process

An ARTestCaseGenerator should be prepared for the runner. This object is a generator which has a List of test case, and the evaluating rule determined by the properties file. This generator is created by the method createGenerator.

The generator needs to determine which test case to be selected among the candidate set. This function is defined in the generateNextTestCase method. The difference is measured by the difference between new test case and the executed test cases.

Then the evaluate method will take one of the executed test cases and use the evaluating metric to calculate that distance. This metric is also determined by the properties file.

The apply method of the metric will calculate the distance by its own definition. After the calculation, runner is able to compare and choose the next test case.

2.2. ART algorithms implemented

2.2.1. Partitioning-Based Strategy. The Partitioning-Based Strategy (PBS) [4], [5]divides the input domain into a number of subdomains, choosing one as the location within which to generate the next test case. Core elements of PBS, therefore, are to partition the input domain and to select the subdomain. In this project, we divide each method as a separate domain

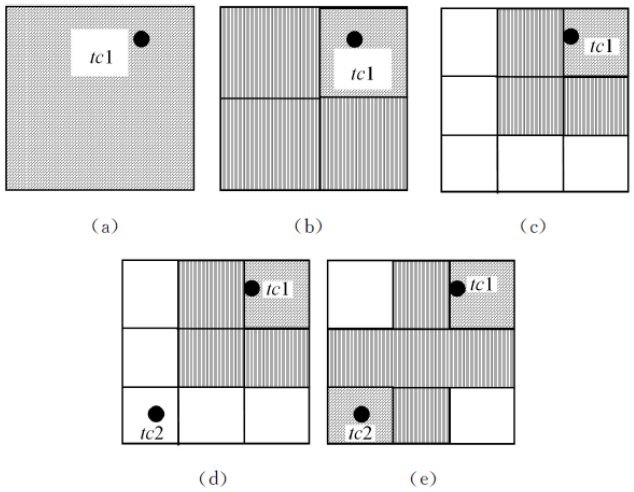

Figure 2. Illusion of ART through iterative partitioning

2.2.2. Select-Test-From-Candidates Strategy. The Select-Test-From-Candidates Strategy (STFCS) chooses the next test case from a set of candidates based on some criteria or evaluation involving the previously executed test cases. In this project, specifically speaking, we have used Fixed-Sized-Candidate-Set(FSCS) [9] strategy to select test cases for execution from the candidate test cases. An element from candidate set is selected as the next test case such that it is farthest away from all previously executed tests.

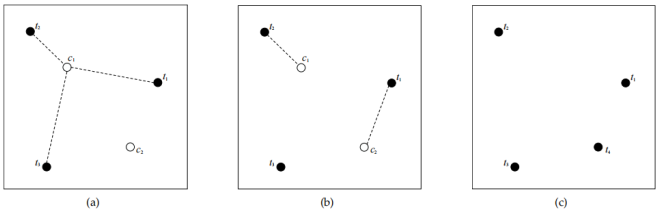

Figure 3. Illustration of FSCS in a two-dimensional input domain

2.3. The distance evaluating metrics

The evaluating metric is defined as a enum class in diversity.java. There are five metric in this test framework now. One of them is for the content of the test case. And the other are for test cases in string type.

JACCARD [6]. For string type test case. This metric divide two test cases into char hash set. The difference is defined as the quotient of the size of intersection and union set.

JARO WINKLER [7]. This metric is to use the JARO WINKLER distance between the test cases after changed into string type as the distance.

LEVENSHTEIN [8]. It is defined as the times an operation of change, insert and delete that needs to be applied to make the two string be the same.

LONGEST COMMON SUBSEQUENCE. Calculate the length of the longest common subsequence. And the difference is the quotient of that length and the length of the entire sentence.

CONTENT. This metric is to divide the test case into parts. Each part has its own weight. The distance is the sum of the weight of the part that are not same.

3. The content of this semester

3.1. Reconstruct and optimize RESTest [3] for developing new ART web testing tool

1) *Take RESTest as a reference to build up ART tools* Implement the details autonomously and take the framework as reference.
2) *Build up the core testing code and add new implementation way of ART*
3) *Compare the performance of different ART algorithms.*
4) *Develop RESTful web API testing tool based on ART.*
5) *Combine Soot for code analysis.* Analyze the source code of the target and give useful information such as code coverage to support the ART algorithms.

3.2. The Architecture of our Testing Tool Framework

1) *Test model generation.* The system model for the API specification, and the test model for all test-related configuration settings.
2) *Abstract test case generation.* Generate the abstract test cases for ART from the system and test models.
3) *Test case generation.* Create test cases and select the best test cases by the feedback message from the feedback block. ART is used in this block to guide the test cases generating strategy.
4) *Test case execution.* Test cases are optionally executed and the test results are exported to a machine-readable format and reported to the user.
5) *Feedback collection.* Collect the execution result and report to the test case generation block.

4. Our Current Work

4.1. Structure Overview

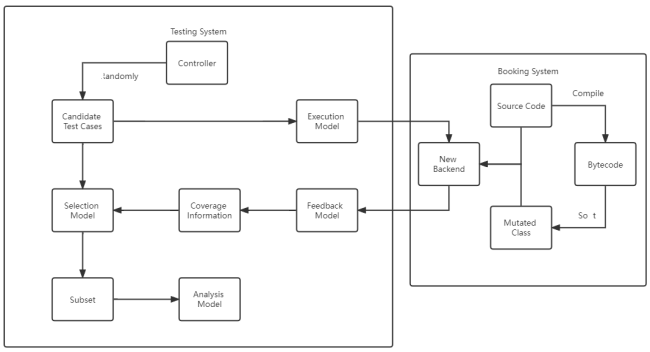

Figure 4. Structure overview

4.2. Soot Instrumentation

4.2.1. What is Soot.

Originally, Soot started off as a Java optimization framework. By now, people use Soot to analyze, instrument,optimize and visualize Java and Android applications.

Soot transforms programs into an intermediate representation, which can then be analyzed. Soot provides four intermediate representations(IR) for analyzing and transforming Java bytecode: Baf, Jimple, Shimple and Grimp. Jimple is Soot's primary IR and most analyses are implemented on the Jimple level.

In this project, we use Soot to instrument the java code, then we can get useful information, for example the code coverage, to support the ART.

4.2.2. Soot Implementation Details.

- First,we need to get the .class file as the option *allow-phantom-refs* can only deal with the bytecode.
- Then we traverse each method in service level and do instrumentation to each line. For each line ,if it is carried, the information will be write to the local file.
- Finally we need to replace the mutated class with the origin class in the target direction.

4.2.3. Coverage Result Format.

For each line the format is *a-b-c-d*,where *a* means which service,*b* means which method,*c* means how many lines this method have and *d* means which line is covered.

4.3. Candidate Test Set Generation

Firstly, we use open-api equipped in maven to generate .yaml file, which make a description to the interfaces of the web application.Then random test cases (candidate test cases) will be generated based on this tool. The test cases within the candidate test case list will be executed as sending a http request querying the interface.

As the core class of the web application has been instrumented by Soot, the coverage information will be given back to the tool after the test cases has been executed. Finally, the tool will write a file containing test cases and their corresponding coverage information.

4.4. Select Test Cases

1) *Read the information of test case and coverage rate.* Read file in the disk or in a database, reconstruct the test cases.
2) *Evaluate by the coverage* Use the coverage meter defined by the number of new lines or methods this test case can cover from the old test cases collection.
3) *Execute the test collection* Collect the best test cases, and execute them to test the web application API.

To select the test cases, we use two ART algorithm as comparison and a List of coverage.

This list is formed by the order of the web application class being tested. It has the size of the number of .class, and each item has a list of size of the method in this .class file. And the unit item is a string containing 0 and 1, representing uncovered and covered.

When doing selection, the tool repeat selecting the test case which has the greatest contribution.

The contribution is defined by the difference between the test case's coverage list and the list has been covered. Two ART algorithm are used in our project, one is FSCS (Fixed Size Candidate List) and the other one is PBS (Partitioning Based Strategy). In FSCS, we tried to find the test cases in the candidate set with the largest difference with the executed test cases; In PBS, we first to partition the source code of the web application by their methods, attempting to make each methods to accessed covered.

After added a test case into the result list, put the new covered lines to the total covered list to record which line has been covered. The covered line will not give contribution any more.

5. Result Presentation and Analysis

5.1. Result Presentation

When ART and RT achieved same line coverage, the number of test cases is shown below:

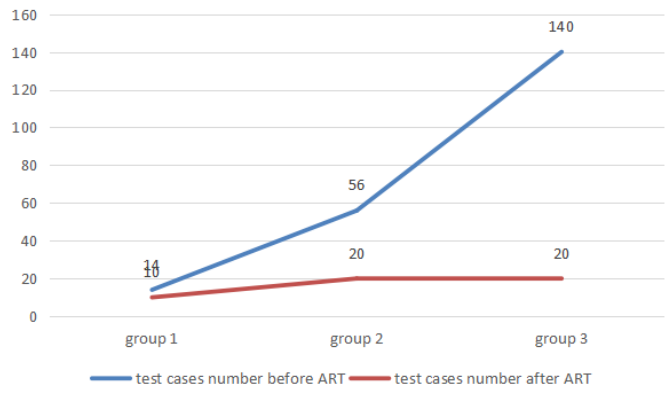

Figure 5. The number of test cases when in the same coverage

For testing our tools, we use the ZhiXin College's booking system as our target. This is because this project is relatively simple and we also have its source code and it supports the RESTful API.

First, we randomly generated 14 test cases and screened them in an attempt to obtain the same coverage with the result of RT by using less test cases, and we found that only 10 test cases are used with the same coverage result achieved.

Secondly, we randomly generated 56 test cases and in this time, only 20 test cases are selected and the coverage information are same as the information created by totally 56 test cases.

Thirdly, we randomly generated 140 test cases and the number of test cases after selection is still 20, which means that only increasing the number of tests can no longer improve coverage.

5.2. Result Analysis

Consider the coverage information generated by the test cases, we found that the number of test samples needed to achieve the highest coverage is limited. Firstly, we believe that it is partially because the ZhiXin College's booking system is relatively simple and the number of method is not really big. Secondly, we found that the test cases from random generated testing is so inefficient that it tends to produce a lot of similar but not correct test cases, which causes a great deal of wasting. Accordingly, our alternative test cases are limited.

6. Future Work and Conclusion

In the future, we are supposed to test a more complicated web application and attempts to combine the ART with other testing algorithm to improve its efficiency and capability of detecting bugs.

In conclusion, we completed the goals set at the beginning of the semester. We realized the web application testing using ART and achieved a good effect, confirmed the high efficiency of ART, also successfully combined with Soot to implement the instrument and analyze the network application.

Our sincere thanks to Prof. Yepang Liu and Prof. Yuqun Zhang for your careful and professional guidance.

References

[1] T. Y. Chen, T. H. Tse, and Y. T. Yu, "Proportional sampling strategy: A compendium and some insights," Journal of Systems and Software, vol. 58, no. 1, pp. 65–81, 2001. of Future of Software Engineering (FOSE'14), 2014, pp. 117–132.

[2] A. Orso and G. Rothermel, "Software testing: A research travel ogue (2000-2014)," in Proceedings of Future of Software Engineering (FOSE'14), 2014, pp. 117–132z

[3] Martin-Lopez A, Segura S, Ruiz-Cortés A. RESTest: Black-box constraint-based testing of RESTful web APIs[C]//International Conference on Service-Oriented Computing. Springer, Cham, 2020: 459-475.

[4] K. K. Sabor and S. Thiel, "Adaptive random testing by static partitioning," in Proceedings of the 10th International Workshop on Automation of Software Test (AST'15), 2015, pp. 28–32.

[5] J. Mayer, "Efficient and effective random testing based on partitioning and neighborhood," in Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE'06), 2006, pp. 499–504.

[6] Niwattanakul S, Singthongchai J, Naenudorn E, et al. Using of Jaccard coefficient for keywords similarity[C]//Proceedings of the international multiconference of engineers and computer scientists. 2013, 1(6): 380-384.

[7] Dreßler K, Ngonga Ngomo A C. On the efficient execution of bounded jaro-winkler distances[J]. Semantic Web, 2017, 8(2): 185-196.

[8] Yujian L, Bo L. A normalized Levenshtein distance metric[J]. IEEE transactions on pattern analysis and machine intelligence, 2007, 29(6): 1091-1095.

[9] T. Y. Chen, F.-C. Kuo, and H. Liu, "On test case distributions of adaptive random testing," in Proceedings of the 19th Internation x0002 al Conference on Software Engineering and Knowledge Engineering