A Parallel Tree Based Strategy for T-Way Combinatorial Interaction Testing

Mohammad F.J. Klaib[1], Sangeetha Muthuraman[2], and A. Noraziah[2]

[1] Department of Software Engineering, Faculty of Science and Information Technology, Jadara University, Irbid, Jordan
[2] Faculty of Computer Systems and Software Engineering, University Malaysia Pahang, Pahang, Malaysia
mohammadklaib@jadara.edu.jo, noraziah@ump.edu.my

Abstract. All software systems are built with basic components which interact with each other through predefined combination rules. As the number of components increases, the interactions between the components also increases exponentially which cause the combinatorial explosion problem. This mean complete (exhaustive) testing becomes unreasonable due to the huge number of possible combinations. Although 2-way interaction testing (i.e. pairwise testing) can relief and detect 50-97 percent of errors, empirical evidence has proved that 2-way interaction testing is a poor strategy for testing highly interactive systems and it has been showed that most of the errors are triggered by the interaction of 2-6 input parameters. In this paper we enhanced our previous strategy, "A Tree Based Strategy for Test Data Generation and Cost Calculation" by applying parallel algorithms to go beyond pairwise testing. The proposed strategy can support higher interaction testing. The designed algorithms are described in details with efficient empirical results.

Keywords: parallel algorithms, combinatorial interaction testing, software testing, T-way testing.

1 Introduction

Testing [1] is an activity that aims to evaluate the attributes or capabilities of software or hardware products, and determines if the products have met their requirements. Testing in general is a very important phase of the development cycle for both software and hardware products [2], [3]. Testing helps to reveal the hidden problems in the product, which otherwise goes unnoticed providing a false sense of well being. It is said to cover 40 to 50 percent of the development cost and resources [7]. Although important to quality and widely deployed by programmers and testers, testing still remains an art. A good set of test data is one that has a high chance of uncovering previously unknown errors at a faster pace. For a successful test run of a system, we need to construct a good set of test data covering all interactions among system components.

Failures of hardware and software systems are often caused due to unexpected interactions among system components [24]. The failure of any system may be catastrophic that we may lose very important data or fortunes or sometimes even lives. The main reason for failure is the lack of proper testing. A complete test requires testing all possible combinations of interactions, which can be exorbitant even for medium sized projects due to the huge number of combinations (Combinatorial explosion problem).

Testing all pairwise (2-way) interactions between input components helps to reveal the Combinatorial explosion problem and can ensure the detection of 50 – 97 percent of faults [10], [11], [12], [13], [14], [15], [16], [23]. Although using pairwise testing gives a good percentage of reduction in fault coverage, empirical studies show that pairwise testing is not sufficient enough for highly interactive systems [9], [17], [4] and constructing a minimum test set for combinatorial interaction is still a NP complete problem [14], [7] and there is no strategy can claim that it has the best generated test suite size for all cases and systems. Therefore, based on the above argument, this work comes to extend our previous strategy "A Tree Based Strategy for Test Data Generation and Cost Calculation" by applying parallel algorithms to go beyond pairwise testing (2-way interactions). The proposed strategy can support higher interaction testing.

The remainder of this paper is organized as follows. Section 2 presents the related work. In Section 3, the proposed tree generation and the iterative T-way cost calculation strategy is illustrated and its correctness has been proved with an example. Section 4 proves the performance of the proposed strategy with efficient empirical results. Section 5 provides the conclusion and lists the main advantages of the proposed strategy.

2 Related Work

Most of existing strategies support pairwise combinatorial interaction testing and a few have been extended to work for T-way testing.

AETG [10,][15] and its variant mAETG [21] employ the computational approach based on the criteria that every test case covers as many uncovered combinations as possible. The AETG uses a random search algorithm and hence the test cases are generated in a highly non-deterministic fashion [22].

In Genetic algorithm [14] an initial population of individuals (test cases) are created and then the fitness of the created individuals is calculated. This approach follows a non deterministic methodology similar to the Ant Colony Algorithm [14] in which each path from start to end point is associated with a candidate solution.

IPO [16] Strategy for pairwise testing starts constructing the test cases by considering the first two parameters, then uses a horizontal and vertical growth until all the pairs in the covering array are covered in a deterministic fashion. IPOG [9], [17] strategy extends IPO to support T-way interactions.

The IRPS Strategy [23] linked lists to search best test cases in a deterministic fashion. G2Way [8], [7] uses backtracking strategy to generate the test cases. TConfig

[18] uses recursive algorithm for T-way testing by applying the theory of orthogonal Latin squares. Jenny [19] first covers one way interaction, then pairs of features, then triples, and so forth up to the n-tuples requested by the user. WHITCH is IBM's Intelligent Test Case Handler [5], [6]. With the given coverage properties it uses combinatorial algorithms to construct test suites over large parameter spaces. TVG [20] combines both behaviour and data modelling techniques.

Although the importance work that have been done in the past by researchers, test suite generation for combinatorial interaction testing still remains a research area and NP complete problem that needs more exploration.

3 The Proposed Strategy

The proposed strategy constructs in parallel the testing tree based on the number of parameters and values. Number of base branches depends on the number of values of the first parameter. i.e. if the first parameter has 3 values then the tree also would have 3 base branches. Therefore every branch construction starts by getting one value of the first parameter i.e. branch T1 gets the first value, T2 gets the second value and so on. After the base branches are constructed one child thread is assigned to every branch and the further construction takes place in a parallel manner. Each of the branches considers all values of all the other parameters two, three…T where T is the total number of parameters. All the branches consider the values of the parameters in the same order. Suppose the following simple system with four parameters to illustrate the concept of the algorithm:

- Parameter A has two values A1 and A2.
- Parameter B has one value B1.
- Parameter C has three values C1, C2 and C3.
- Parameter D has two values D1 and D2.

Here the illustration will be for a 3-way combinatorial interactions testing. The algorithm starts constructing the test-tree by considering the first parameter. As the first parameter has two values the tree is said to have two base branches with the first branch using A1 and the second branch using A2. Then each of the branches is constructed in parallel by considering all the values of the second parameter, then the third and fourth and so on. When the branches are fully constructed the leaf nodes gives all the test cases that has to be considered for cost calculation. Since all of the branches are constructed in parallel there is a significant reduction in time. Figure 1 shows the test tree for the system above.

Figure 1 below shows how the test-tree would be constructed. The initial test cases are T1 (A1, B1, C1, D1), T2 (A1, B1, C1, D2), T3 (A1, B1, C2, D1), T4 (A1, B1, C2, D2), T5 (A1, B1, C3, D1), T6 (A1, B1, C3, D2), T7 (A2, B1, C1, D1), T8 (A2, B1, C1, D2), T9 (A2, B1, C2, D1), T10 (A2, B1, C2, D2), T11 (A2, B1, C3, D1) and T12 (A2, B1, C3, D2).

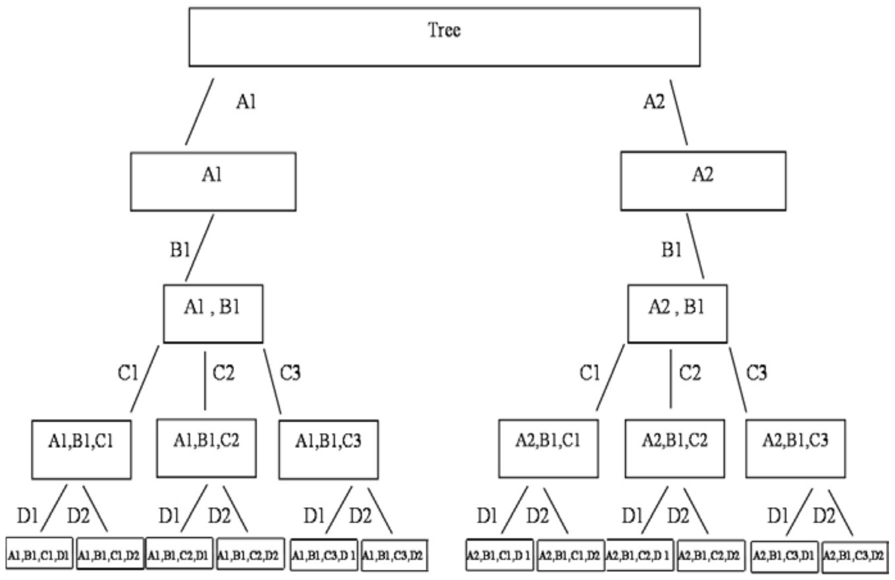

Fig. 1. Test-tree construction

Once the parallel tree construction is over we are ready with all the test cases to start the parallel iterative cost calculation. In this strategy the cost of the leaf nodes in each of the lists are calculated in parallel in order to reduce the execution time.

Table 1. 3-way interaction covering array

A, B, C	**A, B, D**	**A, C, D**	**B, C, D**
A1,B1, C1	*A1,B1, D1*	*A1, C1, D1*	*B1,C1, D1*
A1, B1,C2	*A1, B1,D2*	*A1, C1, D2*	*B1,C1, D2*
A1, B1,C3	*A2, B1,D1*	*A1, C2, D1*	*B1,C2, D1*
A2, B1,C1	*A2, B1,D2*	*A1, C2, D2*	*B1,C2, D2*
A2, B1,C2		*A1, C3, D1*	*B1,C3, D1*
A2, B1,C3		*A1, C3, D2*	*B1,C3, D2*
		A2, C1, D1	
		A2, C1, D2	
		A2, C2, D1	
		A2, C2, D2	
		A2, C3, D1	
		A2, C3, D2	

The cost of a particular test case is the maximum number of T-way combinations that it can cover from the covering array. Table 1 shows the covering array for 3-way combination i.e. [A, B, C], [A, B, D], [A, C, D] and [B, C, D], for the example in Figure 1. The covering array for the above example has 28 3-way interactions which have to be covered by any test suite generated to enable a complete 3-way interaction testing of the system. Table 2 shows how the cost calculation works iteratively to generate the test suite. Table 2 also shows the order in which the various test cases are actually included in the test suite.

Table 2. Generated test suite for 3-way combinatorial interaction

Test Case No.	Test Case	Iteration/ Child Thread No.	Max Weight	Covered pairs
T1	A1,B1,C1,D1	1/1	4	[A1,B1,C1][A1,B1,D1][A1,C1,D1][B1,C1,D1]
T4	A1,B1,C2,D2	1/1	4	[A1,B1,C2][A1,B1,D2][A1,C2,D2][B1,C2,D2]
T8	A2,B1,C1,D2	½	4	[A2,B1,C1][A2,B1,D2][A2,C1,D2][B1,C1,D2]
T9	A2,B1,C2,D1	½	4	[A2,B1,C2][A2,B1,D1][A2,C2,D1][B1,C2,D1]
T5	A1,B1,C3,D1	2/1	3	[A1,B1,C3][A1,C3,D1][B1,C3,D1]
T12	A2,B1,C3,D2	2/1	3	[A2,B1,C3][A2,C3,D2][B1,C3,D2]
T2	A1,B1,C1,D2	3/1	1	[A1,C1,D2]
T3	A1,B1,C2,D1	3/1	1	[A1,C2,D1]
T6	A1,B1,C3,D2	3/1	1	[A1,C3,D2]
T7	A2,B1,C1,D1	3/2	1	[A2,C1,D1]
T10	A2,B1,C2,D2	3/2	1	[A2,C2,D2]
T11	A2,B1,C3,D1	3/2	1	[A2,C3,D1]

The tree example shown in Fig. 2 explains how the test cases are constructed. In reality we may need only the leaf nodes and all the intermediate nodes are not used this increase the efficiency by minimising the number of nodes and giving importance only to the leaf nodes at every stage.

4 Empirical Results

With an example shown in Figure 1 at Section 4, the generated test suit (Table 2) has covered all the 3-way combinations (28) in Table 1, thus proving the correctness of the proposed strategy.

To evaluate the efficiency of the strategy for T -way test data generation, we consider six different configurations. The first three configurations have non-uniform parametric values. The other three configurations have a uniform number of values for all parameters. The six system configurations used are summarized as follows:

- S1: 3 parameters with 3, 2 and 3 values respectively.
- S2: 4 parameters with 2,1,2 and 1 values respectively
- S3: 5 parameters with 3, 2, 1, 2 and 2 values respectively.
- S4: 3 3-valued parameters
- S5: 4 3-valued parameters.
- S6: 5 2-valued parameters.

Table 3. 2-way results

System	Exhaustive number of test cases	2-way Test suite size	2-way Reduction %
S1	18	9	50%
S2	4	4	0%
S3	24	7	70.83%
S4	27	9	66.67%
S5	81	9	88.89%
S6	32	7	78.13%

Table 4. 3-way results

System	Exhaustive number of test cases	3-way Test suite size	3-way Reduction %
S2	4	4	0%
S3	24	16	33.33%
S5	81	31	61.73%
S6	32	12	62.5%

In Tables 3 and 4, column 2 shows the exhaustive number of test cases for each system. The last column shows the percentage of reduction achieved by using our strategy.

Results in Tables 3 and 4 demonstrate that our strategy is an efficient strategy in test size reduction. In Table 3 with pairwise test suite size reduction, in some cases a high reduction is achieved, as in systems S5 and S6 (more than 75%). In case of system S2, there is no reduction achieved because this is the minimum test suite size. In Table 4, which shows the 3-way test suite results there is reduction achieved in case of systems S3, S5 and S6, but in case of S2 no reduction is achieved as this is the minimum test suite size. The other systems such as S1 and S4 have 3 parameters only and therefore cannot be considered for 3-way test suite reductions. Thus, the tables 3 and 4 reveal that the proposed strategy works well for T-way test suite size reduction, for both parameters with uniform as well as non-uniform values.

5 Conclusion

In this paper a tree test generation strategy has been designed to support a parallel higher strength test interactions. The correctness of the proposed strategy has been proved in section 4 (Tables 2). Empirical results in Section 5 shows that our strategy is an efficient strategy in test size reduction and can generate highly reduced test suites. Our strategy includes only the minimum number of test cases which have covered the maximum number of T-way combinations into the generated test suite in each iteration, thus making it different from other strategies. Tables 3 and 4 reveal that the proposed strategy works well for different test strength (T) values, and can produce an efficient and reduced test suite size, for both uniform as well as non-uniform parametric values. Even though a good result in reduction achieved; there is a drawback in the first step of this strategy (i.e. generating the tree of test cases before reduction starts) which produce a huge number of test cases especially when the software under test has a large number of parameters. Improving will be in a future work.

References

1. Kaner, C.: Exploratory Testing. In: Proc. of the Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL (2006)
2. Bryce, R., Colbourn, C.J., Cohen, M.B.: A Framework of Greedy Methods for Constructing Interaction Tests. In: Proc. of the 27th International Conference on Software Engineering, pp. 146–155. St. Louis, MO, USA (2005)
3. Tsui, F.F., Karam, O.: Essentials of Software Engineering. Jones and Bartlett Publishers, Massachusetts, USA (2007)
4. Zamli, K.Z., Klaib, M.F.J., Younis, M.I., Isa, N.A.M., Abdullah, R.: Design and Implementation of a T-Way Test Data Generation Strategy with Automated Execution Tool Support. Information Sciences Journal (2011)
5. Hartman, A., Klinger, T., Raskin, L.: IBM Intelligent Test Configuration Handler. IBM Haifa and Watson Research Laboratories (2005b)
6. Hartman, A., Raskin, L.: Combinatorial Test Services (2004a), http://www.alphaworks.ibm.com/tech/cts (Accessed on August 2008)
7. Zamli, K.Z., Klaib, M.F.J., Younis, M.I.: G2Way: A Pairwise Test Data Generation Strategy with Automated Execution. Journal of Information and Communication Technology 9 (2010)
8. Klaib, M.F.J., Zamli, K.Z., Isa, N.A.M., Younis, M.I., Abdullah, R.: G2Way – A Backtracking Strategy for Pairwise Test Data Generation. In: Proc. of the 15th IEEE Asia-Pacific Software Engineering Conf, Beijing, China, pp. 463–470 (2008)

9. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG: A General Strategy for T-Way Software Testing. In: Proc. of the 14th Annual IEEE Intl. Conf. and Workshops on the Engineering of Computer-Based Systems, Tucson, AZ, U.S.A, pp. 549–556 (2007)
10. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG System: An Approach to Testing Based on Combinatorial Design. IEEE Transactions on Software Engineering 23, 437–444 (1997)
11. Cohen, M.B., Snyder, J., Rothermel, G.: Testing Across Configurations: Implications for Combinatorial Testing. In: Proc. of the 2nd Workshop on Advances in Model Based Software Testing, Raleigh, North Carolina, USA, pp. 1–9 (2006)
12. Colbourn, C.J., Cohen, M.B., Turban, R.C.: A Deterministic Density Algorithm for Pairwise Interaction Coverage. In: Proc. of the IASTED Intl. Conference on Software Engineering, Innsbruck, Austria, pp. 345–352 (2004)
13. Tai, K.C., Lei, Y.: A Test Generation Strategy for Pairwise Testing. IEEE Transactions on Software Engineering 28, 109–111 (2002)
14. Shiba, T., Tsuchiya, T., Kikuno, T.: Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. In: Proc. of the 28th Annual Intl. Computer Software and Applications Conf (COMPSAC 2004), Hong Kong, pp. 72–77 (2004)
15. Cohen, D.M., Dalal, S.R., Kajla, A., Patton, G.C.: The Automatic Efficient Test Generator (AETG) System. In: Proc. of the 5th International Symposium on Software Reliability Engineering, Monterey, CA, USA, pp. 303–309 (1994)
16. Lei, Y., Tai, K.C.: In-Parameter-Order: A Test Generation Strategy for Pairwise Testing. In: Proc. of the 3rd IEEE Intl. High-Assurance Systems Engineering Symp, Washington, DC, USA, pp. 254–261 (1998)
17. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG/IPOD: Efficient Test Generation for Multi-Way Software Testing. Journal of Software Testing, Verification, and Reliability 18, 125–148 (2009)
18. TConfig, `http://www.site.uottawa.ca/~awilliam/`
19. Jenny, `http://www.burtleburtle.net/bob/math/`
20. TVG, `http://sourceforge.net/projects/tvg`
21. Cohen, M.B.: Designing Test Suites for Software Interaction Testing. In: Computer Science, PhD University of Auckland New Zealand (2004)
22. Grindal, M., Offutt, J., Andler, S.F.: Combination Testing Strategies: a Survey. Software Testing Verification and Reliability 15, 167–200 (2005)
23. Younis, M.I., Zamli, K.Z., Mat Isa, N.A.: IRPS – An Efficient Test Data Generation Strategy for Pairwise Testing. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part I. LNCS (LNAI), vol. 5177, pp. 493–500. Springer, Heidelberg (2008)
24. Grindal, M.: Handling Combinatorial Explosion in Software Testing. Linkoping Studies in Science and Technology, Dissertation No. 1073, Sweden (2007