Classic algorithms for Pairwise Testing

Andreas Rothmann

Hochschule Offenburg

andy.rothmann@gmx.de

November 2008

Abstract

This paper gives an overview on the most important classic algorithms for pairwise testing. All algorithms use combinatorial strategies to find a test set, which covers pairwise combinations of system parameters (for example system settings or inputs from the user). The idea of pairwise testing is already 20 years old but for the last five years its popularity has been rising extremely. The reason is that testers have to face more complex software projects with the same time target.

Keywords: Algorithm, Pairwise testing

1. Introduction

Pairwise testing (two-way testing) has its root in empiric results [1]. This shows that most failures are caused by one parameter (single-mode fault) or by the interaction of two parameters (double-mode fault). It is not necessary to test all possible values of different parameters in all combinations. It is sufficient to test each parameter in each of its states, tested in pair with every other parameter in each of its states. In certain circumstances an enormous decrease of test cases is possible. If there is a table with 75 fields and each field is able to have the values 0 and 1, there are 2^{75} possible test cases. Using pair wise testing reduces the number of test cases to 28. To clarify how pairwise testing is working, see the following example.

Precondition:

- *Parameter A has values* A_1, A_2 *and* A_3
- *Parameter B has values* B_1 ,B_2 *and* B_3
- *Parameter C has values* C_1 und C_2

If parameter A, B and C are tested in all possible combinations, there are 18 test sets.

Test case	A	B	C
1	A_1	B_1	C_1
2	A_2	B_1	C_1
3	A_3	B_1	C_1
4	A_1	B_2	C_1
5	A_2	B_2	C_1
6	A_3	B_2	C_1
7	A_1	B_3	C_1
8	A_2	B_3	C_1
9	A_3	B_3	C_1
10	A_1	B_1	C_2
11	A_2	B_1	C_2
12	A_3	B_1	C_2
13	A_1	B_2	C_2
14	A_2	B_2	C_2
15	A_3	B_2	C_2
16	A_1	B_3	C_2
17	A_2	B_3	C_2
18	A_3	B_3	C_2

Table 1: All possible combinations

By only taking single-mode faults and double-mode faults into consideration, 9 test cases are have to be executed.

Test case	A	B	C
1	A_1	B_1	C_1
5	A_2	B_2	C_1
6	A_3	B_2	C_1
8	A_2	B_3	C_1
9	A_3	B_3	C_1
11	A_2	B_1	C_2
12	A_3	B_1	C_2
13	A_1	B_2	C_2
16	A_1	B_3	C_2

Table 2: Reduced number of test cases

2. Classic algorithms

The algorithms (so-called combination strategies), which are used to minimize the number of test cases, are divided in three categories:

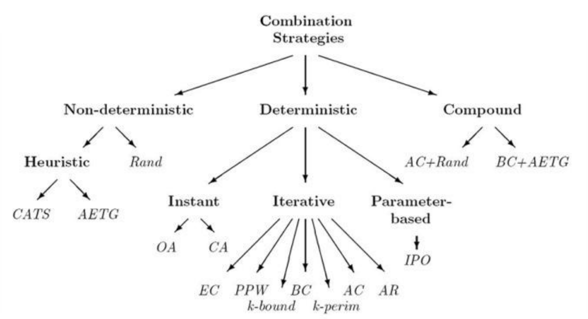

Figure 1: Overview[6]

Non-deterministic combination strategies contain a chance to extent. This chance causes different output (results) after the same input. The random algorithm displays the strongest kind of non-deterministic strategies. Another subcategory is the heuristic algorithm (for example implemented in AETG[1]).

Deterministic algorithm is the second subcategory of combination strategies. Here the same input always leads to the same output. Many algorithms, which belong to the determinic subcategory base on orthogonal arrays[1]. Further elements of deterministic combination strategies are the iterative and parameter-based strategies[2].

The third and last subcategory is called compound strategies. Here all algorithms are added, which have deterministic parts as well as non-deterministic parts.

3. Non-deterministic algorithms

3.1 Heuristic (t-wise[2])

Assume test cases $t_1 – t_{i-1}$ already selected.

Let Q be the set of all possible combinations not yet selected.
Let UC be a set of all pairs of values of any two parameters that are not yet covered by the test cases $t_1 – t_{i-1}$.

A)

- *Select t_i by finding the combination that covers the most pairs of UC. If more than one combination that covers the same amount of pairs, select the first one encountered.*
- *Remove the selected combination from Q.*
- *Remove the covered pairs from UC.*

B)

- *Repeat until UC is empty.*

Figure 2: Heuristic t-wise algorithm

Assume test cases $t_1 – t_{i-1}$ already selected.
Let UC be a set of all pairs of values of any two parameters that are not yet covered by the test cases $t_1 – t_{i-1}$.

A) Select candidates for t_i by
1. *Selecting the variable and the value included in the most pairs in UC.*
2. *Making a random order of the rest of the variables.*
3. *For each variable, in the sequence determined by step two, select the value included in most pairs of UC.*

B) Repeat steps 1-3 k times and let t_i be the test case that covers the most pairs in UC.

Repeat until UC is empty.

Figure 3: Heuristic t-wise algorithm [3]

3.2 Random

This subcategory of the non-deterministic strategies only reduces test cases by chance. So there is no pseudocode avaiable.

4. Deterministic algorithms

4.1 Orthogonal Arrays[1,5]

The use of orthogonal arrays is an established instrument in statistical experiments. It was

[1] AETG (Automated Efficient Test Generator) is a trademark of Telcordia Technologies.
[2] t is an arbitrary number.

developed by Taguchi Gen'ichi. The foundation of orthogonal arrays are n*n latin squares. The most important thing (and the main property of latin squares) is that an entry must not appear twice in a row or a column. By merging k-2 (k is the number of parameters) latin squares, an orthogonal array is created.

Notation:

O(c,k,n,t) is an orthogonal array, where

- c is the number of rows. Each row represents a test case.
- k represents the number of columns (number of parameters).
- each parameter has the maximum number of n values
- t is the strenght of the array. That means that in every submatrix c*t each n^t tuble appears just one time (in the case of pairwise testing t=2).

The clarify how to work with orthogonal arrays, assume three parameters (a,b,c) with three values each. The first parameter (a) takes the value of the row, the second takes the value of the column an the third takes the value, which is displayed in the matrix.

First build the matrix (as per rules above one 3*3 matrix is needed)

1	2	3
2	3	1
3	**1**	2

Figure 4: Orthogonal 3*3 array

For example the highlighted entry covers the test configuration (3,2,1). The number of test configurations matches with the number of entries in the matrix.

Test case	A(row)	B(column)	C(value)
1	1	1	1
2	1	2	2
3	1	3	3
4	2	1	2
5	2	2	3
6	2	3	1
7	3	1	3
8	3	2	1
9	3	3	2

Table 3: 3 parameters, 3 values each

If the number of parameters exceeds three, a combined matrix (C_{xy}), which bases on a set of x mutually Latin Squares, is needed. Suppose all possible combinations of four parameters have to be tested. So we need two (number of parameters -2) matrices. They are defined as matrix A and matrix B. The entries of the combined matrix C_{xy} = (A_{xy}, B_{xy}). Matrix A and matrix B are orthogonal if the ordered pairs (A_{xy},B_{xy}) are distinct for all x and y. Orthogonal arrays are balanced. That means that all values occur at least once and that all values occur the same number of times in a test set.

Matrix A

1	2	3
2	3	1
3	1	2

Matrix B

1	2	3
3	1	2
2	3	1

Matrix C

1,1	2,2	3,3
2,3	**3,1**	1,2
3,2	1,3	2,1

Figure 5: Matrix A, B and the combined matrix C

The configuration of the test cases is created as before. For example the configuration for the highlighted entry (3,1) is row 2, column 2 entry 3,1.

Test case	A (row)	B (column)	C (left value)	D (right value)
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

Table 4: 4 parameters, 3 values each

There are several restrictions for orthogonal arrays:

(1) All parameters have the same number of values.
(2) Values are dependent (some combination of values are not allowed).
(3) An adequate number of Latin Squares can be created.

Solutions:

(1) Select the parameter, which has the maximum number of values. Fill up all other parameters with “no care values” up to this number of values.
(2) Create hybrid pairs, which include allowed combination of values.
(3) Extend the squares size to a value, where enough Squares exist.

4.2 Covering array [4][5][6]

In contrast to orthogonal arrays, covering arrays are not balanced. Indeed all values appear at least once, but not the same number of times.
For notation see chapter 4.1. Covering arrays in comparison to orthogonal arrays can be constructed with k > n+1 parameters. This formula can be changed to $n_{max} = k – 1$. Now the first step is to build an orthogonal array (n^2, n+1, n).

To build covering arrays, the following denotations are needed.

- Basic array B(n^2-1, n+1,n, d) is equivalent to the orthogonal array with the first row removed. The columns are used d times consecutively.
- Reduced array R(n^2-n, n, n, d). is equivalent to the orthogonal array with n rows and the first column removed. The columns are used d times consecutively.
- An array I(c, d) with c rows and d columns. The array is filled with “1”.
- An array N(n^2 - n , n, d) with n^2 – n rows and n*d columns which of which consists of n * *d* sub arrays filled with values up to n.

Following tables refer to table 4.

Basic array, B(8,4,3,1):

1	2	2	2
1	3	3	3
2	1	2	3
2	2	3	1
2	3	1	2
3	1	3	2
3	2	1	3
3	3	2	1

Table 5: Basic array

Reduced array, R(6,3,3,1):

1	2	3
2	3	1
3	1	2
1	3	2
2	1	3
3	2	1

Table 6: Reduced array

Array I(9,1):

1
1
1
1
1
1
1
1
1

Table 7: Array I

Array N(6,3,1):

2
2
2
3
3
3

Table 8: Array N

Depending of the number of parameters and values, the different arrays are joined together.

4.3 Base Choice (BC)

Base choice considers that there are more and less important values. The most important values are selected to build to a default test configuration. In each test configuration, just the values, which are not in this default test configuration, change.

4.4 In-parameter-order(IPO)

For a system with n parameters (n>1), in-parameter-order creates a pairwise covering test set with the first two parameters. This test set will expand for every new parameter. This happens in two steps, the horizontal growth and the vertical growth. During horizontal growth a value of the new parameter is added to every existing test case (row). During vertical growth a new test case is added (column).

General IPO strategy [2]:
Assume that System S has parameters $p_1, p_2 \ldots p_n$, $n \geq 2$. Following strategy creates a test set T for S.

```
begin
    {for the first two parameters p1 and p2}
    T := {(v1,v2) | v1 and v2 are the values
    of p1 and p2 respectively}
    If n = 2 then stop;
    {for the remaining parameters}
    for parameter pi, i = 3,4,..,n do
      begin
          {horizontal growth}
          for each test (v1,v2,...,vi-1) in T do
             replace it with (v1,v2,...,vi-1,vi)
                where vi is a value of pi
          {vertical growth}
          while T does not cover all pairs between
           pi and each of p1,p2,...,pi-1 do
          add a new test for p1,p2,...,pi to T;
       end
end
```

Figure 6: General IPO strategy[2]

One advantage of in-parameter-order became clear:
If an established test set T is expanded to T' (by adding a new parameter or value), T can be reused. So less time is needed to create a new test set.

Following example will show how IPO works. Assume there are two parameters A, B with two values each. The complete test set T is {(A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2)}. By adding C, which has three possible values, the established test set must be expanded.

Horizontal growth:
The expanded test set T' contains the rows {(A_1, B_1, C_1), (A_1, B_2, C_1), (A_2, B_1, C_2), (A_2, B_2, C_2). It is obviously that there are not enough rows to satisfy pairwise coverage. Six pairs are uncovered {(A_1, C_2), (A_1, C_3), (A_2, C_1), (A_2, C_3), (B_1, C_3), (B_2, C_3)}.

Test case	A	B	C
1	A_1	B_1	C_1
2	A_1	B_2	C_1
3	A_2	B_1	C_2
4	A_2	B_2	C_2

Table 5: Horizontal growth

Vertical growth:
Focusing the uncovered pair the test case (A_1, §[3], C_2) must be added to cover the pair (A_1, C_2). {(A_1, C_3), (A_2, C_1), (A_2, C_3)} are handled with the test cases {(A_1, §, C_3), (A_2, §, C_1), (A_2, §, C_3)}.

Test case	A	B	C
1	A_1	B_1	C_1
2	A_1	B_2	C_1
3	A_2	B_1	C_2
4	A_2	B_2	C_2
5	A_1	§	C_2
6	A_1	§	C_3
7	A_2	§	C_1
8	A_2	§	C_3

Table 6: Vertical growth

To cover the two leftover pairs {(B1, C3), (B2, C3)}, don't care values can be easily replaced by B_1, respectively B_2. In the end, four new tests are generated for T'.

Test case	A	B	C
1	A_1	B_1	C_1
2	A_1	B_2	C_1
3	A_2	B_1	C_2
4	A_2	B_2	C_2
5	A_1	§	C_2
6	A_1	B_1	C_3
7	A_2	§	C_1
8	A_2	B_2	C_3

Table 7: T' test set

[3] § is a so called „don't care value". It will be replaced later by a sense full value.

4.5 All Combination (AC)

The all combination algorithm just builds all possible combinations of values of the input parameters. This results in a very large test set.

5. Compound strategies

5.1 All or Random

If the resulting test set size is less X (5000 is recommended for X by Kropp, Koopman, and Siewiorek [8]), then all combinations of parameter values are build. If the test set size will overstep X, than X test cases are randomly selected.

6. Comparison

In practice the most important topic is time. So testers watch out for an algorithm, which produces the minimal amount of test cases. Following table shows which number of test cases results from which algorithm. Depending on the algorithm, the all or random algorithm always has the maximum of X test cases.

Test strategy	Number of test cases (N parameter, V values)
Orthogonal Arrays	ca. V^2_{max}
Covering Arrays	ca. $N^2 + V_{max} \log^2 V_{max}$
Base Choice	$1+ \sum(V-1)$
IPO	ca. V^2_{max}
All Choice	V^n
All or random	max. X
Heuristic t-wise	ca. V^2_{max}

Table 8: Number of test cases

6. Conclusion

This paper gives an overview about the most important combination strategies and turn the attention to examples, which show how the single algorithm works. In chapter 6, the size of the test set from algorithm is showed respectively.

References

[1] R. Mandl. Orthogonal Latin Squares: An application of experiment design to compiler testing. Communications of the ACM, 28(10):1054-1058, October 1985.

[2] Y. Lei and K.C. Tai. In-parameter-order: A test generation strategy for pair-wise testing. In Proceedings of the third IEEE High Assurance Systems Engineering Symposium, pages 254-261. IEEE, November 1998.

[3] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton. The Combinatorial Design Approach to Automatic Test Generation. IEEE Software, pages 83-88, September 1996.

[4] A.W. Williams and R.L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. In Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE96), White Plains, New York, USA, Oct 30 - Nov 2, 1996, Nov 1996.

[5] A.W. Williams. Determination of test configurations for pair-wise interaction coverage. In Proceedings of the 13th International Conference on the Testing of Communicating Systems (TestCom 2000), Ottawa,Canada, August 2000, pages 59-74, August 2000.

[6] Karen Meagher. Covering Arrays on Graphs: Qualitative independence Graphs and extremal Set partition Theory. Chapter 2.

[7] Mats Grindal, Jeff Offutt, Sten F. Andler. Combination Testing Strategies: A Survey. GMU Technical Report ISE-TR-04-05, July 2004

[8] N.P. Kropp, P.J. Koopman, and D.P. Siewiorek. Automated robustness testing of off-the-shelf software components. In Proceedings of FTCS'98: Fault Tolerant Computing Symposium, June 23-25, 1998 in Munich, Germany, pages 230-239. IEEE, 1998.