Other Information of Interest

A catalog of orthogonal arrays can be found on N. J. A. Sloane's web site, A Library of Orthogonal Arrays. (opens a new window)

Orthogonal Array Testing Strategy (OATS) Technique.

Jeremy M. Harrell
Quality Assurance Manager
Seilevel, Inc.

Introduction

The Orthogonal Array Testing Strategy (OATS) is a systematic, statistical way of testing pair-wise interactions. It provides representative (uniformly distributed) coverage of of all variable pair combinations. This makes the technique particularly useful for integration testing of software components (especially in OO systems where multiple subclasses can be substituted as the server for a client). It is also quite useful for testing combinations of configurable options (such as a web page that lets the user choose the font style, background color, and page layout).

Background

Orthogonal arrays were originally discovered as a numerical curiosity by monks [Copeland2001]. The arrays went largely unnoticed, laying dormant in the aging notes of these monks, until the 1950s.

It was then that these "numerical curiosities" were picked up by the statistics community and put to use in statistical test design. Dr. Genichi Taguchi was one of the first proponents of orthogonal arrays in test design. His techniques, known as Taguchi Methods, have been a mainstay in experimental design in manufacturing fields for decades.

Orthogonal arrays are two dimensional arrays of numbers which possess the interesting quality that by choosing any two columns in the array you receive an even distribution of all the pair-wise combinations of values in the array.[1] Here is some terminology for working with orthogonal arrays followed by an example array in Figure 1 [Sloane2001]:

- Runs: the number of rows in the array. This directly translates to the number of test cases that will be generated by the OATS technique.
- Factors: the number of columns in an array. This directly translates to the maximum number of variables that can be handled by this array.
- Levels: the maximum number of values that can be taken on by any single factor. An orthogonal array will contain values from 0 to Levels-1.
- Strength: the number of columns it takes to see each of the $\text{Levels}^{\text{Stength}}$ possibilities equally often.
- Orthogonal arrays are most often named following the pattern $L_{\text{Runs}}(\text{Levels}^{\text{Factors}})$.

Figure 1: An $L_9(3^4)$ orthogonal array with 9 runs,
4 factors, 3 levels, and strength of 2.

	Factors			
Runs	0	0	0	0
	0	1	1	2
	0	2	2	1
	1	0	1	1
	1	1	2	0
	1	2	0	2
	2	0	2	2
	2	1	0	1
	2	2	1	0

Why use this technique?

Test case selection poses an interesting dilemma for the software professional. Almost everyone has heard that you can't test quality into a product, that testing can only show the existence of defects and never their absence, and that exhaustive testing quickly becomes impossible -- even in small systems. However, testing is necessary. Being intelligent about which test cases you choose can make all the difference between (a) endlessly executing tests that just aren't likely to find bugs and don't increase your confidence in the system and (b) executing a concise, well-defined set of tests that are likely to uncover most (not all) of the bugs and that give you a great deal more comfort in the quality of your software.

The basic fault model that lies beneath this technique is:

- Interactions and integrations are a major source of defects.
- Most of these defects are not a result of complex interactions such as "When the background is blue and the font is Arial and the layout has menus on the right and the images are large and it's a Thursday then the tables don't line up properly." Most of these defects arise from simple pair-wise interactions such as "When the font is Arial and the menus are on the right the tables don't line up properly."
- With so many possible combinations of components or settings, it is easy to miss one.
- Randomly selecting values to create all of the pair-wise combinations is bound to create inefficient test sets and test sets with random, senseless distribution of values.

OATS provides a means to select a test set that:

- Guarantees testing the pair-wise combinations of all the selected variables.
- Creates an efficient and concise test set with many fewer test cases than testing all combinations of all variables.
- Creates a test set that has an even distribution of all pair-wise combinations.
- Exercises some of the complex combinations of all the variables.[2]
- Is simpler to generate and less error prone than test sets created by hand.

As an example of the benefit of using the OATS technique over a test set that exhaustively tests every combination of all variables, consider a system that has four options, each of which can have three values. The exhaustive test set would require 81 test cases (3 x 3 x 3 x 3 or the Cartesian product of the options). The test set created by OATS (using the orthogonal array in Figure 1) has only nine test cases, yet tests all of the pair-wise combinations. The OATS test set is only 11% as large at the exhaustive set and will uncover most of the interaction bugs. It covers 100% (9 of 9) of the pair-wise combinations, 33% (9 of 27) of the three-way combinations, and 11% (9 of 81) of the four-way combinations. The test set could easily be augmented if there were particularly suspicious three- and four-way combinations that should be tested.

How to use this technique

The OATS technique is simple and straightforward. The steps are outlined below.

1. Decide how many independent variables will be tested for interaction. This will map to the Factors of the array.

2. Decide the maximum number of values that each independent variable will take on. This will map to

the Levels of the array.

3. Find a suitable orthogonal array with the smallest number of Runs.[3] A suitable array is one that has at least as many Factors as needed from Step 1 and has at least as many levels for each of those factors as decided in Step 2.

4. Map the Factors and values onto the array.

5. Choose values for any "left over" Levels.

6. Transcribe the Runs into test cases, adding any particularly suspicious combinations that aren't generated.

Examples

A Simple Example

Consider a web page with three distinct sections (Top, Middle, and Bottom) that can be individually shown or hidden by the user. You wish to test the interactions of the different sections. Following the instructions laid out previously, let's create a test set for the system.

1. There are three independent variables (the sections of the page).

2. Each variable can take on two values (hidden or visible).

3. An $L_4(2^3)$ orthogonal array will do the trick -- two levels for the values and three factors for the variables. Note that the number of runs is not necessary to pick an appropriate array.

4. Mapping the values onto the array would look like Figure 2 where Hidden=0 and Visible=1:

Figure 2

OA before mapping factors

	Factor 1	**Factor 2**	**Factor 3**
Run 1	0	0	0
Run 2	0	1	1
Run 3	1	0	1
Run 4	1	1	0

OA after mapping factors

	Top	**Middle**	**Bottom**
Test 1	Hidden	Hidden	Hidden
Test 2	Hidden	Visible	Visible
Test 3	Visible	Hidden	Visible
Test 4	Visible	Visible	Hidden

5. There are no "left over" Levels. In other words, there is a value mapped to every level in the array.

6. Taking the test case values from each run, you end up with four test cases. That is all that is needed to test all of the pair-wise interactions amongst the three variables. The test cases might transcribe to:

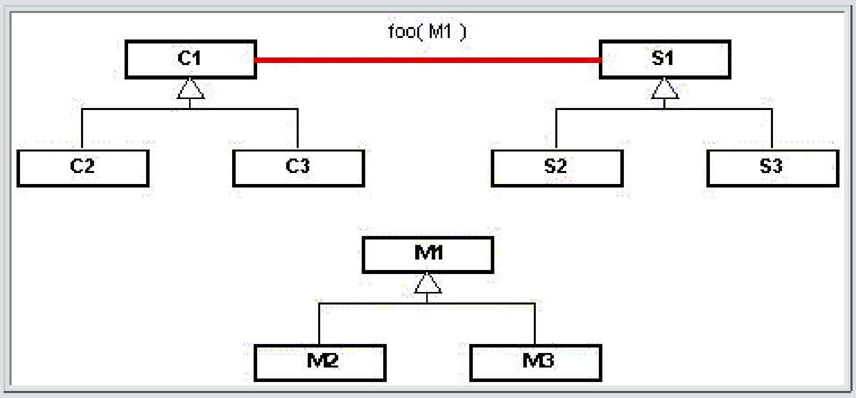
> A. Display the home page and hide all sections.
>
> B. Display the home page and show all but the Top section.
>
> C. Display the home page and show all but the Middle section.
>
> D. Display the home page and show all but the Bottom section.

Note that not all of the possible combinations are tested. It would take eight test cases to test all of the combinations. You can add test cases that are particularly suspicious, but don't show up in the pair-wise list. For example, you might want to add a test case to display all sections or only the middle section to the list above if you felt there was a strong possibility of error with those particular combinations.

An Example That Doesn't Fit the Array

Let's look at an example that doesn't fit perfectly into any available array. This example takes into account an object-oriented system that contains a client class (C1) with two subclasses (C2 and C3). These client classes interact with a server class hierarchy consisting of class S1 with subclasses S2 and S3. The server class contains a method foo() which takes an instance of class M1 as a parameter. M1 has two subclasses, M2 and M3. Figure 3 below depicts the classes involved.

Figure 3



To test all combinations of the classes involved, it would take 27 test cases (three clients that can each send three messages to three servers -- 3 x 3 x 3 = 27). That doesn't seem outrageous, but this assumes that the method foo() can be tested with only a single test case. In most circumstances, it will take many test cases to test a particular method. Also, this interaction is probably a very small portion of the entire system being tested. Using the OATS technique can significantly reduce the number of test cases.

1. There are three independent variables (the client, the server, and the message class).

2. Each variable can take on three values.

3. Ideally, we would us an array that contains three levels and three factors (an $L_?(3^3)$ OA). However, no such published array exists. Therefore, we need to look for the smallest array that will handle our problem. An $L_9(3^4)$ orthogonal will work. It has the three levels for the values and four factors is more than enough for the three variables.

4. Mapping the values onto the array would look like Figure 4 where:

A. For Client, C1=0; C2=1; C3=2.

B. For Server, S1=0; S2=1; S3=2.

C. For Message, M1=0; M2=1; M3=2.

Figure 4

OA before mapping factors

	Factor 1	Factor 2	Factor 3	Factor 4
Run 1	0	0	0	0
Run 2	0	1	1	2
Run 3	0	2	2	1
Run 4	1	0	1	1
Run 5	1	1	2	0
Run 6	1	2	0	2
Run 7	2	0	2	2
Run 8	2	1	0	1
Run 9	2	2	1	0

OA after mapping factors

	Client	Server	Message	
Test 1	C1	S1	M1	
Test 2	C1	S2	M2	
Test 3	C1	S3	M3	
Test 4	C2	S1	M2	
Test 5	C2	S2	M3	
Test 6	C2	S3	M1	
Test 7	C3	S1	M3	
Test 8	C3	S2	M1	
Test 9	C3	S3	M2	

5. There are no "left over" Levels. However, you'll notice that there was an extra Factor in the original array. This factor can simply be ignored; it does not change the properties of the test set generated from the array. You still get an even distribution of the pair-wise combinations.

6. Taking the test case values from each run, you end up with nine test cases. As mentioned before, these nine combinations might map to a larger set of test cases that must be executed against each of the nine combinations.

A Complex, Multi-Level Example

Here's a more complex example that introduces the concept of mixed-level orthogonal arrays. Let's say that we have a system with 5 independent variables (A, B, C, D, and E). Variables A and B each have two possible values (A_{1-2} and B_{1-2}). Variables C and D each have three possible values (C_{1-3} and D_{1-3}). Variable E has six possible values (E_{1-6}). To test all of the possible combinations, it would take a test set containing 216 test cases (2 x 2 x 3 x 3 x 6 = 216). This example shows how using the OATS technique can reduce the number of test cases to 18 in order to test all the pair-wise combinations.

1. There are five independent variables.

2. Two variables can take on two values. Two variables can take on three values. One variable can take on six values.

3. The easiest way to find a suitable OA is to go to your catalog of arrays and look for an array that has at least six levels (the maximum level for any of our variables) and at least five factors. The smallest array with a consistent number of levels you will find is probably the $L_{49}(7^8)$ OA. This array would generate a test set containing 49 tests. That's a lot better than 216, but it's still a lot of tests.

You may have noticed the phrase "consistent number of levels" in the previous paragraph. This is important because there happen to be a few orthogonal arrays that have a mixed number of levels. One such array is the $L_{18}(3^6 6^1)$ OA. The naming of this array means that there are 18 runs for 7 factors, 6 of which contain 3 levels and 1 of which contains 6 levels. Our problem happens to fit inside of this array, and the test set goes from 49 with the first array we identified down to 18. Now that's a lot better than 216 tests!

4. Mapping the values onto the array would look like Figure 5 where:

A. For A, A_1=0; A_2=1.

B. For B, B_1=0; B_2=1.

C. For C, C_1=0; C_2=1; C_3=2.

D. For D, D_1=0; D_2=1; D_3=2.

E. For E, E_1=0; E_2=1; E_3=2; E_4=3; E_5=4; E_6=5.

Figure 5

OA before mapping factors

	Factor 1	**Factor 2**	**Factor 3**	**Factor 4**	**Factor 5**	**Factor 6**	**Factor 7**
Run 1	0	0	0	0	0	0	0
Run 2	0	1	2	2	0	1	1
Run 3	0	2	1	2	1	0	2
Run 4	0	1	1	0	2	2	3
Run 5	0	2	0	1	2	1	4
Run 6	0	0	2	1	1	2	5
Run 7	1	1	1	1	1	1	0
Run 8	1	2	0	0	1	2	1
Run 9	1	0	2	0	2	1	2
Run 10	1	2	2	1	0	0	3
Run 11	1	0	1	2	0	2	4
Run 12	1	1	0	2	2	0	5
Run 13	2	2	2	2	2	2	0
Run 14	2	0	1	1	2	0	1
Run 15	2	1	0	1	0	2	2
Run 16	2	0	0	2	1	1	3
Run 17	2	1	2	0	1	0	4
Run 18	2	2	1	0	0	1	5

OA after mapping factors

	A	B	C	D			E
Test 1	A_1	B_1	C_1	D_1			E_1
Test 2	A_1	B_2	C_3	D_3			E_2
Test 3	A_1	2	C_2	D_3			E_3
Test 4	A_1	B_2	C_2	D_1			E_4
Test 5	A_1	2	C_1	D_2			E_5
Test 6	A_1	B_1	C_3	D_2			E_6
Test 7	A_2	B_2	C_2	D_2			E_1
Test 8	A_2	2	C_1	D_1			E_2
Test 9	A_2	B_1	C_3	D_1			E_3
Test 10	A_2	2	C_3	D_2			E_4
Test 11	A_2	B_1	C_2	D_3			E_5
Test 12	A_2	B_2	C_1	D_3			E_6
Test 13	2	2	C_3	D_3			E_1
Test 14	2	B_1	C_2	D_2			E_2
Test 15	2	B_2	C_1	D_2			E_3
Test 16	2	B_1	C_1	D_3			E_4
Test 17	2	B_2	C_3	D_1			E_5
Test 18	2	2	C_2	D_1			E_6

5. Like the previous example, this array has extra factors that are not needed. They can be safely ignored and are grayed out in Figure 5. This array has "left over" levels. Variables A and B both have three levels specified in the original array, but there are only two possible values for each variable. This has caused a level to be left over for variables A and B after mapping the factors. These left over values have been highlighted in yellow in Figure 5.

In order to have fully specified test cases for the runs that have left over levels, you must provide a value in the cell. The choice of the value is generally arbitrary, but it usually makes good sense to add as much variety to the test cases as possible in order to enlist chance on your side to help find errors [Marick1995]. A good way of doing this is to simply start at the top of a column and cycle through the possible values when filling in the left over levels. Figure 6 shows the table after filling in the remaining levels using the cycling technique mentioned.

Figure 6: OA after mapping left over levels

	A	**B**	**C**	**D**			**E**
Test 1	A_1	B_1	C_1	D_1			E_1
Test 2	A_1	B_2	C_3	D_3			E_2
Test 3	A_1	B_1	C_2	D_3			E_3
Test 4	A_1	B_2	C_2	D_1			E_4
Test 5	A_1	B_2	C_1	D_2			E_5
Test 6	A_1	B_1	C_3	D_2			E_6
Test 7	A_2	B_2	C_2	D_2			E_1
Test 8	A_2	B_1	C_1	D_1			E_2
Test 9	A_2	B_1	C_3	D_1			E_3
Test 10	A_2	B_2	C_3	D_2			E_4
Test 11	A_2	B_1	C_2	D_3			E_5
Test 12	A_2	B_2	C_1	D_3			E_6
Test 13	A_1	B_1	C_3	D_3			E_1
Test 14	A_2	B_1	C_2	D_2			E_2
Test 15	A_1	B_2	C_1	D_2			E_3
Test 16	A_2	B_1	C_1	D_3			E_4
Test 17	A_1	B_2	C_3	D_1			E_5
Test 18	A_2	B_2	C_2	D_1			E_6

6. As mentioned before, you get 18 test sets out of the 216 possible. These 18 test sets will test all of the pair-wise combinations of the independent variables. This demonstrates a significant savings in testing effort over the all combinations approach, and our fault model suggests that it will find the majority of the interaction defects.

Industry results

The author is unaware of any published industry or research results on the cost or effectiveness of using this technique for software testing. This highlights the need for such empirical research both for the OATS technique by itself and for the technique compared to other test selection techniques.

Endnotes

1. This paper is concerned only with orthogonal arrays of strength 2. Orthogonal arrays with different strengths would require a different number of columns to select the tests and would not necessarily be only concerned with pair-wise combinations.

2. The technique will not create test sets that exhaustively test all the variable combinations. It will generally not even cover a fourth of the possible combinations. The number of possible combination is the Cartesian product of the number of values for each factor. That number quickly becomes intractable for practical testing. However, the number of pair-wise combinations is generally much, much smaller.

3. Catalogs of orthogonal arrays can be found in the appendix of many advanced statistics books and books on Taguchi methods. Also, Neil Sloan's web site is an excellent source of orthogonal arrays. See the references and sidebar for the location of the web site.

References

[Copeland2001] Copeland, Lee. "Object-Oriented Testing." Software Quality Engineering. STAR East 2001. The Rosen Centre Hotel, Orlando, Florida. 14 May 2001.

[Marick1995] Marick, Brian. *The Craft of Software Testing: Subsystem Testing Including Object-Based and Object-Oriented Testing*. Upper Saddle River, NJ: PTR Prentice Hall, 1995.

[Sloane2001] Sloane, Neil J. A. *A Library of Orthogonal Arrays*. Information Sciences Research Center, AT&T Shannon Labs. 9 Aug. 2001 <http://www.research.att.com/~njas/oadir/>.