Volume 4, Issue 4, April 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Literature Survey on Combinatorial Testing

Shekhar Udai
Galgotias University
Gr.Noida, Uttar Pradesh, India

Abstract-- A Software application has different configurations of hardware and software, or different types of input parameters and there values. Mutual interactions between these parameters and configurations may affect the working of the software. Errors occur, when the usage of the software increases and interaction between these parameters grows rapidly. Hence, it is logical to test all these parameters and their interactions with a test suit. Combinatorial testing (CT) is applied for finding errors which are triggered by the interaction of parameters. Most of the errors occur due to a single parameter or by the joint combinatorial effect (2-way interaction) of two parameters, with progressively fewer failures induced by interactions between three or more parameters. We are focused to get rid of all the errors that occur because of the interaction between two parameters and there values. Finding an optimal test suit with less number of test cases has been proven to be an NP-complete problem. This greedy approach provides an algorithm which tries to produce a minimal covering array for 2-way interaction and an optimized test suit. All the results that have been obtained through this algorithm (applying on different number of parameters and there values) has been discussed at the end.

Keywords – Test case optimization, NP-complete problem, t-way interactions, Combinatorial Testing, Greedy Algorithm, Minimal Covering Arrays.

I. INTRODUCTION

Different types of testing aims for identifying different types of errors and faults. For example, mutation testing modifies the source code in a meager way that helps the tester to develop effective test cases. Similarly, combinatorial testing is focused on identifying errors and faults, occurs due to the interaction of different parameters of software. Whenever we have a product that processes multiple variables that may interact with each other, we use combinatorial testing. The variables may come from a variety of sources, such as the user interface, the different operating system, peripherals, different databases, or from across a network. The task in combinatorial testing goes beyond testing individual variables (although that must also be done, as well). In combinatorial testing the task is to verify that different combinations of variables are handled correctly by the system. Researchers have proposed and still are proposing new, effective and efficient techniques to refine the existing methods of testing and to find possible ways of improving the testing activity.

II. BACKGROUND

In this section, we will give an overview of CT, including some formal notations and definitions commonly used in the literature. From the successive definitions of test suites for CT, we can understand the evolution of CT. We also review the test case generation methods for CT.

2.1 The State of CT Research

In this section, we present the research topics of CT one by one. By adopting the work of others into the typical testing lifecycle, we propose a generic procedure model of CT, as shown in Figure 2.1. The first step is to build a model of SUT (Section 2.1.1), and the second step generates a test suite for CT (Section 2.1.2) and makes prioritization (Section 2.1.3). After the third step of test execution, if we find bugs and can fix them, then we can conduct regression testing else, if we cannot succeed in the failure diagnosis, we can handle some further testing (Section 2.1.4). Finally, we collect some data and evaluate
the testing results (Section 2.1.5).

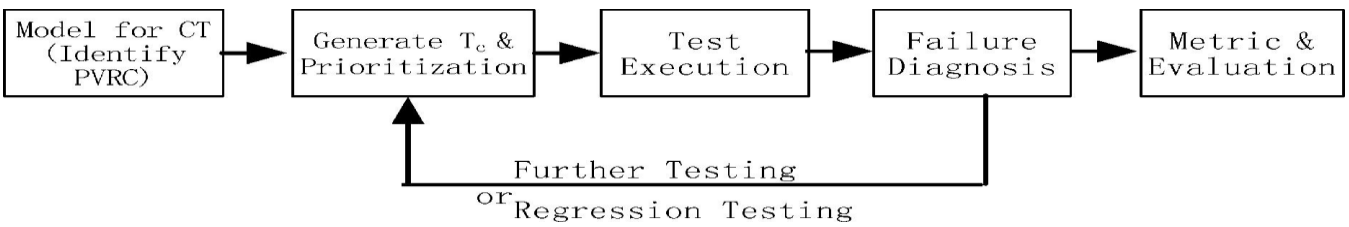

Figure 2.1: The testing procedures for CT [5]

Our testing procedure model is very similar to the common process of testing. However,to address the unique characteristics of CT, some special activities and techniques are 13 added. We will discuss the application and the testing procedure of CT together (Section 2.1.6). Based on the generic model (Figure 2.1), we will discuss each step one by one. For the sake of clarity, we will present constraints and test generation together in Section 2.1.3, and metric and evaluation together in Section 2.1.6. 2.1.1 Model of SUT Modeling of SUT is a very fundamental and important activity for CT, since the effectiveness and efficiency of CT depend on the model used. It is the starting point of CT. Different testers may come up with different models of SUT based on their own skill and experience. A key issue in modeling is to build a precise model at the "right" level of abstraction. We first give the definition of model of SUT, then review some approaches for modeling a SUT.

Example 2.1: Take an example of a Network Game Software (NGS) which may be influenced by the configuration parameters. In our example, $n = 4$, $a1 = a2 = a3 = a4 = 3$, $V1 = \{Netscape, IE, Fire\ fox\}$, $V2 = \{Windows, Linux, Macintosh\}$, $V3 = \{ISDL, Modem, PN\}$, and $V4 = \{Creative, Digital, Maya\}$. $R = 2C/ Ø = \{\{c1, c2, c3, c4\}\}$, where $C = \{ c1, c2, c3, c4\}$, that is, here all the parameters and their combinations may affect the NGS. We will take this example for the following discussion.

Definition 2.1: *SUT Mode:* For CT, a model of SUT includes four elements: parameters that may affect the SUT, values that should be selected for each parameter, interaction relations that exist between parameters and constraints that exist between values of the different parameters, which can be used to exclude combinations that are not meaningful from the domain semantics. A model for the SUT can be denoted as a 4-tuple: *ModelSUT* (*P, V, R, C)*. By Definition 2.1, four key issues need to be resolved in modeling of SUT:

- How to identify parameters that may affect the SUT?
- What values should be selected for each parameter?
- What interactions exist between parameters?
- What constraints exist between parameters and their values?

Only when these issues are settled then CT can effectively brought into play. The first issue is to identify parameters for CT. Parameters in the SUT may represent configuration parameters, like those of the NGS in example 2.1, and similar cases found in Cohen et al. 14 [10],Williams [37], and Xu et al. [41]. They may also represent user input parameters [32], features of the SUT [11], interfaces or GUI [7, 14], and components or objectives [12]. By reviewing various system documents and conducting some case studies [23], we can select an initial set of parameters for the SUT and then refine this set by adding or deleting parameters during the modeling procedure. The second issue is to determine the applicable values for each parameter. This step is critical to the modeling of SUT, since the behavior of the SUT is governed by the specific combination of parameter values. Note that some parameters may have many values. For example, the parameter "Audio" in the NGS, a configure parameter, can have many possible values, as there are hundreds of audio products in the market. When the parameter is an input of continuous values, it can have an infinite number of values. In this case, we must select some typical values. Equivalence partitioning, boundary value analysis, and primary element selection are the usual methods to be used here. The third issue is to identify the actual interactions between parameters. We can study the system documents to identify:

- Parameters which will not interact with any other parameters.
- Parameters which may have strong interactions with each other.
- Interactions which exist between a small numbers of parameters.

The fourth issue is to identify the constraints that exist between certain parameters. A common case occurs when some specific values of one parameter conflict with some values of another parameter. For example, the Browser parameter of the NGS cannot take on "Firefox" value when the Access parameter has 'ISDL" value. This represents a kind of mutual exclusion constraint between parameters. The opposite case may also occur when some specific value of one parameter must be combined with some value of other parameters. For example, the Browser parameter of the NGS must take on "IE" value when the Access parameter has "ISDL" value or when the Audio parameter has "Maya" value. To obtain the information on the interactions and constraints between parameters, we can study the requirement document, design document, codes, and other related documents. We can also interview designers and programmers to extract the relevant information. Static analysis, such as program slicing, can also be used to identify the interactions between parameters as suggested by Schroeder and Korel [33, 34]. 15 There have been many studies on modeling of SUT which helped to advance this field. For example, Dalal et al. [15, 16] learned that iteration and expert knowledge is required to build a proper model. Lott et al. [21] gave the samples of modeling system which can serve as a tutorial for applying CT. There have been some good guidelines for modeling of SUT. Many applicable scenarios have been given in Williams [41], and the procedure for determining the parameters and their values, and generating test suite is given in Krishnan et al. [27]. Mats and Offutt [23] presented a basic eight step process for input parameter modeling and gave some initial experience of using this process. Czerwonka [14] gave some practical ways of modeling that make the pure pairwise testing approach more applicable.

Despite these prior works, there are still several open questions left in this area:

- How to effectively and efficiently model SUT for CT?
- How to validate the model?
- How to evaluate the effectiveness of the different approaches of modeling?

2.1.2 Test Case Generation Technique:

Test case generation is the most active area of CT research. As the problem of covering array generation is NP-hard, researchers have tried various methods to solve it. Till today, four main groups of methods have been proposed: greedy algorithm, heuristic search algorithm, mathematic method, and random method. The first two groups are computational approaches.

2.1.2.1 Greedy Algorithm: Greedy algorithms have been the most widely used method for test suite generation for CT. They construct a set of tests such that each test covers as many uncovered combinations as possible. There are two classes of greedy algorithms. The first class is the one-row-at-a-time variation based on the automatic test case generator AETG [11]. Bryce et al. [2] presented a generic framework for this class of algorithms. A single row of the array is constructed at each step until all T -sets have been covered. Algorithms that fit into this class include: the heuristic algorithm used to generate pairwise test suite of the CATS tool [28], the density-based greedy algorithm for generating a 2-way and higher strength covering array proposed by Bryce and Colbourn [2, 5, 6], and the greedy algorithm with different heuristics used in PICT in 2006. Tung and Aldiwan [36] also 16 gave a greedy algorithm for a parametric test case generation tool that applies a combinatorial design approach to the selection of candidate test cases. The second class of greedy algorithm is the In Parameter Order (IPO) algorithm. This class of algorithm begins by generating all T -sets for the first T factors and then incrementally expands the solution, both horizontally and vertically using heuristics until the array is complete.

2.1.2.2 Heuristic Search Based Algorithm: Heuristic search techniques such as hill climbing, great flood, tabu search, and simulated annealing have been applied to T-way covering array and VCA generation. Also some AI-based search techniques such as Genetic Algorithm (GA) and Ant Colony Algorithm (ACA) have also been used. Heuristic search techniques start from a preexisting test set and then apply a series of transformations to the test set until it covers all the combinations. Ghazi [18] used a GA-based technique that identifies a set of test configurations that are expected to maximize pairwise coverage with a predefined number of test cases. Bryce and Colbourn [5] augmented the one-test-at-a-time greedy algorithm with a heuristic search, which can generate tests that have a high rate of T - tuple coverage. This approach combines the speed of greedy methods with the slower but more accurate heuristic search techniques. The hybrid approach seems to achieve a more rapid convergence of T -tuple coverage than either greedy or heuristic search alone. They compared four heuristic search techniques and found that hill climbing is effective only when time is severely constrained, but tabu search, simulated annealing, and the great flood perform better over a longer time. Cohen et al. [12] reported using the computational method of simulated annealing to generate 3-way covering array and variable strength array. Shiba et al. [29] also used the genetic algorithm and ant colony algorithm to generate the 3-way covering array. Heuristic search techniques can often produce a smaller test set than those from the greedy algorithm, but typically require a longer computation.

2.1.3 Test Case Generation Tools:

More than 20 software tools have been developed for test case generation, for example, CATS (Constrained Array Test System) developed by Sherwood et al. [23], OATS (Orthogonal Array Test System) developed by Phadke et al. [6], AETG developed by Cohen et al. [11], IPO developed by Lei and Tai [27], T-config developed by Williams [38], CTS (Combinatorial Test Service) developed by IBM, and PICT developed by Microsoft [14]. Some new tools are still continually appearing. Many of these tools are freeware. As each tool has its own characteristics and advantages, none is the best for all settings. If possible we may use them together, and then choose the best result. Some integrated tools have been developed to generate a minimal test suite.

2.1.4. Test Case Prioritization

Test case prioritization has been well studied. The result of the prioritization is often a schedule of test cases so that those with the highest priority, according to some criterion, are executed earlier in testing. When testing is terminated after running a subset of the prioritized test suite, those test cases deemed most important are executed. Especially when the resource is limited, the important test cases should be tested as early as possible. Well-ordered testing may reveal faults early because it ensures that the important test cases are executed first. The test case prioritization problem is defined as follows.

Definition 2.2: Given (T, Π, f), where T is a test suite, Π is the set of all test suites obtained by permuting the tests of T, and f is a function from Π to real numbers, the test case prioritization problem is to find $\pi \in \Pi$ such that $\forall \pi' \in$, $f(\pi) \geq f(\pi')$. Π gives thepossible prioritization of T and f is the function to evaluate the prioritization as suggested by Bryce and Memon [4]. Prioritization can be computed in two ways: (1) reorder an existing test suite of CT based on prioritization criteria; and (2) generate an ordered test suite for CT, taking into account the importance of combinations. By Definition 2.2, the key issues in test prioritization are how to define the function f, and validate the effectiveness of the prioritization. Bryce and Colbourn [3] adapted a one-test-at-a-time greedy method to take importance of parameter pairs

into account. With due consideration of seeding and constraint, they used this method to generate a set of tests in order. Bryce and Memon [4] explored the effectiveness of the prioritization with 18 interaction coverage in the event-driven software. Compared with other criteria like ordering test by unique event coverage, ordering test by the length, and random ordering, prioritization with interaction coverage was found to provide the fastest rate of fault detection. Different prioritization methods have different performance. Better methods of test case prioritization for CT may be developed by considering additional factors. For example, we may do prioritization based on required execution time or based on incremental *T*-way coverage The prioritization done on GUI applications was also applied to Web application[30].We also need more empirical study to better understand the limitations and strengths of various proposed methods [10].

2.1.5 Failure Diagnosis

After detecting a failure, we need to investigate the failure to locate and then remove the fault. Kuhn and Reilly [10] suggested that software faults are often triggered by only a few interacting variables. Their results have important implications for CT. If all faults in SUT are triggered by a combination of no more than *n* parameters, then testing all *n*-way combinations of parameters can provide a high confidence that nearly all faults have been discovered. So CT is an effective approach for detecting failures triggered by the combination of parameter values. When a defect is exposed in CT, we should determine which specific combination of parameter values causes the failure, or which value schema of SUT triggers the failure.

2.1.6 Metric and Evaluation

There exist two kinds of evaluation of CT that are

- Measurement and evaluation CT itself
- Measurement and evaluation of the quality of the SUT after CT.

CT has a natural metric, combination coverage, which measures the percentage of the covered parameter value combinations relative to the total combinations. One way to measure the combination coverage is based on the *k*-value schemas tested relative to the total *k*-value schemas.

2.2 The Application of CT and Testing Procedure

Since 1980 combinatorial testing has been applied in various fields and applications. For example, ADA compiler was first tested by combinatorial coverage in 1985 by R. Mandl 19 [6]. Brownlie et al. tested PELMOREX (Weather forecast generator) by pairwise combinatorial coverage in 1992 and developed the OATS (Orthogonal Array Testing Strategy) system for test case generation. Many papers have been published that share the experience of using CT in practice. Huller [19] in year 2000 uses CT to test the ground system of satellite communication. Borroughs, Jain and et al. [1] shows in their paper that, how CT improves the quality and efficiency of internet protocol testing. Williams and Probert [4] explained in 1996 that how CT can be used to test network interface has been also used in other application too. White and Almezen [11] in year 2000 proposed a method for testing GUI objects. Empirical study of this method shows that, if we follow CT strategies then a less number of test cases can also detect the defects of the GUI. Burr and Young [7] generated test cases with the help of CT to test the email system with AETG. Configuration testing and browser testing were also evolved as the child of CT.

REFERENCES

[1] D. R. Kuhn, N. Kacker, Yu Lei, "Practical Combinatorial Testing", NIST Special Publication Oct.2010.

[2] R. Othman and K. Zamli, "T-Way Strategies and Its Applications for Combinatorial Testing", International Journal on New Computer Architectures and Their Applications (IJNCAA): Pages 459-473, 2011.

[3] P. Flores and Y. Cheon, "Generating Test Cases for Pairwise Testing Using Genetic Algorithms", 18th IEEE International Symposium on Software Reliability Engineering (ISSRE'07), Dec.2007.

[4] B. Chen, J. Yan2 and J. Zhang, "Combinatorial Testing with Shielding Parameters", Asia Pacific Software Engineering Conference, 2010

[5] M. Grindal, J. Offutt, S.F. Andler, "Combination testing strategies: a survey Software Testing", Verification and Reliability Volume 15 Issue 3, Pages 167 – 199, 2005.

[6] Renée C. Bryce, Yu Lei and D.R Kuhn, DOI:10.4018/978-1-60566-731-7.ch014.

[7] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", IEEE Transactions On Software Engineering, volume 23, number 7, July 2008.

[8] C. Nie and H. Leung, "A survey of combinatorial testing", 21st International Symposium ACM Computing Society, Pages 11-29, Jan.2011.

[9] L. White, H. Almezene, "Generating test cases for GUI responsibilities using complete interaction sequences", 11th International Symposium on Software Reliability Engineering (IISRE 2000), IEEE Computer Society, Pages 110-121.

[10] R. Mandl, "Orthogonal Latin squares: An application of experiment design to compiler testing", Community ACM 28, 10, Pages 1054–1058, 1985.

[11] J. Huller, "Reducing time to market with combinatorial design method testing", of the International Council on Systems Engineering (INCOSE) Conference, 2000.

[12] K. Borroughs, A. Jain and et al, " Improved quality of protocol testing through techniques of experimental design", In Proceedings of the IEEE International Conference on Record, Serving Humanity Through Communications.' Vol. 2, Pages 745–752, 1994.

[13] A. Williams and R.L Probert, "A practical strategy for testing pair-wise coverage of network interfaces", 7th International Symposium on Software Reliability Engineering (ISSRE'96), IEEE Computer Society, Page 246, 1996.

[14] K. Burr, W. Young, "Combinatorial test techniques: Table-Based automation, test generation, and code coverage", International Conference on Software Testing Analysis and Review, Pages 503–513, 1998.

