Error Locating Driven Array*

Xintao Niu
State Key Laboratory for Novel Software Technology
Nanjing University
China, 210023
niuxintao@gmail.com

Changhai Nie
State Key Laboratory for Novel Software Technology
Nanjing University
China, 210023
changhainie@nju.edu.cn

JiaXi Xu
School of Mathematics and Information Technology
Nanjing Xiaozhuang University
China, 211171
xujiaxi@126.com

ABSTRACT

Combinatorial testing(CT) seeks to handle potential faults caused by various interactions of factors that can influence the Software systems. When applying CT, it is a common practice to first generate a bunch of test cases to cover each possible interaction and then to locate the failure-inducing interaction if any failure is detected. Although this conventional procedure is simple and straightforward, we conjecture that it is not the ideal choice in practice. This is because 1) testers desires to isolate the root cause of failures before all the needed test cases are generated and executed 2) the early located failure-inducing interactions can guide the remaining test cases generation, such that many unnecessary and invalidate test cases can be avoided. For this, we propose a novel CT framework that allows for both generation and localization process to better share each other's information, as a result, both this two testing stages will be more effectively and efficiently when handling the testing tasks. We conducted a series of empirical studies on several open-source software, of which the result shows that our framework can locate the failure-inducing interactions more quickly than traditional approaches, while just needing less test cases.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and debugging—*Debugging aids,testing tools*

General Terms

Reliability, Verification

*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)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WOODSTOCK '97 El Paso, Texas USA
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

Keywords

Software Testing, Combinatorial Testing, Covering Array, Failure-inducing combinations

1. INTRODUCTION

Software system are growing more and more complexity, the testing is needed. Nowadays, the bugs are not the single factor-bug, that is, it is . To detect and isolate such a is hard, for we could not easily know which interaction of from so many. The first, how to detect, the second, even if we have detected, .

Combinatorial testing is a proposing testing technique to handle such problem. The testing object– covering array is a well designed test suite, which can cover each possible interaction with just a small or reasonable size. When one or more test cases trigger some fault, it take the isolating.

In CT it is a pri-critira that the test suite must be , and then using to trigger. This framework is straight and simple, however, we conjecture in this paper that this would be more redundant inspect of fault debugging. Through a series experiment on empirical study, we observe that, there is two main : right . wrong . This redundant can make the tester to test more unnecessary test cases, which is a wasting of computing resource.

So in this paper, we propose a new and really intuial CT fault detecting and isolating framework (CTDI), which combine the more tightly. In detail our framework, handles two main . For we find . we make it as constraint, and feedback to the generating. The second, as we can generate many extra test cases, for these passing test cases, we use them as seed to let them in the generating process. The ended criteria is some coverage is reached.

Our new CT is .

We have imply , and find our approach can .

2. MOTIVATING EXAMPLE

Combinatorial testing can effectively detect the failures caused by the interactions between various options or inputs of the SUT. Covering arrays, the test suite generated by this technique can cover each combination of the options at least once. We conjecture, however, although covering array can effectively, in practice, covering array was too much for detecting and locating the error in particular software.

As an motivating example, we looked through the following scenarios for detecting and locating the errors in the SUT.

Too much redundant fault test cases:

Too much redundant right test cases:

3. BACKGROUND

This section presents some definitions and propositions to give a formal model for the FCI problem.

3.1 Failure-inducing combinations in CT

Assume that the SUT is influenced by n 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,..n). Some of the definitions below are originally defined in .

Definition 1. A *test case* of the SUT is an array of n values, one for each parameter of the SUT, which is denoted as a n-tuple (v_1, v_2,...,v_n), where $v_1 \in V_1$, $v_2 \in V_2$ ... $v_n \in V_n$.

In practice, these parameters in the test case can represent many factors, such as input variables, run-time options, building options or various combination of them. We need to execute the SUT with these test cases to ensure the correctness of the behaviour of the software.

We consider the fact that the abnormally executing test cases as a *fault*. It can be a thrown exception, compilation error, assertion failure or constraint violation. When faults are triggered by some test cases, what is desired is to figure out the cause of these faults, and hence some subsets of this test case should be analysed.

Definition 2. For the SUT, the n-tuple (-,v_{n_1},...,v_{n_k},...)is called a k-value *combination* ($0 < k \leq n$) when some k parameters have fixed values and the others can take on their respective allowable values, represented as "-".

In effect a test case itself is a k-value *combination*, when k = n. Furthermore, if a test case contain a *combination*, i.e., every fixed value in the combination is in this test case, we say this test case *hits* the *combination*.

Definition 3. let c_l be a l-value combination, c_m be an m-value combination in SUT and $l < m$. If all the fixed parameter values in c_l are also in c_m, then c_m *subsumes* c_l. In this case we can also say that c_l is a *sub-combination* of c_m and c_m is a *parent-combination* of c_l, which can be denoted as $c_l \prec c_m$.

For example, in the motivation example section, the 2-value combination (-, 4, 4, -) is a sub-combination of the 3-value combination (-, 4, 4, 5), that is, (-,4,4,-) $\prec$ (-,4,4,5).

Definition 4. If all test cases contain a combination, say c, trigger a particular fault, say F, then we call this combination c the *faulty combination* for F. Additionally, if none sub-combination of c is the *faulty combination* for F, we then call the combination c the *minimal faulty combination* for F (It is also called Minimal failure-causing schema(MFS) in).

In fact, MFS and *minimal faulty combinations* are identical to the failure-inducing combinations we discussed previously. Figuring it out can eliminate all details that are irrelevant for causing the failure and hence facilitate the debugging efforts.

4. ALGORITHMS

4.1 Description

4.2 A case study

5. EMPIRICAL STUDIES

5.1 The existence of

5.1.1 Study setup

5.1.2 Result and discusssion

5.2 Performance of the traditional algorithms

5.2.1 Study setup

5.2.2 Result and discussion

5.3 Performance of our approach

5.3.1 Study setup

5.3.2 Result and discussion

5.4 Threats to validity

6. RELATED WORKS

7. CONCLUSIONS