ICIT 2015 The 7th International Conference on Information Technology
doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

Anatomy of the Parallel Tree Based Strategy for High Strength Interaction Testing

Mohammad F. J. Klaib
Computer Science Department
College of Computer Sciences and Engineering
Taibah University
Madina, Kingdom of Saudi Arabia
Email: mklaib@taibahu.edu.sa
mom_klaib@yahoo.com

***Abstract*—Software and hardware testers concentrate on how to minimize the time involved in testing at the same time to ensure that the system is also tested well and made acceptable. This paper has enhanced and explained in details our previous strategy "A Tree Based Strategy for Test Data Generation and Cost Calculation for Pairwise Combinatorial Interaction Testing" to work effectively in parallel and to go beyond pairwise testing. The proposed strategy can now support a parallel 2-way and general multi-way combinatorial interaction test data generation based on two algorithms; a parallel tree generation algorithm which generates the test cases and a parallel T-way cost calculation algorithm which is used in constructing test suites with minimum number of test cases. Both strategies have been explained here in details.**

Keywords— Parallel algorithms, Software testing, Hardware testing, Multi-way testing

I. INTRODUCTION

A well-tested product or service is necessity to ensure customer's satisfaction. However, exhaustive testing is unaffordable due to combinatorial explosion problem. Combinatorial explosion in testing may occur for configurable systems. When systems under test have many configuration parameters, each with several possible values, testing each configuration is sometimes infeasible.

Combinatorial interaction testing has been one of the methods used to minimize the size and the time involved in testing [28-32], at the same time to ensure that the system is also tested well and made acceptable. The combinatorial interaction testing approach can reduce the number of test cases by systematically selecting a subset from an exhaustive testing combination based on the strength of parameter interaction [9-14, 23,27]. Basic combinatorial interaction testing which is called pairwise or 2-way testing [4-8] provides a systematic approach to identify and isolate faults since many faults are caused by unexpected 2-way interactions among system factors. Empirical investigations have concluded that from 50 to 97 percent of software faults [1, 6, 9, 15, 24- 26] could be identified by pairwise combinatorial interaction testing. However, what about the remaining faults? Especially, in case of highly interactive systems which have a number of interactions with higher strength. How many failures could be triggered only by an unusual interaction involving more than two parameters? Investigations have found that many faults were caused by a single parameter, a smaller proportion resulted from an interaction between two parameter values, and progressively fewer were triggered by 3-6 way interactions [3,18- 22].

Therefore, to ensure a high quality testing of complex applications, it is necessary to generate test suites for higher degree T-way interactions. T-way testing [3, 18, 19, 20, 21, 22] requires every combination of any T parameter values to be covered by at least one test, where T is referred to as the strength of coverage. If all faults in a system can be triggered by a combination of T or fewer parameters, then testing all T-way combinations of parameters can provide high confidence that nearly all faults have been discovered. A number of studies have shown combinatorial methods to be highly effective for software and hardware testing.

Large and/or computationally expensive optimization problems sometimes require parallel or high-performance computing systems. Parallel algorithms have been applied to problems such as weather and climate modelling, bioinformatics analysis, logistics and transportation, and engineering design. Furthermore, commercial applications are driving development of effective parallel software [16, 17, 22, 26] for large-scale applications such as data mining and computational medicine. In the simplest sense, parallel computing involves the simultaneous use of multiple computer resources to solve a computational problem.

In this paper we have enhanced our previous strategy "A Tree Based Strategy for Test Data Generation and Cost Calculation" [24, 25, 26] to work in parallel and to go beyond pairwise (2-way) testing. The proposed strategy can now support a parallel and general T-way combinatorial test data generation involving uniform and non uniform parametric values. The proposed strategy is based on two algorithms; a parallel tree based test data generation algorithm which generates all the test cases, and a parallel T-way cost calculation algorithm which is applied to construct T-way test suites with minimum number of test cases.

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

The remainder of this paper is organized as follows. Section 2 explains the parallel tree generation and the proposed iterative T-way cost calculation strategy with an example. Section 3 gives parallel tree generation algorithms for test case generation and explains its advantages. Section 4 presents the parallel, iterative, T-way cost calculation algorithm for T-way test suites generation, with its working explained. Finally, Section 5 gives the conclusion.

II. THE PROPOSED STRATEGY

The proposed strategy constructs the tree based on the parameters and values given. It constructs every branch of the tree in parallel. The number of branches the tree has 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 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,…..N where N is the total number of parameters. All the branches consider the values of the parameters in the same order. The following simple system with parameters and values, illustrates the concept as shown below:

- 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

We have given the illustration for minimum test suite construction of 2-way and 3-way combinatorial interactions testing using our algorithm, for the system mentioned. 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 main 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. Fig. 1 shows the test tree for the system below.

Fig. 1 above shows how the test-tree would be constructed. The test cases generated by the first branch are stored in the lists T_1 and the test cases generated by the second branch are stored in T_2 respectively. i.e. (A1,B1,C1,D1), (A1,B1,C1,D2), (A1,B1,C2,D1), (A1,B1,C2,D2), (A1,B1,C3,D1), (A1,B1,C3,D2) are stored in T_1, and (A2,B1,C1,D1), (A2,B1,C1,D2), (A2,B1,C2,D1), (A2,B1,C2,D2), (A2,B1,C3,D1) and (A2,B1,C3,D2) are stored in T_2.

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. The cost of a particular test case is the maximum number of T-way combinations that it can cover from the covering array. At First, the algorithm starts by constructing the covering array, for all possible T-way combinations of input variables, if T equals 2 i.e. [A & B], [A & C], [A & D], [B & C], [B & D] and [C & D]. The covering array for the above example has 23 pairwise interactions as shown in Table 1, which has to be covered by any test suite generated, to enable a complete pairwise interaction testing of the system.

Once the covering array is generated the algorithm starts to include all tree branches. which might definitely give the maximum Wmax cost into the test suite. Then these test cases are deleted from the tree branches lists T_1 and T_2, and the corresponding pairs covered by it in the covering array are also deleted. In the third step, the main thread in the algorithm invokes a number of child threads equal to the number of values of the first parameter and calculates the cost of all the test cases

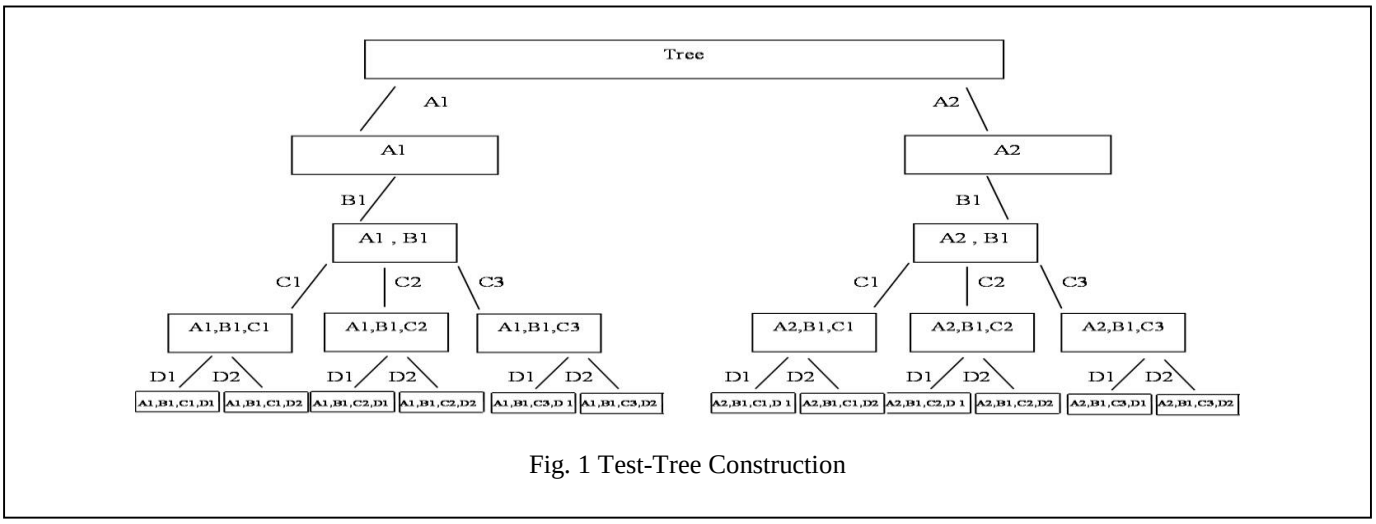

Fig. 1 Test-Tree Construction

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

in each of the branches in a parallel fashion. Each child thread stores all the test cases with the Wmax value from its corresponding branch into a separate sub-list. The child thread that finishes calculating the cost of all the test cases in its branch first locks the covering array. This thread then looks into its sub-list and includes the test cases stored in it into the test suite only after confirming that the test case definitely has the maximum cost or Wmax value. Then the test cases included in the test suite are deleted from the tree branches list and sub-list, and the corresponding pairs that these cover are deleted from the covering array.

TABLE 1. PAIRWISE COVERING ARRAY.

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

The other threads wait in a queue until the execution of the first thread is over, after which these threads resume their execution in the order in which they are queued. These threads on resumption re-evaluate the test cases in their sub-list to confirm that these test cases have the Wmax value before including these into the test suite. Thus in the first iteration all the test cases with the maximum Wmax value from all the branches are included in the test suite. Now the Wmax value is decremented by one and the same parallel execution of all the threads continue until all the pairs in the covering array are covered. For the above example all the test cases which are included in the test suite are identified in four iterations and there are six such test cases. Table 2 shows how the cost calculation works iteratively to generate the test suite. The same test suite gets generated if a sequential execution of the above algorithm takes place.

As the pairwise test suite is generated, we can generate the test suite for 3-way combinatorial interactions and so on the forth until (n-1) way combinatorial interaction test suites are generated. To illustrate the 3-way test suite generation, again the whole process starts by constructing the 3-way covering array and the iterative, parallel cost calculation of the test cases in the various branches as explained before. Table 3 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 Fig. 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 4 shows how the cost calculation works iteratively to generate the test suite. Table 4 also shows the order in which the various test cases are actually included in the test suite.

TABLE 3 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	

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

III. PARALLEL TREE GENERATION ALGORITHMS FOR TEST CASE GENERATION

A. Tree Generation Algorithm for Main Thread

Input: A set of parameters and the values of the corresponding parameters

Output: Lists of test cases. Each list holds the Fig. 4 Cost Calculation Algorithm

Test cases generated by the tree in one particular branch of that tree.

Begin

X = number of values of first parameter p1

{For the first parameter p1}
$T_i=V_i$, where i=1,2,3,……..,X/ parameter p1 has X values

If N=1 then stop and exit;

Create X threads with unique thread ids. Assign each T_i to a separate child thread and execute all the child threads in parallel

Wait for the termination of all the threads to get the results from all the branches.

End

TABLE 2. GENERATED TEST SUITE FOR PAIRWISE COMBINATORIAL INTERACTION.

Test Case No.	Test Case	Iteration/Child Thread No.	Max Weight	Covered pairs
T1	A1,B1,C1,D1	1/1	6	[A1,B1][A1,C1][A1,D1] [B1,C1][B1,D1][C1,D1]
T10	A2,B1,C2,D2	½	6	[A2,B1][A2,C2][A2,D2] [B1,C2][B1,D2][C2,D2]
T6	A1,B1,C3,D2	2/1	4	[A1,C3][A1,D2] [B1,C3][C3,D2]
T11	A2,B1,C3,D1	3/2	3	[A2,C3] [A2,D1] [C3,D1]
T3	A1,B1,C2,D1	4//1	2	[A1,C2] [C2,D1]
T8	A2,B1,C1,D2	4/2	2	[A2,C1] [C1,D2]

TABLE 4 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]

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

```
Begin

  {For the remaining parameters the execution takes place in parallel}

    For parameters Pj, j=2,3,……N do
       Where N is the total number of parameters

    Begin
     For each Test (Vi1, Vi2,…….Vim) in Ti do
        Where i = 1,2,…..X, X is the number of values of parameter p1and m is the maximum number of test cases in list Ti
        at that Time

        Begin
        Replicate the Test as many times as (the number of values of Pj – 1)
        Add all the replicated nodes sequentially after the current original test node and before the other test nodes in Ti
        For each value in Pj do
                Begin

                Append the original node with V1 and all the replicated tests with (V2, V3,……..Vy-1, Vy) where Vy is a
                value of Pj and each of which is considered in order.

                End
        End
      End
End
```

B. Tree generation Algorithm for Child Thread

The tree generation algorithm thus provides the following advantages:

1. A systematic method whereby all possible test cases are generated in order.
2. The above procedure works fine with the parameters having any number of values. Therefore all parameters can have different or same values as any real time system to be tested would have.
3. The procedure appears to generate the full tree by using all the values of the parameters but at every iteration only a set of leaf nodes are left thus having a list of leaf nodes (or test cases) when the procedure ends.
4. Since the test cases in every branch are generated in parallel by the child threads there is significant reduction in time.

The example tree shown in Fig. 1 explains how the test cases are constructed manually. In reality we may need only the leaf nodes and all the intermediate nodes are not used. Therefore in order to increase the efficiency of the implementation we have constructed the same tree as in Fig. 1 using the proposed parallel tree generation algorithm. This proposed algorithm constructs the tree by minimising the number of nodes. Minimisation of the number of nodes is achieved by giving importance only to the leaf nodes at every stage. The main thread just constructs the base branches of the tree each of which consists of one value of the first parameter in an order in which the input was made. Therefore, in the example above there are only two base branches and the value A1 is assigned to branch T_1 and A2 to T_2. Then the main algorithm invokes a number of unique child threads to handle each of the branches separately. At each stage or iteration each of the child threads look at the leaf nodes of their corresponding branches and generate the next level nodes by considering all the values of the current parameter, to generate the new set of nodes. The new set of leaf nodes from an already existing set is calculated using a replication strategy. The existing set of leaf nodes be Esoln, new set of leaf nodes be Nsoln and the number of values of the parameter under consideration be n. Then,

$$Nsoln = Esoln * n \qquad (1)$$

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

Let there be 4 leaf nodes in a branch and the next parameter to be considered has 2 values. Then the new list of nodes for that branch will have 8 new leaf nodes as a result. The algorithm considers every leaf node separately and calculates the number of times this particular node needs to be replicated with the formulae given below:

$$\text{The number of values of } p_j - 1 \qquad (2) \qquad (2)$$

Where p_j – is the j^{th} parameter under consideration for constructing the new set of leaf nodes and j=1, 2,….N – the number of parameters. In the Fig. 1 that is shown above consider the leaf nodes (A1, B1) of list or branch T1 and (A2, B1) of branch T2. To construct the next level of leaf nodes the parameter under consideration is C, which has values C1, C2 and C3. Therefore, the node (A1, B1) needs to be replicated twice. Now we will have three (A1, B1) nodes to which C1 is added to the first, C2 is added to the second and C3 is added to the third and then the replicated nodes are included in the list of leaf nodes after the original node. The same is done to (A2, B1). It is replicated twice and hence we have three of it (one original and two replicated nodes). Now C1 is added to the first (original node), C2 is added to the second (replicated node) and C3 is added to the third (replicated node). Thus we have (A2, B1, C1), (A2, B1, C2) and (A2, B1, C3). If there are more parameters the same is continued until all the parameters are considered. Thus, once the lists of leaf nodes are generated we go to the next strategy of iterative and parallel cost calculation to construct the test suite.

IV. Test Suite Generation by Iterative and Parallel Cost Calculation Strategy

The main thread includes the base test cases which would definitely have a maximum cost value and then invokes a number of unique child threads which operate in parallel on each of the branches lists. The main thread iterates N-2 times thus generating N-2 test suites. In the first iteration, i=2, the child threads iterate through the lists of test cases until all the pairs of the 2-way covering array are covered. Then the minimum 2-way test suite generated is stored and the next iteration begins. Now, i=3 and the child threads iterates again through the lists of test cases until all the 3-way combinations of the 3-way covering array are covered and then the 3-way test suite generated is stored. Thus this is continued until i= N-1. At each iteration, all the test cases with the maximum cost (Wmax) for that particular iteration are included in the test suite. Thus the algorithm guarantees identifying minimum test suites for parameters with same as well as different number of values.

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

A. Strategy T-way Test Suite Generation by Iterative and Parallel Cost Calculation (Main Thread)

Input: Lists of test cases. Each list holds the test cases generated by the tree in one particular branch of that tree.

Output: T-way test suites with minimum number of test cases

Begin
$Temp_b = T_b$ (where b is the number of lists of test cases)

X = number of values of parameter p1

B=min (Value(p1), Value(p2), …….Value(pn)

For i = 2 to N-1 do

Begin
Generate the i-way covering array for the given parameters.

`Wmax = N!/((i!)*((N-i)!))` // N – is the number of parameters

Let T' be an empty set where i-way test suites are stored.

For a = 1 to B do

Begin
$Test_a$ = concatenate the ath values of all the parameters to form a test case.
End

For each $Test_a$ do
Begin
Delete all the T-way combinations that $Test_a$ covers in the covering array
Delete $Test_a$ from the T_i Lists
T' = $Test_a$
End

Creates a set of temporary lists Y_i corresponding to the T_i lists, where i= 1,2,…..X, X is the number of values of parameter p1 or the number of lists.

Create X threads with unique thread ids. Assign every child thread Th_i with one T_i lists, the corresponding Y_i lists, i value and Wmax value, and execute all the child threads in parallel.

Wait for the termination of all the child threads.

Store the i-way test suite generated in the list T'

$T_b = Temp_b$

End
End

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

B. Strategy T-way Test Suite Generation by Iterative and

```
Begin

While (covering array is not empty) do

Begin

   For each Test Tij in Ti do
      Where i=1,2,…….X, X – is the number    of lists and j=1,2,….n where there are n test cases in Ti at that time

     Begin

       Cost[Tij]= The number of T-way combinations covered by it in the covering array

       If (Cost[Tij]==Wmax)
        Begin

          Yi = Tij

        End

     End

     {Whichever thread completes its execution first locks the covering array and updates all its test cases with Wmax values from Yi to the
       Test suite T' and deletes all the corresponding T-way combinations of those test cases included in T' from the covering array. The
       other threads on completing execution enters a queue and does its updation in that queued order by locking and unlocking the covering
       array after the first thread releases its lock on the covering array }

     For each Yi do (lock the covering array and make updation)
     Begin

      If (Yi != empty)
        Begin

          For each Test Tij in Yi do
          Begin

          Count= The number of T-way combinations covered by it in the covering array

          If (Count ==Wmax)
          Begin

            T' = T' U Tij

            Delete all the T-way combinations that Tij covers   in the covering array

           Delete Tij from the lists Ti

         End

          Delete Tij from the lists Yi

         End

        End
    (unlock the covering array)
       End
       Wait until all child threads finishes updating
  Wmax=Wmax-1
End
End
```

doi:10.15849/icit.2015.0112 © ICIT 2015 (http://icit.zuj.edu.jo/ICIT15)

Parallel Cost Calculation (Child Thread)

V. CONCLUSION

In this paper we have explained in details the parallel tree based test data generation and parallel iterative cost calculation strategy for multi-way combinatorial interaction testing and the correctness of the proposed strategy has been proved in section 3 (Tables 1, 2, 3 and 4).

REFERENCES

[1] M. F. J. Klaib, K. Z. Zamli, N. A. M. Isa, M. I. Younis, R. Abdullah, "G2Way – A Backtracking Strategy for Pairwise Test Data Generation", in the 15th IEEE Asia-Pacific Software Engineering Conference, Beijing, China, 2008, pp. 463-470.

[2] D. M. Cohen, S. R. Dalal, M. L. Fredman, G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", in IEEE Transactions on Software Engineering, vol. 23, 1997, pp. 437-444.

[3] Y. Lei, R. Kacker, D. Kuhn, V. Okun, J. Lawrence, "IPOG/IPOD: Efficient Test Generation for Multi-Way Software Testing", in Journal of Software Testing, Verification, and Reliability, vol. 18, 2009, pp.125-148.

[4] M. B. Cohen, "Designing Test Suites for Software Interaction Testing", in Computer Science, University of Auckland, Ph.D, New Zealand, 2004.

[5] D. M. Cohen, S. R. Dalal, A. Kajla, G. C. Patton, "The Automatic Efficient Test Generator (AETG) System", in the 5th International Symposium on Software Reliability Engineering, Monterey, CA, USA, 1994, pp. 303-309.

[6] Y. Lei, K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing", in the 3rd IEEE International. High-Assurance Systems Engineering Symp, Washington, DC, USA, 1998, pp. 254-261.

[7] T. Shiba, T. Tsuchiya, T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing", in the 28th Annual International Computer Software and Applications Conf. (COMPSAC'04), Hong Kong, 2004, pp. 72-77.

[8] K. C. Tai, Y. Lei, " A Test Generation Strategy for Pairwise Testing", in IEEE Transactions on Software Engineering, vol. 28, 2002, pp. 109-111.

[9] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, B. M. Horowitz, "Model Based Testing in Practice", in the International Conf. on Software Engineering (ICSE), 1999, pp. 285–294.

[10] D. R. Kuhn, M. J. Reilly, "An Investigation of the Applicability of Design of Experiments to Software Testing", in the 27th NASA/IEEE Software Engineering Workshop, 2002, pp. 69-80.

[11] D. R. Kuhn, V. Okum, "Pseudo-Exhaustive Testing for Software", in: the 30th Annual IEEE/NASA Software Engineering Workshop (SEW '06), 2006, pp. 25-27.

[12] D. R. Kuhn, D. R. Wallace, A. M. Gallo, "Software Fault Interactions and Implications for Software Testing", in IEEE Transactions on Software Engineering vol. 30, June 2004, pp. 418-421.

[13] J. Yan, J. Zhang, "A Backtracking Search Tool for Constructing Combinatorial Test Suites",in Journal of Systems and Software - Elsevier, vol. 81, October 2008 pp. 1681-1693.

[14] R. Bryce, C. J. Colbourn, "Prioritized Interaction Testing for Pairwise Coverage with Seeding and Avoids", in Information and Software Technology Journal (IST, Elsevier), vol. 48, October 2006, p. 960-970.

[15] D. R. Kuhn, Y. Lei, R. Kacker, "Practical Combinatorial Testing: Beyond Pairwise", in IT Professional- IEEE Computer Society vol. 10, May 2008, pp. 19-23.

[16] D. A. Bader, W. E. Hart, C. A. Phillips, "Parallel Algorithm Design for branch and bound", in Tutorials on Emerging Methodologies and Applications in Operations Research. Mathematics and Statistics, vol. 76. Springer, New York , 2005 pp. 5.1- 5.44.

[17] R. Setia, A. Nedunchezhian, S. Balachandran, "A New Parallel Algorithm for Minimum Spanning Tree Problem". in the 16th Annual IEEE International Conference on High Performance Computing. Cochin, India, 2009, pp 1-25.

[18] Kamal Z. Zamli, Mohammad F.J. Klaib, Mohammed I. Younis, Nor Ashidi Mat Isa, and Rusli Abdullah,"Design and implementation of a t-way test data generation strategy with automated execution tool support" Information Sciences, vol 181,issue 9, May 201,pp 1741-1758.

[19] Mohammad F. J. Klaib, Sangeetha Muthuraman, A.Noraziah, "A Tree Based Strategy for Interaction Testing", in The 5th International Conference on Information Technology 2011 (ICIT2011), AL-Zaytoonah University of Jordan, Faculty of Science & Information Technology, Jordan, , May, 2011, pp 1-5.

[20] Y. Lei, R Kacker, D. R Kuhn, V Okun,J. Lawrence, IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing, Software Testing, Verification and Reliability, vol 18, Issue 3, September 2008 pp 125–148.

[21] Z. Hisham C. Soh, M.. I Younis, S.Abdullah, K. Zamli, "Distributed t-way Test Suite Generation Algorithm for Combinatorial Interaction Testing", in the International conference on IT to Celebrate S.Charmonman's 72nd Birthday (Charm09), Thailand, March 2009, pp. 431-437

[22] M. I. Younis, K. Z. Zamli, "MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems",in ETRI Journal,vol. 32, no. 1, February 2010 pp. 73-83..

[23] M. Grindal, B. Lindstrom, J. Offutty, S. F. Andler, "An Evaluation of Combination Strategies for Test Case Selection". Technical Report HS-IDA-TR-03-001, Department of Computer Science, University of Skövde, 2003.

[24] M. F. Klaib, S. Muthuraman, N. Ahmad, and R. Sidek, "Tree Based Test Case Generation and Cost Calculation Strategy for Uniform Parametric Pairwise Testing",in Journal of Computer Science, Journal of Computer Science, 6 (4), 2010,pp: 425-430.

[25] M. F. J Klaib, S. Muthuraman, N. Ahmad, and R. Sidek, "A Tree Based Strategy for Test Data Generation and Cost Calculation for Uniform and Non-Uniform Parametric Values", in International Symposium on Frontier of Computer Science, Engineering and Applications (CSEA2010), Bradford, UK, 2010,pp 1376 - 1383.

[26] M. F. J Klaib, S. Muthuraman, N. Ahmad, and R. Sidek, "A Parallel Tree Based Strategy for Test Data Generation and Cost Calculation for Pairwise Combinatorial Interaction Testing", in The Second International Conference on Networked Digital Technologies (NDT2010) Charles University in Prague, Czech Republic: Springer, 2010, pp 509-522.

[27] D.R. Kuhn, R.N. Kacker and Y. Lei, "Combinatorial Coverage as an Aspect of Test Quality", Journal of Defense Software Engineering,2014.

[28] D.R. Kuhn, R.N. Kacker and Y. Lei, "Measuring and Specifying Combinatorial Coverage of Test Input Configurations", Innovations in Systems and Software Engineering: a NASA journal, 2014, pp1-15.

[29] J. Torres-Jimenez, I. Izquierdo-Marquez, "Survey of Covering Arrays", in the 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2013), Timisoara, Romania, 23-26, 2013, pp. 20-27.

[30] R.N. Kacker, D.R. Kuhn, Y. Lei, and J.F. Lawrence, "Combinatorial Testing for Software, an Adaptation of Design of Experiments",Measurement, vol. 46, no. 9, 2013, pp. 3745-3752.

[31] X. Niu, C. Nie, Y. Lei, A.T.S. Chan, "Identifying Failure-Inducing Combinations Using Tuple Relationships", in the 6th IEEE International Conference on Software, Testing, Verification and Validation (ICST 2013), Luxembourg, March 18-22, 2013, pp. 271-280.

[32] M.N. Borazjany, L.S.G. Ghandehari, Y. Lei, R.N. Kacker and D.R. Kuhn, "An Input Space Modeling Methodology for Combinatorial Testing", in the 6th IEEE International Conference on Software, Testing, Verification and Validation (ICST 2013), Luxembourg, March 18-22, 2013, pp. 372-381.