AN APPROACH FOR TEST CASES OPTIMIZATION IN REFERENCE TO HEALTH DATA USING SOFTWARE TECHNIQUE

Sunita Khurana, Shakti Banerjee*
School of Statistics, Devi Ahilya University, Indore, M.P., India

Abstract: In this paper we consider a software technique of orthogonal array testingapplying C programming approach. The main objective of a screening experiment is to identify significant factors of the studies. All the fields (Engineering, Medical, Agriculture etc.) require designs with minimum number of runs. Many designsuse the number of factors k equal to$n-1$, where n is the number of runs. But scientists can't even afford the number of runs required for these designs. For this purpose we try designs to be used with C language for efficient work on enterprise applications to complete any statistical experimental design with minimum time and cost. Orthogonal array testing strategygenerates reduced numbers of test cases automatically. It is very highly desirable in the context of industrial experimentation.

Keywords: **Orthogonal Array Approach, Pairwise testing strategy,Mixed orthogonal array approach, Software Testing, Test Case Generation, Test Case Reduction, OATS(Orthogonal array testing Strategy)**

1. INTRODUCTION

Pairwise testing strategy almostrequires fewer tests than orthogonal array based solutions.It is possible in some situations for them to have an equal number of tests. In software testing however, the added costs imposed by the extra tests are not worth. Software orthogonal array testing consists of test planning, test case design, test case execution, test result data collection and evaluation. It covers that appropriate actions is to be taken for improving the software solutions [8]. For any given software application we have huge number of test cases and we have to identify only those test cases that would lead us to expose maximum number of undetected errors. The benefit of OATS is lower execution time, increased productivity and higher code coverage [2].

The number of possible black-box test cases for any non-trivial software application is extremely large. The challenge in testing is to reduce the number of test cases to a subset that can be executed with available resources and can also exercise the software adequately so that majority of software defects are exposed. Black-box testing is one of the software techniques in designing experiments based on combinatorial covering designs [3].

These designs cover or execute combinations of input parameters in a systematic and effective way and are most applicable in testing data-driven systems where the manipulation of data inputs and the relationship between input parameters is the focus of testing experiments [5].

Definition

An Orthogonal Array of strength t with N rows, k columns $(k \geq t)$ based on s symbols is an Nx k array with entries $0,1,\ldots,s-1$, say, so that every N x t subarray contains each of the st possible t-tuples equally often as a row (say λ times) N must be a multiple of s^t , and $\lambda = N/s^t$ is the index of the array Notation: $OA(N; k; s; t)$ or sometimes $OA(N; s^k; t)$[2].

The following terminologies have been used in this approach:

Runs(N): 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(k): the number of columns in an array. This directly translates to the maximum number of variables that can be handled by this array.

Levels(s): 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 Levels Strength possibilities equally often.

Orthogonal arrays are most often named following the pattern L_{Runs} (Levels ^{Factors}).

Suppose a system which has 4 parameters and each of them has 3 values. To test all the possible combinations of these parameters (i.e. exhaustive testing) we will need a set of $3^4 = 81$ test cases. But instead of testing the system for each combination of parameters, we can use an orthogonal array to select only a subset of these combinations. Using orthogonal array testing, we can maximize the test coverage while minimizing the number of test cases to consider. We here assume that the pair that maximizes interaction between the parameters will have more defects and then the technique works and minimizes the test cases [2].

2. Orthogonal Array Testing Strategy (OATS)

We present the description of our proposed algorithm for 4 factors and nth rows in the following steps to construct the orthogonal array for testing a program with f factors, each factor having p levels (the different values of factors used in testing).

An OA is generally presented as a two-dimensional array, table, or matrix of N rows and k columns. Each entry in the array is one element of a set of s "symbols", often taken tobe $\{0, 1, 2 \ldots s - 1\}$or$\{1, 2, 3 \ldots s\}$. A standard notation often used to reference an OA of N rows, k columns, and s symbols, of strength $t$is$OA\ (N, k, s, t)$. OAs is used as a class of factorial designs. For this purpose, N is the number of experimental runs$k$ is the number of factors, s is the number of levels of each factor, and the array is the set of entries in a design matrix for an $s\ k$ experiment.

During orthogonal array testing, we observe the average change in the response when a factor is changed from one level to another level. Hence to achieve the entire test coverage, we should have $(f * p)$ number of test cases. We are using orthogonal array approach to reduce the number of test cases.

For construction of array first we construct "p - tuples" (e_1, e_2 ….e_p)as $e_1 = (0, 1, 2 \ldots p - 1)$, $e_2 = (1, 2 \ldots p)$, $e_i = (e_{i-1} + e_1)\ mod\ p$

2.1 Algorithm of Orthogonal Array Testing Strategy (AOATS)

AOATS can be applied to area in software testing as to:

•Derive test cases with multiple configurations for black-box testing of entire systems,

•Degree of interoperability testing in a large heterogeneous network i.e. select significant number of failures, caused by parameter interactions.

• Assume a suite of test cases for all possible configurations which cannot be afforded.

• Balance three or more input parameters.

•Calculate the minimal set of test configurations i.e. compute the minimal set of test parameter combinations.

Step I: //* Read inputs and identify whether P is prime or not. If not then generate next prime number. Also calculate the number of rows*//

```
begin {Main}
        Read the number of factors (F) and levels (P).
begin
for i:=1 to P do
{
if(P%i==0)
c++;
}
end
if(c==2)
        P=P;
else
        P=P-1;
        // calculate number of rows
        R=P*P;
```

Step II: //* Construct the Tuples e_1, e_2, ...,e_p*//

```
        begin
        for j:=0 to P do
        {
                e1[j]=j;
        }
        end;
        begin
for j:=1 to P do
 {
                e2[j]=j;
        }
        end;
        begin
for (j=0;j<P;j++)
        {
                e3[j]=e1[j]+e2[j+1];
                if (e3[j]>P)
                {
                        e3[j]=e3[j]-P;
                }
        else
        print(e3[j]);
        }
        end;
```

Step III: //* Fill the table *//
// First column of table
begin
for k:=1 to R do
{
if(k<=P)
print(1);
else
if(k<=P+P)
print(2);
else
print(3);
end;

// Fill reaming column for rows P
begin
for j:=2 to F do
{
if (k>P)
break;
print(k);
}
end;

// Fill the reaming table with the help of tuples generated in Step II
for i:=2 to P do
begin
{
if(k==l[i])
{
for j:=2 to F do
begin
if(i==2)
print(e_2[j-1]);
else
print(e_3[j-2]);
}
end;
}
end;
for i:=1 to P do
begin
{
if(k==m[i])
{
for(j=2;j<=F;j++)
begin
if(i==1)
{
if((e_2[j-1]+1)>P)
print((e_2[j-1]+1)-P);
else
print((e_2[j-1])+1);
}
else
{
if((e_3[j-2]+1)>P)
print((e_3[j-2]+1)-P);
else
print((e_3[j-2])+1);
}
}
if(k==q[i])

```
for j:=2 to F do
begin
if(i==1)
 {
if((e2[j-1]+2)>P)
print( (e2[j-1]+2)-P);
else
print( (e2[j-1])+2);
  }
else
 {
if((e3[j-2]+2)>P)
print( (e3[j-2]+2)-P);
else
print( (e3[j-2])+2);
  }
}
end;
}
end;
end; {Main}
```

Using above algorithm finally we get orthogonal array design which is shown in Table 2.1

Table 2.1

ROWS	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

here Table 2.1 gives a OA with 4 factors each at 3 levels in 9 runs.In the next step we check orthogonality of such design.

3. Concept of Orthogonality

Before thinking about the relationship between strength and resolution, it is helpful to briefly review two related senses in which the word "orthogonality" is used. In describing OAs we say that two columns of *s* symbols are orthogonal, if each of the *s*, 2 (ordered) pairs of symbols appears in an equal number of rows. We can extend this idea to pair of sets of columns. For example, in OAs of strength 4, every pair of columns is orthogonal to every other pair, in the sense that each of the *s*, 4 ordered pairs of symbols appears in an equal number of rows. The concept easily generalizes to any two sets of columns, and applies whether the two sets have columns in common or not.The orthogonal arrays have the following special properties that reduce the number of experiments to be conducted [2].

1.The vertical column under each independent variables of the above table has a special combination of level settings. All the level settings appear an equal number of times. For above L9 array under variable 4, level 1, level 2 and level 3 appears thrice. This is called the balancing property of orthogonal arrays.

2.All the level values of independent variables are used for conducting the experiments.

3.The sequence of level values for conducting the experiments shall not be changed. This means one can't conduct experiment 1 with variable 1, level 2 setup and experiment 4 with variable 1, level 1 setup. The reason for this is that the arrays of each factor columns are mutually orthogonal to any other column of level values. The inner product of vectors corresponding to weights is zero. If the above 3 levels are normalized between -1 and 1, then the weighing factors for level 1, level 2 , level 3 are -1 , 0 , 1 respectively. Hence the inner product of weighing factors of independent variable 1 and independent variable 3 would be

(-1 * -1+-1*0+-1*1)+(0*0+0*1+0*-1)+(1*0+1*1+1*-1)=0

3.1 Algorithm of Orthogonality

Step I: //* Read the input *//

```
begin {Main}
for i:= 1 to n do
begin
for j:= 1 to k do
input (read[i][j]);
```

```
end;
```

Step II: //* Find the inner product of one column with the remaining column *//

```
for i:=1 to n do
begin
      for j:=1 to k do
      begin
         {
if(j==1)
temp= p[i][j];
p1[i][j]= temp* p[i][j+1];
        }
end;
end;

for i:=1 to n do
begin
for j:=2 to k do
begin
          {
if(j==2)
temp= p[i][j];
p2[i][j]= temp* p[i][j+1];
           }
      end;
end;

for i:=1 to n do
begin
      for j:=3 to k do
begin
     {
if(j==3)
temp= p[i][j];
p3[i][j]= temp* p[i][j+1];
            }
end;
end;
```

Step III: //* Compute the sum of each inner product of each column*//

```
for j:=1 to k do
begin
  {
                T1=0;
                for i:=1 to n do
begin
   T1=T1+p1[i][j];
end;
   }
end;

for j:=2 to k do
begin
  {
 T2=0;
for i:=1 to n do
begin
    T2=T2+p2[i][j];
end;
   }
end;
```

```
for j:=3 to k do
begin
  {
T3=0;
for (i=1; i<=n; i++)
begin
   T3=T3+p3[i][j];
end;
  }
end;
end;{Main}
```

***4. Case Study*:** In the data below, cross-sectional study used in my paper was conducted on urban and rural diabetic patients. Demographic data and knowledge of participants was recorded regarding various aspects of diabetes type II. There is a need to enable the patients to understand the causes, risk factors, symptoms, signs, complications and various treatment modalities. Clinical sign and symptoms were observed like mouth ulcer, dry skin, dry throat, night blindness, memory loss, depression, hair fall etc. were observed during the treatment.

To investigate the relationships between participants, household, community level characteristics and diabetes awareness we used test design using Orthogonal Array which creates an efficient and concise test suite with fewer test cases without compromising test coverage. The experiments described in these examples are double-blind, which means that both the subjects and the experimenters do not know which treatment any subject has received.

In our case study using $L_9(3^4)$ array, we consider a system which has 4 parameters (Age, Type of exercise, Diet rating,weight observed)and each of them has 3 levels. First, parameter age is represented with level 1-(below 40), level 2-(40-55),level 3-(55-70) similarly other parameters represented in the table below. The permutationsof factor levels comprising a single treatment are so chosen that their responses are uncorrelated between treatments and therefore each treatment gives a unique piece of information.

The factors and various levels for each of the factors are listed below in Table 4.1:

Table 4.1: Factors and Levels listed for the Compatibility Testing:

Factors	Level 1	Level 2	Level 3
Age	Below40	40-55	55-70
Type of exercise	No exercise	Walk daily	Yoga
Diet rating	Poor	Good	Excellent
Weight observe	Decrease	Remain same	Increase

here the highest level =3, which is a prime number. Hence the OA with p=3 are constructed as follows in Table 4.2:

Table 4.2: Orthogonal Array constructed for the Compatibility Testing of treatments in patients.

Test Number	Age	Type of exercise	Diet rating	Weight observed
1	Below40	No exercise	Poor	Decrease
2	Below40	Walk daily	Good	Remain same
3	Below40	Yoga	Excellent	Increase
4	40-55	No exercise	Good	Increase
5	40-55	Walk daily	Excellent	Decrease
6	40-55	Yoga	Poor	Remain same
7	55-70	No exercise	Excellent	Remain same
8	55-70	Walk daily	Poor	Increase
9	55-70	Yoga	Good	Decrease

For 4 Factors, each with 3 levels, the total number of test cases = 3^4 = 81.With the Orthogonal Array Approach, we have been able to reduce it to 9 Test Cases. Thus we have been able to reduce the testing effort to 11.11% of the total effort. Hence, Effort Saved = 88.89%.

***5. Conclusion*:**In this paper, we consider a problem of generating small sets of test cases where the combinations that have to be covered by large input-output parameter relationships of a software system. That is, we only consider combinations of input parameters that affect an output parameter. To solve this problem, we implemented algorithm approach to construct an interaction testing system to generate small test cases. Particularly we are trying to apply an Algorithm of Orthogonal Array Testing Strategy (AOATS) in statistical way for testing pair-wise interactions. Algorithm of Orthogonal Array Testing Strategy (AOATS) provides a natural mechanism for testing systems to be deployed on a variety of arrays with software configurations which illustrates the results in the above case study that focuses on test reduction technique using AOATS approach. Case study shows the reduction of total test cases from 81 to 9 with the use of this algorithm 88.89% efforts have been saved.

References

[1] Banerji,S.: Orthogonal Array Approach for Test Case Optimization. International Journal of Advanced Research in Computer and Communication Engineering Vol. 1, Issue 9, ISSN 2319-5940 (2012)
[2] Banerjee,S., et.al.: Some contribution in construction of orthogonal array testing approach for optimizing test cases in diabetic people. Elixir statistics 101C 43830-43836, ISSN 2229 -712X (2016)
[3] Bach,J., Shroeder, P.: Pairwise testing - a best practice that isn't" In Proceedings of the 22nd Pacific Northwest Software Quality Conference, pages 180–196, (2004)
[4] Beizer,B.:Software Testing Techniques. Van Nostrand Reinhold, 2nd edition, (1990)
[5] Burr,K., Young, W.: Combinatorial test techniques: Table-based automation, test generation, and test coverage. Intl.Conference on Software Testing, Analysis, and Review (STAR), San Diego CA, (1998)
[6] Clarke, L. A.: A System to Generate Test Data and Symbolically Execute Programs. IEEE Transactions on Software Engineering, Vol. SE-2, No. 3, pp. 215-222(1976)
[7] DeCock, D.,Stufken, J.: On Finding Mixed Orthogonal Arrays of Strength 2 with Many 2-Level Factors. Statistics and Probability Letters, 50, pp.383–388 (2000)
[8] Korel, B.: Automated Software Test Data Generation. Conference on Software Engineering, Vol 10, No. 8, pp. 870-879 (1990)
[9] Kuhn, D.R., Reilly, M.J.: An Investigation of the Applicability of Design of Experiments to Software Testing. 27th NASA/IEEE Software Engineering Workshop, NASA Goddard Space Flight Center, (2002)
[10]Lekivetz, R.: A New Algorithm For Obtaining Mixed-Level Orthogonal And Nearly-Orthogonal Arrays. Master of Science in the Department of Statistics and Actuarial Science, Simon Fraser University (2006)
[11]Li,W. W., WuC. F. J.: Column wise-Pairwise Algorithms with Applications to the Construction of Supersaturated Designs. Technometrics, 39, pp.171–179 (1997)
[12]Morell,L. J.:A Theory of Error-Based Testing. Ph.D. thesis, University of Maryland, College Park MD, Technical Report TR-1395(1984)
[13]Salwan,R., Sehgal,R.:Test cases reduction technique considering the time and cost as evaluation standards. International Conference on Advanced Computing, Communication and Networks'11 , Amity University uttar Pradesh ,India.