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 has their own advantages.**

I. Introduction

With the growing complexity of software systems, various interactions of factors can affect the behaviour of the system. To test all the possible interactions of them is impractical due to the large interactions space. Therefore, efficient sampling of the whole testing space are required. Combinatorial testing has been proven to be effective to deal with this sampling work. It works by generating a relative small set of test cases, i.e., covering array, to test all the possible moderately low-stregn interactions in system. The justification behind of CT is that most failures in the system are caused by the interactions with number of involved factors no more than 6 [1]. And hence can fulfil testing requirement at most time.

Many works are proposed to generate covering arrays. Besides the behind these algorithms, there is a common for these work, i.e., they should be should given a the strength *t* in prior to guide the generation. In practice, however, such a *t* can not be propoer determined. There are two reasons for this, first, many software system has complicated interactions space which make a challenge to determine the maximal strenght. In fact, in the worst case, any interaction with any strength can have an distinct affect of the SUT. Second, even though the *t* has been properly determined, there may be no time to completedly executing all the test cases in the test cases. This is becasue, are distinct, some may [2]. In such a case, the is no meaning less.

For this two reasons, the adaptive covering array is prefered. Fouche in this paper proposed a incremantal coveing array. Which the higher can be based on several lower strength covering array.

In consideration of the size, . 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:

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,..k). Then a *test case* of the SUT is a group of values that are assigned to each parameter, which can be denoted as $(v_1, v_2, ..., v_k)$.

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 * 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, ... T_i,....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 utilze the previous tesing results. For this, a natural idea, is to set the as the seeds, like this.

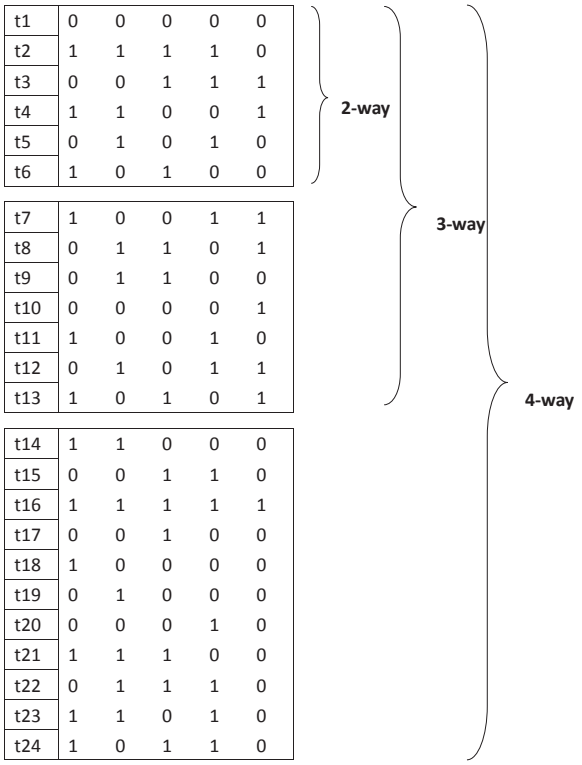

Fig. 1. Incremental covering arrays using AETG with seeds

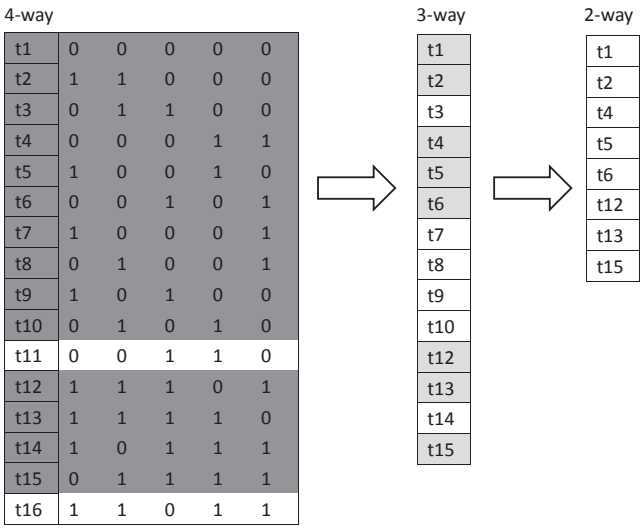

Fig. 2. Incremental covering arrays from a top-down sequence

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 [2]

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)

References

[1] 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.

[2] 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.