Statistical testing data generation for UAS

Jianwei Sui*, Yunzhan Gong, Dahai Jin and Yawen Wang
State Key Laboratory Of Networking And Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China

*E-mail: jwsui@bupt.edu.cn

Abstract. The reliability of UAS(Unmanned Aircraft Systems) is of vital importance in practical applications. The most effective method to guarantee the reliability of UAS is to test UAS with multiple types of testing data. In this paper, we propose a method for the testing data generation of UAS in consideration of the combination and weight of the input data, which improves the availability of the method in UAS. Then we make four experiments to verify the effectiveness of the proposed method: (1) the random testing data generation experiment; (2) testing data generation considering the combination of input data experiment; (3) testing data generation considering the combination and weight of input data experiment; (4) the history data from the actual usage situation. Experiments results show that the method proposed in this paper is closer to the reality. At last, we analyse the code coverage by using Code Test System(CTS). And the analysis reflects that the proposed method has a better coverage than other methods.

1. Introduction
UAS reliability test is to test software on purpose of verifying whether UAS achieves reliability requirements and evaluating UAS reliability level. [1] The most commonly used method of reliability test is statistical testing method based on usage model, building software usage model and generating test cases based on the usage model. [2] Markov model is a famous model which is usually used to generate operation sequences in the traditional test method. [3] However, the test data generated by the method is redundant due to the disregard of interaction between operations; And the importance and using frequencies of the data are also the factors that cannot be ignored. On account of the two problems above, this paper proposes a method which considers the interaction among different operations and the weight of operation data. At last, we perform experiments which show that the method proposed in this paper could reduce the redundancy of test data and improve the testing efficiency while guaranteeing the coverage ratio of test data. [4][5]

2. Usage model and related definition

2.1. Usage model
In this paper, the directed graph is used to represent the usage model. Figure 1 shows all the states during the running process of a UAS software with an initial state as start and an ending as end.

2.2. The interaction between operations and the input data types
In the actual software, as shown in figure 2, the input data of each step may have a lot of types. Also, as is mentioned before, some interactions usually exist between different operations or steps.
Considering these two factors (the diversity of input data types, the interaction between operations or steps), generating test data for each operation sequence is required. [6][7] And while making experiments in actual situation of software, we found that strong interactions usually only exist in adjacent operations, while weak or even none interactions exist in non-adjacent operations. As a result, it makes a more efficiency way to generate test data with only interactions between adjacent operations counted. [8][9]

**Figure 1.** Markov usage model for a UAS software.

| Initial state | Type 1 | Type 2 | Type 3 | Type 4 | Final state |
|---------------|--------|--------|--------|--------|------------|
|               | 0.25   | 0.5    | 0.25   | 1.0    |             |
|               | 0.25   | 0.4    | 0.5    | 1.0    |             |
|               | 1.0    | 0.5    | 0.2    | 1.0    |             |

**Figure 2.** One operation sequence from Markov usage model.

2.3. Mathematical definition

Definition 1. We define n-gram \((v_1, v_2, ..., v_n)\), \(v_i \in V_1, v_2 \in V_2, ..., v_n \in V_n\) as a testing sequence.

Definition 2. \(A = (a_{ij})_{mn}\) is a \(m \times n\) matrix, of which the \(j\)th column represents operation \(f_j\) of operation sequences. All of the elements in the \(j\)th column are taken from the collection \(V_j (j = 1, 2, ..., n)\).

Definition 3. Adjacent Matrix: In order to facilitate the handling of Markov model, the adjacent matrix \(G\) is usually used according to the Markov model, and \(G[i][j] = p_{ij}\). \(p_{ij}\) is the transition probability of the way from point \(i\) to point \(j\). \(G[i][j] = 0\) shows there is no access from point \(i\) to point \(j\).

Definition 4. Operation Sequence Weight: An operation sequence as \(k, i \in [1, p]\), the Weight of Operation Sequence \(k, i \in [1, p]\) is defined as \(\text{Weight}(k_i) = \prod p_{ij}\), with \(p_{ij}\) representing the transition probability of \(j\)th path in operation sequence.

3. Analysis for N-Dimensional adjacent algorithm

3.1. Outline of method
Based on the Markov model, this algorithm can generate test data and the weight of test data for each operation sequence in the usage model. In algorithm 1, the adjacent matrix G of Markov model is got to traverse all paths of the usage model and calculates the transition probability of each operation sequence.

3.2. Algorithm 1
Input: initial state: start; the final state: end; adjacent matrix G = \{the adjacent matrix of Markov model \}; initial path; initial weight
Output: K=\{all paths from start to end \}, Weight = \{ weight of all operation sequences \}
getPaths(start, end, G, path, weight) {
    hasFlag[start]=true;
    for(each node ‘i’ in adjacent matrix G) {
        if(G[start][i]==0 || hasFlag[i]) {continue;}
        if(i==end) {
            record the result;
            continue;
        }
        getPaths(i, end, G, path+”->”+i, weight*G[start][i]);
        hasFlag[i]=false;
    }
}

3.3. Algorithm 2
Input: operation sequence k, number of types in each operation number=\{a[1],…,a[n]\}, weight of every data type, dimension N.
Output: N-dimension coverage array A.
for(each element in array number){
    m[i]=number[i]*number[i+1]*...*number[i+N-1];
}
m=max\{m[i]\};
initial array A[m][n];
calculate the element of the first N columns of A;
for(every line of A[m][n]){calculate the weight of each test data;}
calculate the total weight of test data;
for(every line of A[m][n]){calculate the proportion for each test data;}

3.4. Test data evaluation
In order to evaluate whether the test data generated by the proposed method can reflect the actual operation situation accurately, GFI(Goodness of Fit Index) is imported as to quantify the accuracy.[10][11]
Definition 7: Defining
\[ GFI = \frac{\chi^2_{\text{independent}} - \chi^2_{\text{proposed}}}{\chi^2_{\text{independent}} - \chi^2_{\text{ideal}}} = \frac{\sum_{i=1}^{n} \left( \frac{b_i - a_i}{a_i} \right)^2}{a_i} \]
a_i represents the times of test data n actually occurs, and b_i represents the times that test data n is assigned. Assuming that the \( \chi^2_{\text{independent}} \) meaning there is no correlation is the maximum value of \( \chi^2 \).
\( \chi^2_{\text{ideal}} \) meaning the ideal condition with the best correlation with reality is the minimum value. The formula is modified as :
From the formula, we can see that the value of GFI is between 0 and 1.

4. Experiment

Experiments are made to compare the proposed method with the other methods, and the GFI between method and actual situation shows that the method proposed in this paper performs better than other methods when applied in operation sequence with strong interaction. Also, the coverage ratio shows that the proposed method could achieve good performance with less test data.

4.1. Analysis of GFI

| Sequence number | Method 1 | Method 2 | Method 3 | Reality |
|-----------------|----------|----------|----------|---------|
| 1               | 93       | 287      | 153      | 152     |
| 2               | 24       | 287      | 247      | 225     |
| 3               | 146      | 287      | 153      | 160     |
| 4               | 48       | 243      | 225      | 159     |
| 5               | 50       | 192      | 150      | 149     |
| 6               | 4        | 152      | 158      | 160     |
| 7               | 0        | 241      | 247      | 251     |
| 8               | 90       | 243      | 241      | 243     |
| 9               | 150      | 120      | 152      | 109     |
| 10              | 0        | 122      | 122      | 121     |
| 11              | 44       | 287      | 151      | 192     |
| 12              | 140      | 187      | 167      | 187     |
| 13              | 40       | 287      | 610      | 595     |
| 14              | 53       | 287      | 372      | 400     |
| 15              | 0        | 287      | 252      | 252     |
| 16              | 0        | 287      | 242      | 242     |
| 17              | 0        | 287      | 242      | 242     |
| 18              | 140      | 333      | 333      | 313     |
| 19              | 48       | 287      | 549      | 612     |
| 20              | 97       | 287      | 287      | 287     |
| 21              | 48       | 287      | 287      | 287     |
| 22              | 0        | 287      | 564      | 540     |
| 23              | 51       | 287      | 564      | 540     |
| 24              | 54       | 287      | 287      | 287     |
| 25              | 52       | 287      | 120      | 120     |
| 26              | 54       | 287      | 287      | 287     |
| 27              | 51       | 287      | 175      | 183     |
| 28              | 0        | 287      | 287      | 287     |
| 29              | 0        | 287      | 120      | 120     |
| 30              | 2        | 287      | 287      | 287     |
| 31              | 97       | 287      | 434      | 434     |
| 32              | 48       | 287      | 449      | 449     |
| total           | 1624     | 9184     | 9067     | 9086    |
| GFI             | 0        | 0.6328   | 0.96     | 1       |

In order to verify that the method proposed in this paper have better GFI, four groups of experiments are designed. In method 1, only use algorithm 1 is used to traverse the Markov model on purpose of...
getting all operation sequences, and test data for each path is generated randomly without input data diversity and the interaction between steps; In method 2: algorithm 1 and algorithm 2 are both used to generate test data for operation sequence with input data diversity and the interaction among operations considered, ignoring the operation sequence weight. In method 3: we take the operation sequence weight into consideration compared to method 2, and test data is assigned according to the weight; And group 4 is the actual usage of a software as the control group.

As shown in figure 2, taking path ‘Initial—A—B—E—D—Final’ as an example, the weight of the path is 0.093. Assuming that the total number of test data is 200000, and we can see that the actual number of test data generated by the method is 18600 according to the weight 0.093. And from the path, we can get the number of test data combination is 192. While according to the analysis before, only 32 kinds of combination of test data is actually needed when take the interaction between operations into consideration. The result is shown as table 1, in method 1, a lot of useless test data is wasted on the 160 invalid operation combination with the lowest efficiency. Method 2 performs better and wastes less test data when compared to method 1, with considering the interaction between adjacent operations. Obviously, method 2 is more efficient than method 1. And in method 3, by considering both the interaction among sequences and the weight of operation sequence, it assigns more test data to the operation sequence with higher weight and wastes less test data in invalid operation sequences, making it closest to reality.

As is mentioned before, GFI is imported to quantify the accuracy with reality. As shown in table 1, for method 1, ‘GFI = 0’ shows that method 1 divorces from reality seriously without considering the interaction between operations; for method 2, ‘GFI = 0.6328’ shows that method 2 can be partly consistent with the reality; for method 3, ‘GFI = 0.96’ shows that method 3 is basically in line with the reality.

4.2. Code coverage analysis
Coverage ratio is an important indicator for the quality of test data in software testing. In this section, we use the tool CTS to analyse the method coverage ratio and branch coverage ratio. The CTS is a code testing system developed by author’s laboratory supporting executing and analysing programs. CTS can feeds back a report that shows the method coverage ratio and branch coverage ratio of current program. At first, test data is generated by method 3, then execute the tested program with the test data in CTS and get the method coverage ratio and the branch coverage ratio. [12]

As shown in table 2, the test data generated by the proposed method could cover over ninety percent of total methods and eighty-eight percent of branches. This shows that the method proposed in this paper could achieve high method coverage ratio and branch coverage ratio with less test data and higher efficiency.

| Total methods | Covered methods | Total branches | Covered branches |
|---------------|-----------------|----------------|-----------------|
| 25            | 23              | 67             | 59              |
| Cover Ratio   | 92%             | Cover Ratio    | 88.1%           |

5. Related work
Model-based testing is widely used in software testing. Conformiq DesignerModels supports creating models as UML and QML state machines. [13] And results can be exported to other test management tools. BPM-Xchange can generate test cases from models based on different criteria, and it also supports importing models from other tools and exporting test cases to Excel. [14] The DTM (Dialogues Testing Method) tool could choose test data based on structural coverage and execute based on custom models. [15] fMBT (free Model-Based Testing) can generate test data from model which is written in Python, while also supporting several heuristic strategy. [16]

6. Conclusion and future work
In this paper, in order to solve the problem of test data redundancy in model-based testing, a method which takes the interaction between operations and the weight of operation into account is proposed from a practical point of view. And GFI and coverage ratio are imported to quantify the quality of the test data generated by the proposed method. At last, experiments are made on CTS to show the GFI and coverage ratio, of which the result shows that the method proposed in this paper performs better.

While applied to the software of which interaction only exists in adjacent operations, the proposed method performs well. However, on the other hand, while applied to the software of which interaction doesn’t only exists in adjacent operations, it may cause the degradation in performance. And we’ll improve this situation in future work.

References
[1] IEEE Reliability Society, “IEEE Recommended Practice on Software Reliability”, IEEE Std 1633-2008, New York, 27 June 2008.
[2] R.C.Cheung, “A user-oriented software reliability model”, IEEE transactions on software Engineering, SE-12(1):118–125, Mar. 1980.
[3] J.A.Whittaker and M.G.Thomason, “A Markov Chain Model for Statistical Software Testing,” IEEE Trans. Software Eng., Vol. 20, No. 10, pp.812–824, 1994.
[4] Liang Shi, Baowen Xu, Changhai Nie. Combinatorial Design Approaches for Automatic Test Generation. Journal of Electronics(China), 2005, 22(2): 205-208
[5] Anderson J A, Lorenz C D, Traveset A. General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units[J]. Journal of Computational Physics, 2008 227(10):5342—5359.
[6] Zhang D, Sui J, Gong Y. Large scale software test data generation based on collective constraint and weighted combination method[J]. Tehnicki vjesnik - Technical Gazette, 2017, 24(4).
[7] Kuhn R, Lei Y, Kacker R. Practical combinatorial testing: Beyond pairwise. IT Professional, 2008,10(3):19–23.
[8] Grindal M. Handling combinatorial explosion in software testing [Ph.D. Thesis]. Linköpings Universitet, 2007.
[9] Wang ZY, Nie CH, Xu BW, Shi L. Optimal test suite generation methods for neighbor factors combinatorial testing. Chinese Journal of Computers, 2007,30(2):200–211 (in Chinese with English abstract).
[10] Wang ZY, Nie CH, Xu BW. Generating combinatorial test suite for interaction relationship. In: Pezzé M, ed. Proc. of the 4th Int’l Workshop on Software Quality Assurance (SOQUA 2007). New York: ACM Press, 2007. 55–61.
[11] Yan J, Zhang J. A backtracking search tool for constructing combinatorial test suites. The Journal of Systems and Software, 2008, 81(10):1681–1693.
[12] Ma F, Zhang J. Finding orthogonal arrays using satisfiability checkers and symmetry breaking constraints. In: Ho TB, Zhou ZH, eds. Proc. of the 10th Pacific Rim Int’l Conf. on Artificial Intelligence (PRICAI 2008). LNAI 5351, Berlin/Heidelberg: Springer-Verlag, 2008. 247–259.
[13] P. Joshi, M. Naik, K. Sen, and D. Gay, "An effective dynamic analysis for detecting generalized deadlocks," in Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering, Santa Fe, New Mexico, USA, 2010, pp.327-336.
[14] Bryce RC, Colbourn CJ. The density algorithm for pairwise interaction testing. Software Testing, Verification and Reliability, 2007. 17(3):159–182.
[15] Meagher K, Moura L, Zekaoui L. Mixed covering arrays on graphs. Journal of Combinatorial Designs, 2007,15(5):393–404.
[16] Zekaoui L. Mixed covering arrays on graphs and tabu search algorithms [MS. Thesis]. Ottawa-Carleton Institute for Computer Science, University of Ottawa, 2006.