

Incremental covering array: two generation strategies

Xintao Niu and Changhai nie
State Key Laboratory for Novel
Software Technology
Nanjing University, China, 210023
Email: changhainie@nju.edu.com

Hareton Leung
Department of computing
Hong Kong Polytechnic University
Kowloon, Hong Kong
Email: cshleung@comp.polyu.edu.hk

Abstract—Combinatorial testing (CT) is an effective technique for testing the interactions of factors in the software under test (SUT). By designing an efficient set of test cases, i.e., covering array, CT ensures to check every possible validate interactions in SUT. Most existing covering array generation algorithms require to be given a *strength* t in prior, such that only the interactions with the number of factors no more than t are needed to be checked. In practice, however, such t cannot be properly determined, especially for systems with complicated interactions space. Hence, adaptively generation of covering arrays is preferred. In this paper, we proposed two strategies for generating covering arrays which can increase the coverage strength when required. An preliminary evaluation of the two strategies is given, which showed that, in consideration of the size of the covering array, both the two strategies have their own advantages.

I. INTRODUCTION

With the growing complexity and scale of modern software systems, various factors, e.g., input values and configure options, can affect the behaviour of the system. Worse more, some interactions between them can trigger unexpected negative effects on the software. To ensure the correctness and quality of the software system, it is desirable to detect and locate these *bad* interactions. The simplest way to solve this problem is to take exhaustive testing for all the possible validate interactions of the software system. It is, however, not impractical due to the interactions explosion. Therefore, to select a group of representative test cases from the whole testing space is required.

Combinatorial testing has been proven to be effective to deal with this sampling work [1]. It works by generating a relative small set of test cases, i.e., covering array, to test a group of particular interactions in the system. The number of factors involved in those selected interactions is limited in a moderate range, which is usually from 2 to 6 [2].

Many algorithms are proposed to generate covering arrays. Apart from many differences between them, those works have a common condition, i.e., they all need to be given a the strength t in prior. The strength t indicated the largest number of factors involved in the interactions they need to cover. In practice, however, such a t can not be properly determined. There are two reasons for this. First, many software systems suffer from complicated interactions space which make it

challenging to estimate the strength t . In such case, even experienced testers can make mistakes and estimate a wrong value for t , which can significantly affect the effectiveness and efficiency of CT. Second, even though t has been properly determined, there may be no enough time to completely executing all the test cases in the covering array. This is because testing for the software only makes sense before the next version is released. This time interval between two release versions may sometimes too short for a complete testing to a high strength covering array [3].

To make up for the shortcoming of traditional covering arrays, the incremental covering array [3] has been proposed. Such object can be deemed as adaptive covering array, which can increase the strength t when required. As it can generate higher strength covering array based on lower strength covering array, it can reduce much more cost when comparing with generating multiple different strengths of covering arrays. Additionally, it can better applied on the testing software of which the releasing time is frequently changed and cannot be predicted.

In consideration of the size of the overall test cases, we argue that this approach of generation of incremental covering array may generate too many test cases to construct the higher-strength covering array. This can be easily understood, as generating higher-strength covering array based on the lower-strength covering array (called *down-top* generation later) does not aim to optimize the size of the higher-strength covering array. As a result, it may generate more test cases than those generated by approaches that focus on generating particular higher-strength covering array.

And also the key idea of this paper : To generate higher strength covering array from the lower strength, may be not effective at the size of the overall needed test cases. This can be easily understood, as it is not focus on the higher covering array, which should . to in the .

Then a . In fact, to execute the test cases , not mean to generate them in the order.

Our contributions includes:

- 1) We propose two strategies for the generation of incremental covering arrays.
- 2) We conduct a series experiments to compare and evaluate the two strategies.

- 3) A recommendation for selecting which strategy when generating incremental covering array in practice is given.

II. BACKGROUND

This section gives some formal definitions in CT. Assume that the behaviour of SUT is influenced by k parameters, and each parameter p_i has a_i discrete values from the finite set V_i , i.e., $a_i = |V_i|$ ($i = 1, 2, \dots, k$). Then a *test case* of the SUT is a group of values that are assigned to each parameter, which can be denoted as (v_1, v_2, \dots, v_k) .

A. covering array

Definition 1. A *covering array* CA is a $N \times k$ table, with each row representing a test case of the SUT. each $t \times N$ sub-table contains each possible combination of values for the t parameters. Here we call

B. incremental covering arrays

Definition 2. incremental covering arrays is a sequence of covering arrays $(T_1, T_2, \dots, T_i, \dots, T_k)$, s.t., each $T_i \subseteq T_{i+1}$.

In practice .

III. MOTIVATION EXAMPLE

This section presents the example for . Consider we are testing a object, initially, we believe 1 way is enough, but when we give so, it find that is not , then we must increase the , to three way. In such a case, to take a Coverin array to regenerate 3-way covering array is follish, as it may introuduce many redunenta.t For example, many 3 way schemas in fact have been checked before.

Then it is of course we need to generate the coverin array based on the previous covering array, such that we can utilize the previous tesing results. For this, a natural idea, is to set the as the seeds, like this.

Now let us think the following problem, as the . If we had first generate the 3-way covering array, and .

need to . And hence then , we need to increase the strength, as . To . Then first, a natural

IV. GENERATING INCREMENTAL COVERING ARRAYS

V. PRELIMINARY EVALUATION

This section , on which there are 20 covering arrays . in Table.

VI. RELATED WORK

S.Fouché [3]

VII. CONCLUSIONS AND FUTURE WORKS

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (No. 61272079), the Research Fund for the Doctoral Program of Higher Education of China (No.20130091110032), the Science Fund for Creative Research Groups of the National Natural Science Foundation of China(No. 61321491), and the Major Program of National Natural Science Foundation of China (No. 91318301)

t1	0	0	0	0	0
t2	1	1	1	1	0
t3	0	0	1	1	1
t4	1	1	0	0	1
t5	0	1	0	1	0
t6	1	0	1	0	0

2-way

t7	1	0	0	1	1
t8	0	1	1	0	1
t9	0	1	1	0	0
t10	0	0	0	0	1
t11	1	0	0	1	0
t12	0	1	0	1	1
t13	1	0	1	0	1

3-way

t14	1	1	0	0	0
t15	0	0	1	1	0
t16	1	1	1	1	1
t17	0	0	1	0	0
t18	1	0	0	0	0
t19	0	1	0	0	0
t20	0	0	0	1	0
t21	1	1	1	0	0
t22	0	1	1	1	0
t23	1	1	0	1	0
t24	1	0	1	1	0

4-way

Fig. 1. Incremental covering arrays using AETG with seeds

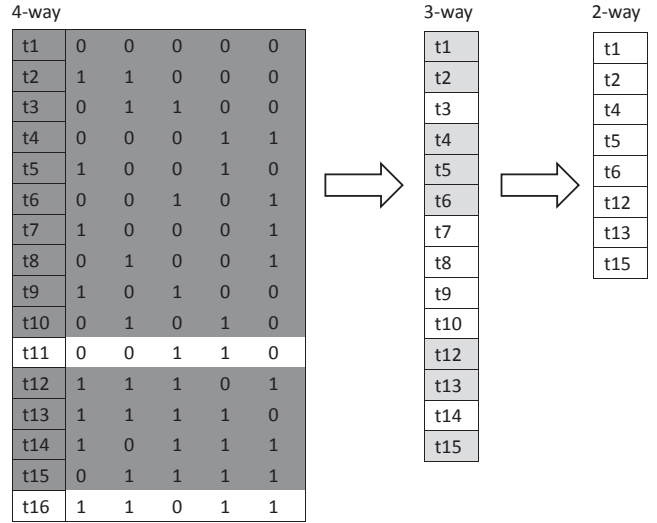


Fig. 2. Incremental covering arrays from a top-down sequence

REFERENCES

- [1] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 11, 2011.
- [2] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*. IEEE, 2002, pp. 91–95.
- [3] S. Fouché, M. B. Cohen, and A. Porter, "Incremental covering array failure characterization in large configuration spaces," in *Proceedings of the eighteenth international symposium on Software testing and analysis*. ACM, 2009, pp. 177–188.