XX IMEKO World Congress
Metrology for Green Growth
September 9–14, 2012, Busan, Republic of Korea

COMBINATORIAL TESTING FOR SOFTWARE USED IN METROLOGY

Raghu Kacker[1], *D. Richard Kuhn*[1], *Yu Lei*[2], *James Lawrence*[1, 3]

[1]U.S. National Institute of Standards and Technology
[2]University of Texas at Arlington
[3]George Mason University in Virginia
100 Bureau Dr., Gaithersburg, MD 20899-8910, USA

Abstract: Most modern instruments of measurement for science, engineering, and commerce have embedded software. Also, software is required for mathematical computations. Therefore testing, verification, and validation of software used in metrology is important. Combinatorial testing is a versatile methodology which could be useful for many testing situations. It is based on the insight that while the behaviour of a software system may be affected by a large number of factors, only a few factors are involved in a failure inducing fault. This paper is an introduction to combinatorial testing as an adaptation of design of experiment methods for testing software.

Keywords: Covering Arrays**;** Design of Experiments Metrology; Orthogonal Arrays; Software Engineering.

1. INTRODUCTION

A variety of testing methods are employed to verify that the software used in metrology work properly. A useful approach is *dynamic testing* in which the software system under test (SUT) is exercised for a set of test cases, the expected behaviour of the system is predetermined for each test case, and the actual behaviour is compared against the expected. The SUT passes a test case when the behaviour is as expected and fails otherwise. When the SUT fails for one or more test cases the underlying faults which induce the failure are searched and then corrected. *Combinatorial testing* is a type of dynamic testing in which various test factors each with discrete test settings are identified and then each test case is expressed as a combination of one test setting for every test factor [Mathur (2008)]. There are two basic challenges in combinatorial testing. (1) How to specify the test factors and their test settings? (2) How to determine a small number of test cases which are sufficient to identify the failure inducing combinations of test settings? This short paper relates to the second challenge only.

Combinatorial testing is based on the insight that while the behaviour of a system may be affected by a large number of factors, only a few factors are involved in a fault. Combinatorial testing began as *pairwise testing* in which first orthogonal arrays and then covering arrays were used to make sure that all test settings of each factor and all pairs of the test settings were tested. In section 2, we discuss pairwise testing. Subsequent investigations of actual software failures showed that pairwise (2-way) testing may not always be sufficient. In addition certain combinations of the test settings may not be valid. In section 3, we discuss combinatorial t-way testing (CT) for $t \geq 2$ with support of constraints to exclude invalid combinations. Combinatorial testing for $t \geq 2$ is now practical because efficient tools for generating test suites for t-way testing with support of constraints have become available. Brief summary and concluding remarks appear in section 4.

2. PAIRWISE TESTING OF SOFTWARE

Orthogonal arrays (OAs) are mathematical arrangements of symbols which satisfy certain combinatorial properties [Hedayat et al (1999)]. The matrix shown in table 1 is an orthogonal array (OA) referred to as OA(8, $2^4 \times 4^1$, 2). The first parameter (which is 8) indicates the number of rows and the second parameter (which is $2^4 \times 4^1$) indicates that there are five columns of which four have 2 distinct elements each, denoted here by {0, 1}, and one has 4 distinct elements, denoted here by {0, 1, 2, 3}. The third parameter indicates that this OA has *strength* 2, which means that every set of two columns contains all possible pairs of elements exactly the same number of times. Thus every pair of the first four columns contains each of the four possible pairs of elements {00, 01, 10, 11} exactly twice and every pair of columns involving the fifth column contains each of the eight possible pairs of elements {00, 01, 02, 03, 10, 11, 12, 13} exactly once. In an orthogonal array of strength t, every set of t columns contains all possible t-tuples of elements *exactly the same number of times*. An electronic library of known OAs is maintained by Sloan (webpage).

Covering arrays (CAs) are a generalization of OAs. In a covering array of strength t, every set of t columns contains all possible t-tuples of elements *at least once* [Sloan (1993), Lawrence et al (2011)]. The first six rows of table 1 form a covering array of six rows, five columns (of which four have 2 distinct elements each and one has 3 distinct elements), and strength 2 referred to as CA (6, $2^4 \times 3^1$, 2). Methods for constructing CAs can be put in three categories: algebraic methods, meta-heuristic methods (such as simulated annealing and tabu search), and greedy search methods. Algebraic methods apply only to certain special combinatorial structures; however, when they apply they are extremely fast mathematical techniques and may produce CAs of smallest possible size. Meta-heuristic methods are

computationally intensive and they have produced some CAs of the smallest size known. Greedy methods are faster than meta-heuristic methods; they apply to arbitrary structures but may or may not produce smallest size CAs. Descriptions of these methods and references can be found in Lawrence et al (2011). Colbourn (webpage) maintains a web page of smallest known sizes of covering arrays of strength t up to seven. Available webpages of smallest known CA tables include the following: Forbes (webpage), Nurmela (webpage), and Torres-Jimenez (webpage).

The term *design of experiments* (DoE) refers to a methodology for conducting controlled experiments in which a system is exercised in a purposeful (designed) manner for chosen test settings of various input variables (called *test factors*) and the corresponding values of one or more output variables (called *system responses*) are measured to generate information for improving the performance of a class of systems. Conventional DoE methods were developed in the 1920s largely by Fisher (1935) and his contemporaries and followers to improve agricultural production. Later DoE were adapted for experiments with animals, clinical trials, and to improve manufacturing processes subject to uncontrolled variation. Frequently, the effects of many test factors each having multiple test settings are investigated at the same time and the DoE plans satisfy relevant combinatorial properties [Cochran and Cox (1950), Kempthorne (1952), Snedecor and Cochran (1967), Box, Hunter, and Hunter (1978), and Montgomery (2004)].

In the 1980s, along with the advent of computers and telecommunication systems based on software, the problem of testing software and software embedded systems became important. Taguchi (1987) inspired the use of OAs (of strength 2) as templates for generating suites of test cases for software testing. The test factors are associated with the columns of an OA and the elements in the columns are replaced with the test settings of respective test factors. Then the rows of the OA provide a suite of test cases for pairwise testing. The combinatorial property of OAs assured that all test settings of each test factor and all pairs of test settings of every pair of test factors were tested [Brownlie et al (1992)]. Thus began pairwise (2-way) testing of software systems.

Table 1: Orthogonal array OA(8, $2^4 \times 4^1$, 2)

	1	2	3	4	5
1	0	0	0	0	0
2	1	1	1	1	0
3	0	0	1	1	1
4	1	1	0	0	1
5	0	1	0	1	2
6	1	0	1	0	2
7	0	1	1	0	3
8	1	0	0	1	3

Pairwise testing is a type of dynamic testing in which the SUT is exercised for a test suite of test cases which satisfies the property that for every pair of test factors all possible pairs of the discrete test settings are tested (at least once). Pairwise testing is an economical alternative to exhaustive testing of all combinations of the test factors. For example, if we had 9 test factors with the combinatorial test structure $3^3 4^4 5^2$ then exhaustive testing would require 172 800 tests. However pairwise testing requires only 29 test cases, a dramatically smaller number than required for exhaustive testing [Kuhn et al (2010)]. Thus pairwise testing could greatly improve the efficiency of software testing. Many faults in software involve only one or two factors; therefore, pairwise testing was found to be effective. The earliest papers on pairwise testing of software include Mandl (1985) and Tatsumi (1987). Tatsumi (1987) is a signal paper which included two important insights. (1) In software testing all combinations need not be tested the same number of times, only each combination needs to be tested at least once. (2) In generating test suites for testing software invalid combinations must be excluded.

OAs as templates for generating test suites for software testing had two serious limitations [Sherwood (1994)]. (1) Often, an OA matching the required combinatorial test structure does not exist. For example, an OA of strength 2 matching the test structure $2^4 \times 3^1$ does not exist. However, OA(8, $2^4 \times 4^1$, 2) can be used to construct a pairwise test suite for the test structure $2^4 \times 3^1$. (2) Frequently, OA based test suites included invalid pairs of test settings. For example, suppose the five test factors of combinatorial test structure $2^4 \times 3^1$ were (1) operating system (OS) with two test settings {XP, Linux}, (2) browser with two test settings {Internet Explorer (IE), Firefox}, (3) protocol with two test settings {IPv4, IPv6}, (4) CPU type with two test settings {Intel, AMD}, and (5) database management system with three test settings {MySQL, Sybase, Oracle}. Since the browser Internet Explorer (IE) does not run on the OS Linux, the pair (Linux, IE) is invalid. An OA based test suite may not exclude the pair (Linux, IE).

The connection between software testing and CAs was elucidated by Dalal and Mallows (1998). CAs are better suited than OAs for generating test suites for software testing. (1) CAs can be constructed for any combinatorial test structure. (2) For a combinatorial test structure if an OA exists then a CA of the same or fewer test runs can always be obtained. (3) CAs can be constructed for any required strength (t-way) testing, while OAs are generally limited to strength 2 and 3 [Sloan (webpage)]. (4) In constructing combinatorial test suites based on CAs invalid combinations can be excluded.

3. COMBINATORIAL TESTING OF SOFTWARE

A team of NIST researchers investigated recall data from the FDA due to failures of software embedded in medical devices and failure reports for a browser, a server, and a database system from NASA to generate insights into the

kinds of testing that could have detected the underlying faults [Kuhn et al (2004)]. As a by-product of these investigations researchers determined the numbers of individual factors that were involved in the faults underlying actual failures. It turned out that only a few factors (out of many that affected the behaviour of a system) are involved in a failure inducing fault. Most failures are induced by single factor faults or by the joint combinatorial effect (interaction) of two factors. Progressively fewer failures are induced by interactions between three, four, or more factors. The maximum degree of interaction in actual faults so far observed is six.

Combinatorial *t*-way testing (CT) is a type of dynamic testing for software in which the SUT is exercised for a test suite of test cases which satisfies the property that for every subset of *t* test factors (out of all *k* factors, where $k \geq t$) all relevant *t*-tuples (excluding invalid combinations) of the test settings are tested at least once. When the SUT fails for one or more test cases, the pass/fail data are used to isolate the failure-inducing combinations of the test settings. CT methods can significantly improve the efficiency of software testing. An alternative to CT is *random testing* in which test cases are formed from random draws of the discrete test settings of the test factors. Generally combinatorial testing requires fewer test cases than random testing for equivalent coverage [Kuhn et al (2009)]. Combinatorial (*t*-way) testing may be regarded as an adaptation of the DoE methods for testing software systems because in both cases information about a system is gained by exercising it and the test suite (DoE plan) satisfies relevant combinatorial properties. The choice of test factors and their test settings defines and limits the scope of the combinations that are tested. Clearly, faults involving factors and settings which are not chosen for testing may not be exercised and revealed [Tatsumi (1987)]. Methods for specification of test factors and their test settings are largely application domain specific and a subject of continuing research.

Combinatorial testing is a versatile methodology which could be useful in a broad range of testing situations. CT can be used for (1) testing various configurations of a system (configuration testing) and for (2) testing various possible inputs to the system (testing input space) [Mathur (2008)]. CT can also be used for testing (1) data-bases and (2) state models [Sherwood (2011)]. We have investigated use of CT for the following applications. (1) Testing concurrent systems [Lei et al (2007)], (2) Testing web applications [Wang et al (2008)], (3) Testing of access control implementations [Hu et al (2008)], (4) Navigation of dynamic web structures [Wang et al (2009)], (5) Analyzing system state-space coverage [Maximoff et al (2010)], (6) Detecting deadlocks for varying network configurations [Kuhn et al (2009)], (7) Detecting buffer overflow vulnerabilities [Wang et al (2011)], (8) Conformance testing for standards [Montanez-Rivera et al (2012)].

We have developed a tool called ACTS for generating *t*-way combinatorial test suites based on CAs for arbitrary combinatorial test structures and any strength *t* with support of constraints. The tool ACTS is a freely distributed research tool and is downloadable from a NIST webpage [Kuhn et al (webpage)]. ACTS is one of many (free and commercial) tools for combinatorial *t*-way testing for $t \geq 2$ [Czerwonka (webpage)].

Special features of the ACTS tool include the following. (1) ACTS excludes those combinations of the test settings which are invalid according to the user specified constraints. (2) ACTS supports two test generation modes: scratch and extend. The former builds a test suite from the scratch, whereas the latter allows a test suite to be built by extending a previously constructed test suite which can save earlier effort in the testing process. (3) ACTS supports construction of mixed-strength test suites. For example, in a situation with 10 test factors all could be covered with strength 2 and a particular subset of 4 (which are known to be inter-related) could be covered with higher strength 4. (4) ACTS verifies whether a test suite supplied by the user covers all *t*-way combinations. (5) ACTS allows the user to specify expected output for each test case in terms of the number of output parameters and their values. (6) ACTS supports three interfaces: a Graphical User Interface (GUI), a Command Line Interface (CLI), and an Application Programming Interface (API). The GUI interface allows a user to perform most operations through menu selections and button clicks. The CLI interface can be more efficient when the user knows the exact options that are needed for specific tasks. The CLI interface is also very useful for scripting. The API interface is designed to facilitate integration of ACTS with other tools.

4. SUMMARY AND CONCLUDING REMARKS

Combinatorial *t*-way testing began as pairwise (2-way) testing in which the SUT is exercised for a test suite of test cases which satisfies the property that for every pair of test factors all possible pairs of the test settings are tested at least once. Investigations of actual faults indicate that while pairwise testing is useful, it may not always be adequate. Therefore combinatorial *t*-way testing for *t* greater than 2 may sometimes be needed. Combinatorial *t*-way testing for $t \geq 2$ is now possible because efficient and free downloadable tools for generating test suites for *t*-way testing with support of constraints have become available. Combinatorial testing is a versatile methodology which could be useful in a broad range of testing situations including verification of software used in metrology.

Disclaimer: NIST does not recommend or endorse any commercial product referenced in this paper or imply that the referenced products are necessarily the best available for the purpose.

5. REFERENCES

[1] George E. P. Box, W. G. Hunter, and J. Stuart Hunter (1978) Statistics for Experimenters, New York: Wiley.

[2] R. Brownlie, J. Prowse, and M. S. Phadke (1992) "Robust testing of AT&T PMX/Starmail using OATS," AT&T Technical Journal, 71, pp 41-47.
[3] William G. Cochran and M. G. Cox (1950) Experimental Designs, New York: Wiley.
[4] Charles J. Colbourn (webpage) http://www.public.asu.edu/~ccolbou/src/tabby/catable.html
[5] Jacek Czerwonka (webpage) http://www.pairwise.org/
[6] Siddhartha R. Dalal, and C. L. Mallows (1998) "Factor-covering designs for testing software," Technometrics, 40, pp 234-243.
[7] R. A. Fisher (1935) The Design of Experiments, Edinburgh: Oliver and Boyd.
[8] Michael Forbes (webpage) http://math.nist.gov/coveringarrays/
[9] A. S. Hedayat, N. J. A. Sloan, and J. Stufken (1999) Orthogonal Arrays: Theory and Applications, New York: Springer.
[10] Vincent C. Hu, D. R. Kuhn, and Tao Xie (2008) "Property verification for generic access control models," Proceedings of IEEE/IFIP International Symposium on Trust, Security and Privacy for Pervasive Applications (TSP 2008), Shanghai, China, 17-20 December 2008, pp 243-250.
[11] Oscar Kempthorne (1952) Design and Analysis of Experiments, New York: Wiley.
[12] D. Richard Kuhn, D. R. Wallace, and A. J. Gallo, Jr. (2004) "Software fault interactions and implications for software testing," IEEE Transactions on Software Engineering, 30, pp 418-421.
[13] D. Richard Kuhn, R. N. Kacker, Yu Lei (2009) "Random vs. combinatorial methods for discrete event simulation of a grid computer network," Proceedings of Modelling and Simulation World, NASA CP-2010-216205, National Aeronautics and Space Administration, Virginia Beach VA, 14-17 October 2009, pp 83-88.
[14] D. Richard Kuhn, R. N. Kacker, and Yu Lei (2010) Practical Combinatorial Testing, NIST Special Publication 800-142, US Government Printing Office, Mail: Stop SSOP, Washington DC 20402-0001 (http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf).
[15] D. Richard Kuhn et al (webpage) http://csrc.nist.gov/groups/SNS/acts/index.html
[16] James F. Lawrence, R. N. Kacker, Yu Lei, D. R. Kuhn, and M. Forbes (2011) "A survey of binary covering arrays," The Electronic Journal of Combinatorics, 18, P84.
[17] Yu Lei, R. Carver, R. N. Kacker, and D. Kung (2007) "A combinatorial strategy for testing concurrent programs," Journal of Software Testing, Verification, and Reliability, 17, pp 207-225.
[18] Robert Mandl (1985) "Orthogonal Latin squares: an application of experiment design to compiler testing," Communications of the ACM, 28, pp 1054-1058.
[19] Aditya P. Mathur (2008) Foundations of Software Testing, Boston: Addison-Wesley.
[20] Joshua R. Maximoff, M. D. Trela, D. R. Kuhn, and R. N. Kacker (2010) "A method for analyzing system state-space coverage within a *t*-wise testing framework," 4th Annual IEEE Systems Conference, San Diego CA, 5-8 April 2010, pp 598-603.
[21] Carmelo Montanez-Rivera, D. R. Kuhn, M. Brady, R. M. Rivello, J. Reyes, and M. K. Powers (2012) "Evaluation of fault detection effectiveness for combinatorial and exhaustive selection of discretized test inputs," Software Quality Professional, American Society for Quality (to appear).
[22] Douglas C. Montgomery (2004) Design and Analysis of Experiments, 4-th edition, New York: Wiley.
[23] K. Nurmela (webpage) http://www.tcs.hut.fi/~kjnu/covarr.html
[24] George B. Sherwood (1994) "Effective testing of factor combinations," Proceedings of the 3rd International Conference on Software Testing, Analysis and Review, Jacksonville Florida, pp 151-166.
[25] George B. Sherwood (2011) "Getting the Most from Pairwise Testing, A Guide for Practicing Software Engineers," Colts Neck New Jersey: Testcover.com.
[26] Neil J. A. Sloan (1993) "Covering arrays and intersecting codes," Journal of Combinatorial Designs, 1, pp 51-63.
[27] Neil J. A. Sloan (webpage) http://www2.research.att.com/~njas/oadir/
[28] George W. Snedecor, and W. G. Cochran (1967) Statistical Methods, Iowa State University Press.
[29] Genichi Taguchi (1987) System of Experimental Design, Vol. 1 and Vol. 2, White Plains New York: UNIPUB, Kraus International (English translations of the 3rd edition of Jikken Keikakuho (Japanese) published in 1977 and 1978 by Maruzen).
[30] Keizo Tatsumi (1987) "Test-case design support system" Proceedings of the International Conference on Quality Control (ICQC 87), Tokyo, 20-23 October 1987, pp 615-620.
[31] Jose Torres-Jimenez (webpage) http://www.tamps.cinvestav.mx/~jtj/CA.php
[32] Wenhua Wang, S. Sampath, Yu Lei, and R. N. Kacker (2008) "An interaction-based test sequence generation approach for testing web applications," Proceedings of 11th IEEE International Conference on High Assurance Systems Engineering, Nanjing China, 3-5 December 2008, pp 209-218.
[33] Wenhua Wang, Yu Lei, S. Sampath, R. N. Kacker, D. R. Kuhn, and J. F. Lawrence (2009) "A combinatorial approach to building navigation graphs for dynamic web applications," Proceedings of 25th IEEE International Conference on Software Maintenance, Edmonton Canada, 20-26 September 2009, pp 211-220.
[34] Wenhua Wang, Yu Lei, D. Liu, D. Kung, C. Csallner, D. Zhang, R. N. Kacker and D. R. Kuhn (2011) "A combinatorial approach to detecting buffer overflow vulnerabilities," Proceedings of 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Hong Kong, 27-30 June 2011, pp 269-278.