Test Case Generation for Web Application Based on Markov Reward Process

Jing Cao1,2, Xiaoqiang Liu1*, Hui Guo1, Lizhi Cai2, Yun Hu3

1College of Computer Science and Technology, Donghua University, Shanghai, 201620, China
2Shanghai Key Laboratory of Computer Software Testing &Evaluation, Shanghai, 201112, China
*Corresponding author’s e-mail: liuxq@dhu.edu.cn

Abstract: Web applications often face continuous updating due to functional change or UI renew, while it remains a challenge to guarantee their correctness. The goal of software testing is to find defects in a limited time range whereas exhaustive testing is an ideal yet time-consuming process. In this research, we propose an approach to generating test cases automatically based on the Markov reward process which innovatively contains a reward function for test results to guide the generation of test cases. By using the N-step algorithm, this approach can generate the test flow with the highest risk priority which can capture software defects as quickly as possible. The experiment on an e-commerce system shows that there is significant improvement on the defect detection capability of test cases generated through Markov reward process.

1. Introduction
With the widespread use of web applications, it has become larger and more complex, including more function points and operation flows. In the continuous integration process of software development, the quality of applications has also become particularly important. Model-based test case generation method is a prominent technique in software testing. Commonly used models include FSM (Finite State Machine) model [1], UML (Unified Modeling Language) model [2, 3, 4] and Markov model [5, 6]. These models used to describe the function and operation flow of the system under test (SUT). Once the model was constructed, it can generate a large number of test cases randomly or according to fixed strategies. However, these methods will generated many invalid and repeated test cases, which will cause the low defect detection efficiency and the execution process to be time-consuming. Furthermore, it faces the problem of combinatorial path explosion in a system with complex functional operation flows.

This paper proposes a novel test case generation approach based on the Markov reward process. We collect the HTTP request data produced by users accessing the system and use it to build an abstract Markov model of SUT. Under the guidance of reward function, the model learns to prioritize the error prone test nodes by observing the execution of previous test cases. Therefore the method can continuously generate test cases with the highest risk priority which optimizes the test case set and speeds up the test efficiency. The experiments on a practical system show a preferable URL coverage and a higher defects detection rate.
The remainder of the paper is structured as follows. The section II describes the implementation details for the methods used to generate priority test case. Experiments and results of our methods are discussed in Section III. Finally, conclusions are given in the last section.

2. Method

In this research, we assume that the HTTP request data represents users’ interaction with SUT and that sequential requests represent the functional operation path of SUT. This section introduces our approach to generating test cases automatically by using Markov reward process. Figure 1 shows the workflow of the approach. The Markov model is constructed and initialized based on HTTP requests collecting from user interaction history at first. Then, the model training and application process can be launched: a test flow will be generated by the model according to N-step algorithm, and be combined with the request parameters to form test cases. Furthermore, the test case is executed to collect test results and the reward is calculated by the reward function which is used to update the model. After updating the Markov model, the model will generate the next round of test flow, which is a cyclic iterative process.

Figure 1. Workflow for automatically generate test case by using Markov reward process.

2.1. HTTP requests collection and data preprocessing

The first step of the approach is to collect a large amount of HTTP request data to build the Markov model. So we have developed a browser plug-in named ByteDust for Google Chrome to collect the HTTP requests. A HTTP request contains method, request headers and URL (see Figure 2.).

We will pre-process these request data in three different ways: First, we will segment them according to the user’s cookie and timestamp to obtain the surfing transition relationship described with a request list, where two adjacent requests represent a transfer relationship. Figure 3 shows a request list containing four request objects and three transition relationships. Second, we will de-parameterise the URLs and cluster the de-parameterized URLs in order to reduce the number of states (which will be defined in the first paragraph of 2.2). Third, the parameter value of similar URLs will be extracted and then saved into a pool to provide data source for subsequent generation of test cases (which will be conducted in 2.4).
2.2. Construction of Markov model based on the request list

When constructing the Markov model, we firstly need to define several elements such as state, transition relationship and transition probability. In this paper, the state is the accessible URL (without request parameters) of SUT, transition relationship is the next reachable request in the request list and transition probability is the probability of transiting from one state to another. Transition relationship of states are represented by a two-dimensional table (as shown in Table 1) where the columns represent state request’s URL and the rows represent the next reachable request’s URL.

| State1 | State2 | State3 | ... | StateN |
|--------|--------|--------|-----|--------|
| 0.00   | 0.23   | 0.00   | ... | 0.02   |
| 0.41   | 0.00   | 0.01   | ... | 0.00   |
| 0.00   | 0.00   | 0.02   | ... | 0.37   |
| 0.01   | 0.00   | 0.00   | ... | 0.00   |

$TP_{(S_i, S_k)}$ represents the transition probability that transfers from URL (i) to URL (k), and the larger the probability is, the more likely the transition happens. To initialize the $TP_{(S_i, S_k)}$, we calculate it through Formula 1. $T_{(S_i, S_k)}$ represents the number of transitions from URL (i) to URL (k), and $T_{Si}$ represents all requests through URL(i).

$$TP_{(S_i, S_k)} = \frac{T_{(S_i, S_k)}}{T_{Si}} \quad (1)$$

During model training and application, the web server will create a response object which contains response value, response code and response time after it receives the HTTP request from the client test case. We define a reward function considering these three factors. It is an accumulated value of fails in assertion, code and timeout (as shown in Formula 2). When transfer from State(i) to State(k), $N_{assertfail}$ represents the reward value for assertion failure, $N_{codefail}$ represents the reward value for response code error and $T_{timeout}$ represents the reward value for the response overtime.

$$Reward_{(S_i, S_k)} = N_{assertfail} + N_{codefail} + T_{timeout} \quad (2)$$

2.3. Generation of test flow with the highest priority by N-step algorithm

A test case is a sequence of requests which consists of a test flow and the parameter of each request. Here, an ordered set of states is defined as a test flow in the form of $(S_i, S_j, ... , S_k)$. To ensure the successful execution of a generated test case, a test flow must start from the initial state and end with a final state of the Markov model. Based on the initial states, the Markov model can generate the current test flow with the highest risk priority by using the N-step algorithm. The N-step algorithm can
determine the next state by calculating the accumulated transition probability from the forward n states after the current state. For example, if n is 3, the algorithm will simulate three transitions from the current state, calculate the cumulative transition probability, and then select the state with the highest transition probability S1 (shown in Figure 4).

There are two termination conditions for the algorithm. One is that the test flow goes directly to the final state of the system and the other is that when the length of the test flow reaches the maximum we defined.

2.4. Generation of test cases by combining the test flow with the request parameters
As an ordered set of states, a test flow can only be used in testing after its combination with the request parameters. We adopt the 2-factor method of PICT (Pairwise Independent Combinatorial Testing tool) for the combination process. This method designs test cases with high code coverage according to the principle of pairwise testing. All the test cases in a test case set belong to the same test flow with different parameter data. Once the combination is completed, a JSON file is obtained and the executable test cases can be generated from it.

2.5. Execution of test cases and updating the model
The test cases are executed on a Chrome browser and the test result of each request which contains response code, response assertion result and response time are gathered. Then the reward value of each request, namely Reward\(_{(S_i,S_k)}\), is calculated by the reward function and the Markov model is updated according to Formula 3 where TP\(_{(S_i,S_k)}\) is the transition probability before updating, TP\(_{(S_i,S_k)'}\) is the transition probability after updating, and \(\sum\text{Reward\(_{(S_m,S_n)}\)}\) is the accumulated reward value in a test case. After the model is updated, the initial state is set again and the next test iteration can be started.

\[
TP_{(S_i,S_k)'} = TP_{(S_i,S_k)} + \frac{\text{Reward\(_{(S_i,S_k)}\)}}{\sum\text{Reward\(_{(S_m,S_n)}\)}}
\] (3)

3. Experiments and Results
The experiments are conducted on an e-commerce system. We collected HTTP requests from August 1, 2020 to September 1, 2020. From Table 2 we can see that a total of 8743 requests have been collected. Through data preprocessing, we have obtained 38 unique URL (without parameters), 1530 request lists, and 1070 parameter groups.

| Experimental data | requests | URL | request lists | parameter groups |
|-------------------|----------|-----|---------------|------------------|
| Number            | 8743     | 38  | 1530          | 1070             |

The Markov model is initialized by the collected data and the initialized model is represented in Figure 5. A node represents a state and the size of the node represents the risk priority of the state.
After initializing the model, the training iteration on the model starts. Figure 6 shows the Markov model in training where the size of some nodes become larger owing to some test defects, which indicates that the probability of defects in this state is the greater.

We compare the Markov reward process approach with the manual test method, and calculate the URL coverage rate and defect detection rate. It can be seen from Figure 7 that under the same URL coverage rate, the number of test cases generated by our approach is less than the half of that of the manual test. In order to evaluate the effectiveness of defect detection of the generated test cases, we have manually generated some defects including response error and response timeout and injected them into the four versions of SUT. Table 3 shows the number of defects injected in each version.

| Version | version 1.0 | version 2.0 | version 3.0 | version 4.0 |
|---------|-------------|-------------|-------------|-------------|
| Number of defects | 10          | 15          | 16          | 28          |

For each system version, we used two methods to generate 100 test cases separated, and count how many software defects were found. Figure 8 shows that the number of the defects detected by our approach is more than that of the manual method. The average defect detection rate of our method is 82.63%, higher than 65.77% of the manual test. By comparing the URL coverage rate and defect detection rate, we can see that the test case generation approach based on Markov reward process is better than the manual test method and can satisfy the test goal of detecting software defects as soon as possible.

| Version | version 1.0 | version 2.0 | version 3.0 | version 4.0 |
|---------|-------------|-------------|-------------|-------------|
| URL coverage rate | | | | |

4. Conclusions
In this paper, we have developed a novel approach to automatically generating test cases for web applications based on Markov model. Through initializing and training the Markov model with real HTTP requests, the model captures the current highest priority (highest risk/most error-prone) test case in the way of N-step algorithm. By injecting defects into the web application, the effectiveness of our
approach is verified. Experiments show that test case generation approach based on Markov reward process is superior to manual test method in terms of test efficiency and defect detection rate. Here we only consider the response of the current request in the test case in the reward function. In future research, we expect to include the responses of previous requests before the current request in the reward function, which can hopefully further improve the test efficiency.

References
[1] Ahmed B S, Gargantini A, Bures M. An Automated Testing Framework For Smart TV apps Based on Model Separation [J]. arXiv preprint arXiv:2002.00404, 2020.
[2] Markthaler M, Kriebel S, Salman K S, et al. Improving model-based testing in automotive software engineering[C]/2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). IEEE, 2018: 172-180.
[3] Qu M C, Wu X H, Tao Y C, et al. Path generation algorithm for UML graphic modelling of aerospace test software [J][J]. MS&E, 2018, 322(6): 062019.
[4] Mani P, Prasanna M. Test case generation for embedded system software using UML interaction diagram [J]. Journal of Engineering Science and Technology, 2017, 12(4): 860-874.
[5] Vain J, Tsiopoulos L, Kharchenko V S, et al. Multi-Fragment Markov Model Guided Online Test Generation for MPSoC[C]/ICTERI. 2017: 594-607.
[6] Li Huan. Research on functional test cases of logistics software based on Markov chain (Markov) model [D]. Beijing University of Posts and Telecommunications, 2018.