Asian Journal of Information Technology 15 (23): 4800-4805, 2016
ISSN: 1682-3915

Test Case Generation for Pairwise+Testing

V. Chandra Prakash and Kadiyala Priyanka
Faculty of Computer Science and Engineering, KL University, Guntur Andhra Pradesh, India

Abstract: Software system faults can be completely detected only through exhaustive testing. But, it cannot be performed on many of the real-life systems because it may be too expensive as it may consume an enormous amount of time. The t-way combinatorial testing enables testing to be performed at low cost and less time. The value of t starts from 2 and takes 3, 4, 5 and so on. As the value of t increases, the size of the test suite increases and thus it takes more time for testing the system. In pairwise testing (t = 2), every pair of parameter values in the input domain is covered by at least one test case. It is highly effective in detecting up to 70% of faults triggered by a system. In 3-way testing, every triple of parameter values in input domain is covered by at least one test case. It can detect up to 90% of faults triggered by a system but the size of test suite is relatively larger. Therefore, pairwise testing is widely used in industry. When a test suite is generated for pairwise testing, it may contain a lot of gaps. The gaps have to be filled with some arbitrary values in order to proceed for testing. In this paper, we present an approach called pairwise+ testing in which the gaps in the test suite are filled in such a way that many of the triples that are useful for a 3-way testing are accommodated in the test suite. Thus the test suite generated for pairwise testing is enriched to cover a part of 3-way testing. Depending on the amount of coverage, the percentage of failures triggered by pairwise+ testing can be estimated to be anywhere within the range of 70 and 90%.

Key words: Combinatorial testing, pairwise testing, pairwise+ testing, test case generation, 3-way testing

INTRODUCTION

Innovations and new functionalities lead to tremendous growth in software applications over the past few years. Testing plays a vital role for these applications during software development process. By detecting faults in the system, it prevents disastrous consequences such as loss of data, fortunes and even lives. In order to detect faults, the functionality of the system is to be tested against the input domain. For this purpose, exhaustive testing is required. But, it is an expensive, time and resource consuming operation even if the system to be tested has fewer input parameters and values. Therefore, it is mandatory to reduce the size of a test suite in a systematic way. To address this issue, the focus is emphasized on sampling techniques that are based on interaction testing which is also called as t-way testing.

Combinatorial testing: Combinatorial testing is one of the software testing methods. It is also called as t-way testing. It detects the faults based on interaction between input parameters which in turn covers interaction between system components. It provides a systematic way to select combinations of system inputs for testing. It requires every combination of any t input parameter values to be covered by at least one test case. Given these combinations as input to the system, the expected output should be any one of the functional behaviors of the system. If the actual output is in contrast with it, the system contains a bug. Combinatorial testing is proved to be an effective testing technique to test the hardware or software which detects faults in the system. It has been used to test input domain, configurations, web forms, protocols, graphical user interfaces, software product lines, etc.

Combinatorial testing can be broadly applied at two levels (Kuhn *et al*., 2013). Level 1 is combinations of configuration values and level 2 is combinations of input values. They can be applied on both levels at the same time or separately. The number of possible combinations of values for a system with n parameters, each having d values is d^n. Kuhn and Kacker (2011) wrote a handbook on combinatorial testing which discusses the topic in detail.

T-way testing: T-way testing is one of the strategies in combinatorial testing (Lei *et al*., 2007). It requires that every combination of any t parameter values to be

Corresponding Author: V. Chandra Prakash, Faculty of Computer Science and Engineering, KL University, Guntur, Andhra Pradesh, India

```
Π = A set of triples = 3-way combinations of values involving parameters P1, P2, ...,and Pn
Algorithm Pairwise+(TestSuite ts) {
1.  Generate test suite ts for pairwise testing using any strategy;
2.  if ( the test suite ts doesn't contain a don't care )        exit;
3.  Generate Π containing a set of triples;
4.  Remove all the triples from Π that are already covered in ts;
5.  for each triple in Π
6.  {  for each test case in ts
7.     {       if ( the test case contains a don't care )
8.             {       if ( the triple can be accommodated in the test case )
9.                     {       Accommodate the triple in test case;
10.                            Remove the accommodated triple from Π;
11.                    } // end if at line 8.
12.            } // end if at line 7.
13.    } // end for at line 6.
14. } // end for at line 5.
15. return ts ;
}
```

Fig. 1: Algorithm pairwise+ testing

Table 1: A pairwise test suite for 4 2-valued parameters

Test case No.	A	B	C	D
1	A1	B1	C1	D1
2	A1	B2	C2	D2
3	A2	B1	C2	D2
4	A2	B2	C1	D1
5	-	-	C1	D2
6	-	-	C2	D1

covered by at least one test. T is the strength of coverage and usually takes values as 2-4 and so on. T-way testing reduces the size of the test suite compared to exhaustive testing. For example, a system with 20 parameters each having 10 values requires 1020 tests for exhaustive testing but only 180 tests for t-way testing (where t = 2 , i.e., pairwise testing) (Cohen *et al*., 2003).

Pairwise testing: In t-way testing, if the value of t is 2, it is called pairwise testing. It is also known as 2-way testing or all-pairs testing. It is widely used in industry. It requires that, every pair of parameter values in the input domain is covered by at least one test case. To illustrate pairwise testing, consider the following example system with 4 2-valued parameters i.e.:

- Parameter A has two values A1 and A2
- Parameter B has two values B1 and B2
- Parameter C has two values C1 and C2
- Parameter D has two values D1 and D2

Equation 1 shows all the 24 possible pairs for the given input parameters:

$$\begin{Bmatrix} (A1,B1)(A1,B2)(A2,B1)(A2,B2)(A1,C1)(A1,C2) \\ (A2,C1)(A2,C2)(A1,D1)(A1,D2)(A2,D1)(A2,D2) \\ (B1,C1)(B1,C2)(B2,C1)(B2,C2)(B1,D1)(B1,D2) \\ (B2,D1)(B2,D2)(C1,D1)(C1,D2)(C2,D1)(C2,D2) \end{Bmatrix}$$

Possible pairs for 4 2-valued parameters: To give each pair as input to the system is a difficult task when there are many parameters and values. Therefore, there are many approaches like AETG, IPO, ACTS, etc. to build an optimal test suite. The test suite for the above example using IPO strategy Lei and Tai (1998) is shown in Table 1. In the test suite, each row represents a test case and each column represents an input parameter. It can be verified that all the possible pairs given in Fig. 1 are accommodated (covered) shown in Table 1.

The 3-way testing: Faults can also be caused by interaction of more than two parameters (Kuhn and Reilly, 2002; Kuhn *et al*., 2004). In order to effectively detect those faults, it is necessary to use higher strength coverage testing. Higher strength coverage takes the value of t as 3, 4, 5 and so on. In t-way testing, if the value of t is 3, it is called 3-way testing. In this type of testing, every triple of parameter values in input domain is covered by at least one test case. 3-way testing can detect up to 90% of faults triggered by a system but the size of test suite is relatively larger than that of pairwise testing. To illustrate 3-way testing, let us consider the system mentioned in section 1.3 i.e., a system with 4 2-valued parameters. Similar to pairs in pairwise testing, we generate triples. The possible triples for the considered example are as shown in Eq. 2.

Table 2: The 3-way test suite for 4 2-valued parameters

Test case No.	A	B	C	D
1	A1	B1	C1	D1
2	A1	B1	C2	D2
3	A1	B2	C1	D2
4	A1	B2	C2	D1
5	A2	B1	C1	D2
6	A2	B1	C2	D1
7	A2	B2	C1	D1
8	A2	B2	C2	D2

$$\{(A1,B1,C1)(A1,B1,C2)(A1,B2,C1)$$
$$(A1,B2,C2)(A2,B1,C1)(A2,B1,C2)$$
$$(A2,B2,C1)(A2,B2,C2)(A1,B1,D1)$$
$$(A1,B1,D2)(A1,B2,D1)(A1,B2,D2)$$
$$(A2,B1,D1)(A2,B1,D2)(A2,B2,D1)$$
$$(A2,B2,D2)(B1,C1,D1)(B1,C1,D2)$$
$$(B1,C2,D1)(B1,C2,D2)(B2,C1,D1)$$
$$(B2,C1,D2)(B2,C2,D1)(B2,C2,D2)$$

Possible triples for 4 2-valued parameters: There are 24 possible triples. Table 2 shows a 3-way test suite in which all the triples are covered.

Don't cares: In Table 2, test case 5 contains don't care represented with '-' in column A (parameter A). It does not have any value. Similarly, we can find don't care in B column of test case 5 and A, B columns of test case 6. Without filling these don't cares, the test suite cannot be used for testing the System Under Test (SUT). Therefore, they have to be filled with any valid value of the corresponding parameter. One of the approaches used to fill a don't care is to fill it with a randomly selected value of corresponding column. For example, in test case 5 of Table 1, the first don't care occurs in parameter A. It can be filled with A2, a valid value randomly chosen from the set of values of parameter A.

Problem: In pair wise testing, when generating test cases we may find a lot of gaps (don't cares). They have to be filled by any random values to proceed for testing. To illustrate this, consider a system with 4 parameters A, B, C and D where

- A has two values A1 and A2
- B has four values B1, B2, B3 and B4
- C has one value C1
- D has four values D1, D2, D3 and D4

The test suite for such a system is as follows: The above test suite contains 25% of don't cares. The approach described to fill don't cares in section 1.5 is totally acceptable for pairwise testing. But, after filling the don't cares also, the test suite can detect only up to 70% of faults triggered by a system. Can we increase the percentage of fault detection of a test suite? If yes, does the size of the test suite increase? How to enrich the test suite?

Literature review: Combinatorial testing is one of the most interesting research topics. It is being enhanced from year to year over the past 20 year. Initiating from the proposal of different testing approaches (AETG, IPO, IPOG, IPO-s etc) and applying them to different applications (Traffic collision avoidance system, Personal identity verification etc.), many works have been studied. Few of them are mentioned in this study.

Mandl (1985) introduced Orthogonal Array Testing Strategy (OATS). It considers the system to be orthogonal when factors influencing the system are independent of each other. It uses "Orthogonal Latin Squares" to generate optimal test suite which covers all possible pairwise interactions.

Cohen *et al.* (1996, 1997) proposed an Automatic Efficient Test Generator (AETG) system. It uses a greedy algorithm to build the test suite which repetitively adds one test at a time until all the combinations of input domain are covered. They also explained that the number of test cases grow logarithmically as the number of parameters of a system increase.

Lei and Tai (1998) proposed an In-Parameter-Order (IPO) strategy which also uses a greedy algorithm to build a pairwise test suite. It starts from the combinations of first two input parameters and then covers the combinations of first three parameters and proceeds this way until combinations of all parameters are covered. This strategy achieves lower order complexity than AETG strategy.

Kuhn and Okum (2006) investigated on pseudo-exhaustive testing for software systems by considering the real-world examples Traffic Collision Avoidance System (TCAS) and Personal Identity Verification (PIV) smart card.

Lei *et al.* (2007, 2008) of National Institute of Standards and Technology (NIST, a US organization) proposed an approach called IPOG (G is short for Generalization). As IPO considers only pairwise testing, IPOG considers t-way testing where t is the strength of coverage. They also introduced a tool called fire eye a t-way testing tool.

Calvagna and Gargantini (2009) proposed IPO-s approach which is a parameter-based heuristic algorithm for the construction of pairwise covering test suites. The construction of test suite is based on symmetries of covering arrays.

Kuhn *et al.* (2009) have investigated on random vs combinatorial methods for discrete event simulation of a

grid computer network. In their study they explained that random generation of inputs in detecting deadlocks leads to exhaustive testing because deadlock may occur involving any number of parameter interactions and this is not known in advance. Therefore they proved that combinatorial methods provide better way for deadlock detection in different network configuration systems.

In 2010, Raghu N. Kacker investigated on advanced combinatorial test methods for system reliability. They calculated the number of failures detected for different systems like Medical devices, Browser, Server, NASA and Network Security when testing using t-way testing where t takes the values ranging from 2-6. The study revealed that on an average, 2-way detected 70% of faults triggered and 3-way detected 90% of faults triggered. The 4-way to 6-way detected higher values reaching up to 100% of faults triggered.

Kuhn and Kacker (2011) proposed combinatorial (t-way) methods for detecting complex faults in Regression testing. They introduced a new strategy which automatically generates tests suite covers complex combination of values or analyze existing test suites.

Kuhn and *et al.* (2011) applied combinatorial methods for event sequence testing. They say that the order of occurrence of n distinct events is also important. They tested t-events in every possible t-way order.

Yu *et al.* (2013) introduced the ACTS tool. ACTS is used to construct t-way combinatorial test suites where the value of t ranges from 1-6. It is a freely available tool which can be downloaded and installed on any system.

Duan *et al.* (2015) proposed an approach which improves the IPOG's vertical growth based on a graph coloring scheme. They considered the vertical growth problem as a "Minimum Vertex Coloring". By applying a greedy coloring algorithm for vertical growth they determined the order in which missing tuples are covered.

MATERIALS AND METHODS

The pairwise+ testing approach: Pair wise testing is more widely used than 3-way testing because the number of test cases for 3-way testing is relatively larger. For example, consider a system with 4 6-valued parameters. When we use ACTS tool, the number of test cases generated for pairwise testing is 49 where as the number of test cases generated for 3-way testing is 408. As 3-way testing needs many test cases, it is expensive and time consuming. Therefore, pairwise testing is mostly preferred. But, 3-way testing can detect up to 90% of faults triggered by a system where as pairwise testing can detect up to 70% of faults triggered by a system. In such a situation, to gain benefits of both pairwise and 3-way testing, we propose a new type of combinatorial testing called "Pairwise+ testing" which can detect 70-90% of faults triggered by the system without increasing the number of test cases generated for pairwise testing. In addition, the cost and time required for testing are same as pairwise testing. Our approach fills don't cares in an efficient way. It generates the test suite with minimal don't cares.

We consider the triples for the system with the same input domain that are used for pairwise testing. The test suite for pairwise testing naturally covers some of the triples. Our algorithm tries to accommodate as many uncovered triples as possible in the test suite. The don't cares are filled with values such that more number of uncovered triples are accommodated. Figure 1 is the algorithm for the pairwise+ testing.

RESULTS AND DISCUSSION

Consider the example system in section 1.6. In Table 3, test case 9 contains don't care in column A and column C i.e., (-, B1, -, D2). Don't care in columns A can be filled by the value A1 whereas don't care in column C can be filled by the C1. Therefore the test case (A1, B1, C1, D2) covers the triples (A1, B1, D2) and (B1, C1, D2). Repeat the same procedure for the test cases 10-16 and fill don't cares with valid values such that they contain uncovered triples. This decreases the number of don't cares in the test suite and sometimes the resulting test suite may not contain any don't cares at all. For our example, the number of triples that can be generated is 56. Table 4 is the test suite after accommodation of the triples.

As shown in Table 4, all 40 triples are accommodated in the test suite and there are no don't cares left to accommodate more triples. The percentage of triples accommodated is 71.5%. Therefore, 71.5% of 3-way testing is said to be covered in the test suite of pairwise testing. As investigated by Kuhn *et al.* (2012), on an average, the percentage of failures triggered by pairwise testing is 70% and 3-way testing is 90%. Since our approach covers 71.5% of 3-way testing in pairwise testing, the percentage of failures triggered by our test suite is estimated to be 85%. For some examples, the percentage of triples accommodated range between 70 and 90% which covers up to 3/4th of 3-way testing.

Table 3: Test suite with many don't cares

Test case No.	A	B	C	D
1	A1	B1	C1	D1
2	A1	B2	C1	D2
3	A1	B3	C1	D3
4	A1	B4	C1	D4
5	A2	B1	C1	D4
6	A2	B2	C1	D3
7	A2	B3	C1	D2
8	A2	B4	C1	D1
9	-	B1	-	D2
10	-	B1	-	D3
11	-	B2	-	D1
12	-	B2	-	D4
13	-	B3	-	D1
14	-	B3	-	D4
15	-	B4	-	D2
16	-	B4	-	D3

Table 4: Test suite after triple accommodation

Test case No.	A	B	C	D
1	A1	B1	C1	D1
2	A1	B2	C1	D2
3	A1	B3	C1	D3
4	A1	B4	C1	D4
5	A2	B1	C1	D4
6	A2	B2	C1	D3
7	A2	B3	C1	D2
8	A2	B4	C1	D1
9	A1	B1	C1	D2
10	A1	B1	C1	D3
11	A1	B2	C1	D1
12	A1	B2	C1	D4
13	A1	B3	C1	D1
14	A1	B3	C1	D4
15	A1	B4	C1	D2
16	A1	B4	C1	D3

Table 5: Comparative results for IPO and Pairwise+

Variable	S1	S2	S3	S4	S5
IPO (Don't cares)	14.00	36.00	18.00	75.00	168.0
Pairwise+ (Don't cares)	0.000	4.000	6.000	2.000	30.00
3-way testing covered (%)	70.58	76.66	79.31	87.56	90.12

Therefore, the percentage of fault detection of our test suite is estimated to be 85%. Note that the size of the test suite is same as pairwise testing. In this way, the test suite can be enriched. For analysis, consider the systems S1-S5. The exponential notation used to represent the size of input domain by Hartman and Raskin (2004) is used here, that is means a task with n parameters of range d (Table 5):

$$S1: 2^1,3^1,1^1,5^1;\ S2: 2^2,5^2$$

$$S3: 2^1,6^1,1^2,3^1;\ S4: 2^1,5^1,1^1,5^2$$

$$S5: 2^1,6^1,1^1,6^1$$

CONCLUSION

In this study, we introduced a new approach called "Pairwise+ testing". The algorithm is based on the innovative idea of accommodating triples in test suite of pairwise testing. In this approach, we have accommodated the maximum number of triples into the test suite.

IMPLEMENTATION

We have implemented this testing approach which is written in Java. As shown in Table 5, the results were most excellent by covering more than half percentage of 3-way testing in pairwise testing. Therefore, pairwise+ testing can replace pairwise testing wherever it is used. In future, there is a scope of enhancing the existing algorithm or propose a new algorithm which may cover more percentage of 3-way testing in pairwise testing.

REFERENCES

Calvagna, A. and A. Gargantini, 2009. IPO-s: Incremental generation of combinatorial interaction test data based on symmetries of covering arrays. Proceedings of the International Conference on Software Testing, Verification and Validation Workshops ICSTW'09, April 1-4, 2009, IEEE, Denver, Colorado, ISBN: 978-1-4244-4356-7, pp: 10-18.

Cohen, D.M., S.R. Dalal, J. Parelius and G.C. Patton, 1996. The combinatorial design approach to automatic test generation. IEEE. Software, 13: 83-88.

Cohen, D.M., S.R. Dalal, M.L. Fredman and G.C. Patton, 1997. The AETG system: An approach to testing based on combinatorial design. IEEE Trans. Software Eng., 23: 437-444.

Cohen, M.B., P.B. Gibbons, W.B. Mugridge and C.J. Colbourn, 2003. Constructing test suites for interaction testing. Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, IEEE, New Zealand, ISBN: 0-7695-1877-X, pp: 38-48.

Duan, F., Y. Lei, L. Yu, R.N. Kacker and D.R. Kuhn, 2015. Improving IPOGs vertical growth based on a graph coloring scheme. Proceedings of the 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), April 13-17, 2015, IEEE, Graz, Austria, pp: 1-8.

Hartman, A. and L. Raskin, 2004. Problems and algorithms for covering arrays. Discrete Math., 284: 149-156.

Kuhn, D.R. and M.J. Reilly, 2002. An investigation of the applicability of design of experiments to software testing. Proceedings of the 27th NASA/IEEE Software Engineering Workshop, Dec. 5-6, IEEE Computer Society, Washington DC., USA., pp: 69-80.

Kuhn, D.R. and V. Okum, 2006. Pseudo-exhaustive testing for software. Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop, April 24-28, IEEE Computer Society, Washington DC., USA., pp: 153-158.

Kuhn, D.R., D.R. Wallace and A.M. Gallo, 2004. Software fault interactions and implications for software testing. IEEE. Trans. Software Eng., 30: 418-421.

Kuhn, D.R., J.M. Higdon, J.F. Lawrence, R.N. Kacker and Y. Lei, 2012. Combinatorial methods for event sequence testing. Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, April 17-21, 2012, IEEE, Montreal, Quebec, Canada, ISBN: 978-1-4577-1906-6, pp: 601-609.

Kuhn, D.R., R. Kacker and Y. Lei, 2009. Random vs. combinatorial methods for discrete event simulation of a grid computer network. Proceedings of the Conference on Mod Sim World, October 14-17, 2009, National Aeronautics and Space Administration, Virginia Beach, Virginia, pp: 14-17.

Kuhn, D.R., R.N. Kacker and Y. Lei, 2013. Introduction to Combinatorial Testing. CRC Press, Boca Raton, Florida, Pages: 309.

Kuhn, R. and R. Kacker, 2011. Practical combinatorial (t-way) methods for detecting complex faults in regression testing. Proceedings of the 2011 27th IEEE International Conference on Software Maintenance (ICSM), September 25-30, 2011, IEEE, Williamsburg, Virginia, ISBN: 978-1-4577-0663-9, pp: 599-599.

Lei, Y. and K.C. Tai, 1998. In-parameter-order: A test generation strategy for pairwise testing. Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering, November 13-14, 1998, Washington DC. USA., pp: 254-261.

Lei, Y., R. Kacker, D.R. Kuhn, V. Okun and J. Lawrence, 2007. IPOG: A general strategy for t-way software testing. Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, March 26-29, 2007, Tucson, AZ., pp: 549-556.

Lei, Y., R. Kacker, D.R. Kuhn, V. Okun and J. Lawrence, 2008. IPOG/IPOG-D: Efficient test generation for multi-way combinatorial testing. Software Test. Verification Reliab., 18: 125-148.

Mandl, R., 1985. Orthogonal Latin squares: An application of experiment design to compiler testing. Commun. ACM., 28: 1054-1058.

Yu, L., Y. Lei, R.N. Kacker and D.R. Kuhn, 2013. Acts: A combinatorial test generation tool. Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, March 18-22, 2013, IEEE, Luxembourg, ISBN: 978-1-4673-5961-0, pp: 370-375.