ITTW: T-Way Minimization Strategy Based on Intersection of Tuples

Mohammed I. Younis
School of Electrical and Electronics
Universiti Sains Malaysia
14300 Nibong Tebal, Penang, Malaysia
Email: younismi@gmail.com

Kamal Z. Zamli
School of Electrical and Electronics
Universiti Sains Malaysia
14300 Nibong Tebal, Penang, Malaysia
Email: eekamal@eng.usm.my

Abstract—**Despite being an important activity in the software development cycle (i.e. to ensure quality and reliability), exhaustive testing is prohibitively impossible. Systematic minimization strategy based on coverage or t-way parameter interactions are often sought for to help minimize the test space. Unlike coverage based minimization strategy which takes a white box approach, t-way parameter interactions based strategy (i.e. termed t-way testing) takes a black box approach in the sense that no information regarding the implementation is required to perform the reduction. Instead, t-way testing strategy solely relies on parameter interactions between input variables. Building from earlier work, this paper proposes a new strategy, called ITTW, to minimize the number of test set using the intersection of t-way tuples. In doing so, this paper also demonstrates the correctness of the ITTW strategy. Additionally, this paper also compares ITTW strategy against existing strategies, namely: IPOG, IPOD, IPOF, IPOF2, ITCH, Jenny, TVGII, and TConfig. The results demonstrate that ITTW, in most cases, outperforms other existing strategies in terms of test size and execution time.**

Keywords — **Pairwise Testing; Multiway Testing; t-way Testing; Test Plan Data Generation**

I. INTRODUCTION

As a way to evaluate and enhance quality and reliability, software testing is an immensely important activity within the software engineering life cycle [1] [2]. Here, many input parameters and system conditions need to be considered for testing. Often, the consideration of these parameters and conditions yields exorbitantly many tests from the combination space. Nonetheless, considering for all tests within the combination space is prohibitively expensive due to constraints in resources and time. To address this issue, there is a need to sample and selectively capture the most minimum test set within the combination space but not in the expense of sacrificing quality and reliability.

From empirical observation, the number of input variables involved in software failures are relatively small (i.e. in the order of 3 to 6) in some classes of system. If t or fewer variables are known to cause fault, test case can be generated on some *t-way* combinations (i.e. resulting into a smaller set of test case for consideration). Here, t implies the strength of the combinations where t can take up a range of value from $2 \leq t \leq$ the number of parameters, n. If t is to equal to the number of parameters, n, then t is all full strength (i.e. exhaustive combination). Viewing from a different perspective, t also implies the interaction of parameters. Thus, if t = n is chosen, all n parameter interactions are considered and covered. Similarly, if t=n-1 is chosen, only n-1 interaction of parameters are considered and covered.

Building from earlier work, this paper proposes a new strategy, called ITTW, to minimize the number of test set using the intersection of t-way tuples. In doing so, this paper also demonstrates the correctness of the ITTW strategy. Additionally, this paper also compares ITTW strategy against existing strategies, namely: IPOG, IPOD, IPOF, IPOF2, ITCH, Jenny, TVGII, and TConfig. The results demonstrate that ITTW, in most cases, outperforms other existing strategies in terms of test size and execution time.

The rest of the paper is organized as follows. Section 2 highlights the related works on t-way minimization strategy. Section 3 gives the proposed strategy. Section 4 demonstrates the correctness of the proposed strategy Section 5 gives the evaluation of ITTW strategy. Section 6 compares ITTW with other existing strategies. Finally, section 7 states our conclusion and suggestion for future work.

II. RELATED WORK

A number of strategies have been developed on in the past on t-way testing. Although useful, existing strategies appear to focus more on pairwise (t=2) testing (e.g. AETG [3], AETGm [4], OATS (Orthogonal Array Test System) [5][6], G2Way [7], IRPS [8], AllPairs [9], IPO [10], TCG (Test Case Generator) [11], ReduceArray2 [12], DDA (Deterministic Density Algorithm) [13], and OATSGen [14]). Here, we are more interested on a general strategy for t-way test generation for comparative purposes with our work. Thus, what follows is our survey on existing strategies that supports both pairwise and higher order t (i.e. $t \geq 2$).

IBM's Intelligent Test Case Handler (ITCH) [15] uses combinatorial algorithms based on exhaustive search to construct test suites for t-way testing. Although useful as part of IBM's automated test plan generation, ITCH results is not optimized as far as the number of generated test cases is concerned (i.e. some t-way interaction is covered by more than one test).

Jenny [16], TConfig [17] and TVG [18] are public domain tools (available for download) that support t-way testing. While we are able to execute the tools, their

details implementations are not known due to limited information available in the literature.

IPOG [19] is a generalization of a pairwise strategy (i.e. t=2) based on vertical and horizontal extension. IPOG strategy first generates a pairwise test set for the first two parameters. It then continues to extend the test set to generate a pairwise test set for the first three parameters and continues to do so for each additional parameter until all the parameters of the system are covered via horizontal extension. If required (i.e. for interaction coverage), IPOG also employs vertical extension in order to add new tests after the completion of horizontal extension.

More recently, some variants to IPOG exist namely: IPOD [20], IPOF [21], and IPOF2 [22]. In general, they share common property that search through one column at a time. IPOD is optimized to speed up IPOG [20], whilst IPOF and IPOF2 are designed to reduce the test size of IPOG [22].

As seen above, much useful and significant progress has been achieved as far the support for t-way testing is concerned. Nonetheless, constructing the minimum test set for t-way testing is a NP-complete problem [23][24], that is, it is unlikely an efficient algorithm exists that can always generate an optimal test set (i.e. each t-way pair is covered only once) [23]. Taking the aforementioned challenges, it is the development of an optimum strategy, called ITTW by systematic intersection of the tuples, for t-way testing is the main focus of this paper.

III. ITTW STRATEGY

The ITTW strategy starts by generating all tuples (i.e. interaction pairs). Then, the ITTW strategy puts "don't care " values for non interacting elements. Next, the ITTW strategy tries to optimize "don't care " values by means of systematic intersection of the tuples with the remaining tuples in a greedy fashion, that is, to ensure the final selected test case has the most covered pairs. For clarity the complete algorithm is given in Fig. 1.

Algorithm ITTW-Test (int t, ParameterSet ps)
{
1. initialize test set ts to be an empty set
2. denote the parameters in ps, in an arbitrary order, as P1, P2, …, and Pn
4. for (int i = t; i ≤ n; i ++){
5. let π be the set of t-way combinations of values involving parameter Pi and t -1 parameters among the first i – 1 parameters
6. }// i loop
7. while (π not empty){
8. rearranges π in decreasing order.
9. Choose the first tuple and generate test case (τ) that combine maximum number of tuples
10. delete the tuples covered by τ, add τ to local ts
11. } //while
12. return ts;
}

Figure 1. The ITTW Strategy

In order to illustrate how the ITTW strategy works, consider the following running example. Ideally, this running example serves as a model of a typical software system implementation (see Fig. 2).

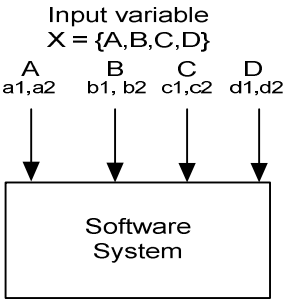

Figure 2. Model of a typical software system implementation

Referring to Fig. 2, let the input variable consists of a set X = {A, B, C, D}. For simplicity sake, let us assume that the starting test case for X, termed *base test case*, has been identified in Table 1 (with 4 parameters and 2 values). Here, symbolic values (e.g. a1, a2, b1, b2, c1, c2) are used in place of real data values to facilitate discussion.

TABLE I. BASE VALUES

	Input Variables			
Base Values	**A**	**B**	**C**	**D**
	a1	b1	c1	d1
	a2	b2	c2	d2

In this case, at full strength t= 4 (i.e. exhaustive combinations), the number of test cases = (the number of values) $^{\text{the number of parameters}} = 2^4 = 16$.

TABLE II. EXHAUSTIVE TEST SUITE

	Input Variables			
Base Values	**A**	**B**	**C**	**D**
	a1	b1	c1	d1
	a2	b2	c2	d2
All possible values	a1	b1	c1	d1
	a1	b1	c1	d2
	a1	b1	c2	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c1	d2
	a1	b2	c2	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c1	d2
	a2	b1	c2	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c1	d2
	a2	b2	c2	d1
	a2	b2	c2	d2

These 16 exhaustive combinations can be generated based on a simple technique (see Table 2). Here, one can view each variable as a column matrix. For column A, one must repeat the input a1 8 times followed by a2 (also 8

times) to reach 16. This is because there are 16 combinations with 2 specified inputs (i.e. 16/2 = 8 times). Now for column B, one must alternately repeat the input b1 4 times followed by b2 (also 4 times) to reach 16. Similarly, for column C, one must repeat c1 2 times followed by c2 (also 2 times) to reach 16. Finally, for column D, one can alternately repeat d1 and d2 to reach 16.

Using the same example, Fig. 3 illustrates how the ITTW strategy works. Here, assume that the interaction strength is 3 (i.e. t=3). The possible 3-way interaction tuples are between ABC, ABD, ACD, and BCD. In each of the 3-way interaction tuples, there exists "don't care " that can be optimized in the merged π set. In this case, the optimization depends on the intersection of tuples. If the intersection covers the most uncovered pair, then the result of the intersection is selected in the final selected test set.

As seen in Fig. 3, applying the ITTW strategy yield a final test set of 8 test cases only. Here, there is a reduction of 50% (i.e. from 16 in the exhaustive test to 8).

IV. INTERACTION COVERAGE ANALYSIS

The key concern on all t-way combination strategy is the fact that whether or not all the t-way interaction of variables are covered. In our earlier example given in Section 3, all the 3 way interaction of the variables needs to be covered at least once (i.e. to demonstrate its correctness)..

The 3 interaction of variables (ABC, ABD, ACD, and BCD) are tabulated in Table 3 along with the analysis in terms of the tuples occurrences. Here, the occurrences are analyzed from the result in Fig. 3 for ITTW strategy.

The analysis in Table 3 demonstrates that all the 3 way pairs are covered by at least once, so we can conclude that there implementation is correct. Specifically for all t-way combination strategy, the 3 way pairs of {a1,b1,c2}, {a2,b1,c2}, {a2,b2,c1}, {a1,b1,d1}, {a2,b2,d2}, {b1,c2,d1}, {b2,c1,d2} are covered only once. From this analysis, it is evident that each tuple occurs at most once indicating the resulting final tests are absolutely minimal.

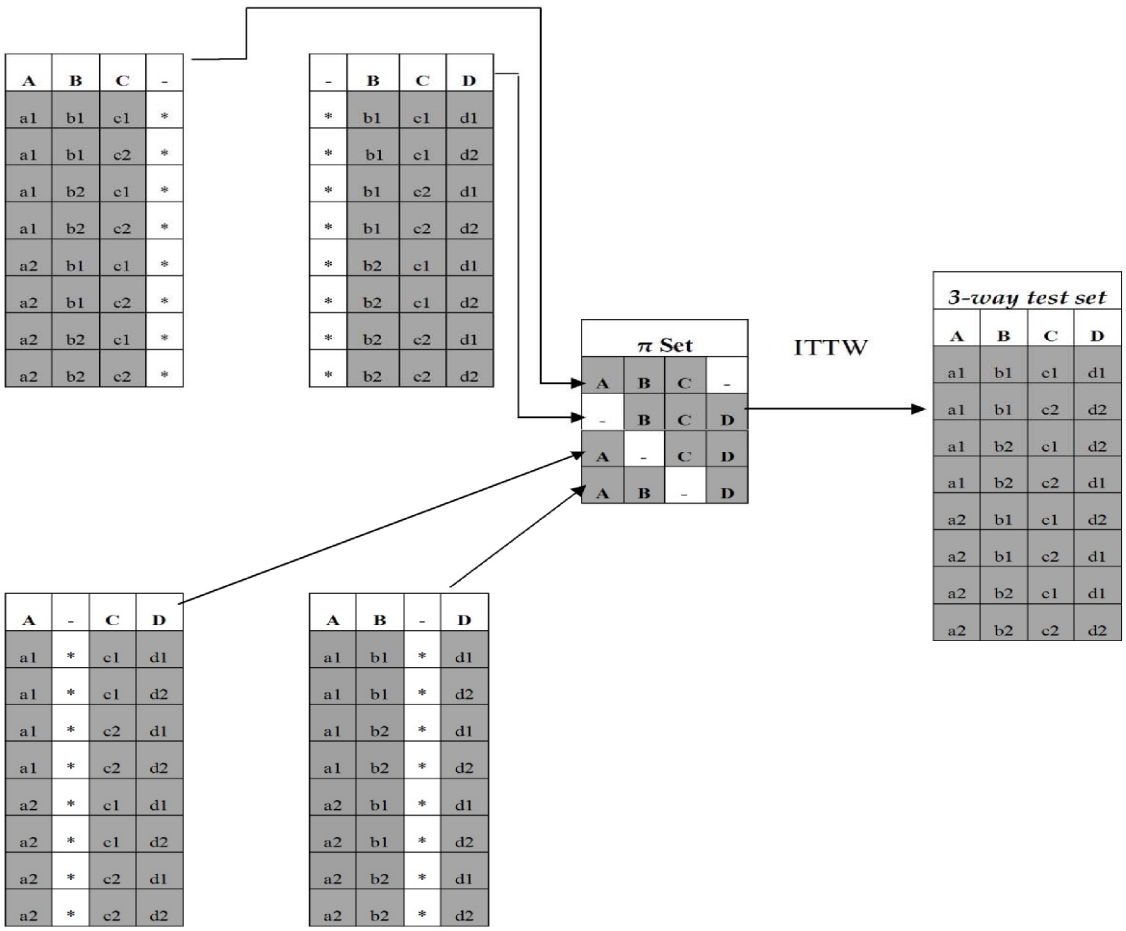

Figure 3. ITTW Illustrative Example

TABLE III. TUPLES ANALYSIS

Interaction	Tuples	Occurrences
ABC	a1,b1,c1	1
	a1,b1,c2	1
	a1,b2,c1	1
	a1,b2,c2	1
	a2,b1,c1	1
	a2,b1,c2	1
	a2,b2,c1	1
	a2,b2,c2	1
ABD	a1, b1,d1	1
	a1,b1,d2	1
	a1,b2,d1	1
	a1,b2,d2	1
	a2, b1,d1	1
	a2,b1,d2	1
	a2,b2,d1	1
	a2,b2,d2	1
ACD	a1,c1,d1	1
	a1,c1,d2	1
	a1,c2,d1	1
	a1,c2,d2	1
	a2, c1,d1	1
	a2,c1,d2	1
	a2,c2,d1	1
	a2,c2,d2	1
BCD	b1,c1,d1	1
	b1,c1,d2	1
	b1,c2,d1	1
	b1,c2,d2	1
	b2, c1,d1	1
	b2,c1,d2	1
	b2,c2,d1	1
	b2,c2,d2	1

V. EVALUATION

In this section, we aim to evaluate the ITTW strategy in terms of test size as well as execution time. In order to perform the evaluation, three groups of experiment are applied.

- In the first group, we fix the number of parameters (p) to 10, and varying the values (v) from 2 to 10.
- In the second group, we fix (v) to 5, and changing (p) from 6 to 26. In both first and second groups we fix t to 4.
- In the third group, we fix p=10, and v=5 and changes t from 2 to 7.

The results for the first, second, and third groups are tabulated in Tables 4, 5, and 6 respectively. Here, all the results are obtained using Windows XP, with 1.6 GHz CPU, 1 GB RAM, and JDK 1.6 installed on it. It should be noted that the time is recorded in second.

TABLE IV. RESULT FOR GROUP 1

v	Size	Time (second)
2	41	0.078
3	219	0.188
4	726	0.486
5	1604	0.984
6	3938	18.234
7	5540	25.94
8	12781	97.063
9	20489	158.765
10	31395	342.703

TABLE V. RESULT FOR GROUP 2

p	Size	Time (second)	p	Size	Time (second)
5	625	0.004	16	2575	19.812
6	625	0.015	17	2711	29.609
7	1125	0.125	18	2854	49.016
8	1248	0.25	19	2972	79.657
9	1415	0.5	20	3104	131.656
10	1604	0.984	21	3212	199.282
11	1756	1.671	22	3329	299.031
12	1942	2.984	23	3435	378.532
13	2128	4.922	24	3550	495
14	2272	7.438	25	3647	614.469
15	2429	12.547	26	3756	798.531

TABLE VI. RESULT FOR GROUP 3

t	Size	Time (second)
2	47	0.016
3	279	0.11
4	1604	5.360
5	8393	6.891
6	39701	28.609
7	175314	70.350

From Table 4, we plot the test size against the number of values as given in Fig. 4. We also plot the execution time versus number of values as shown in Fig. 5.

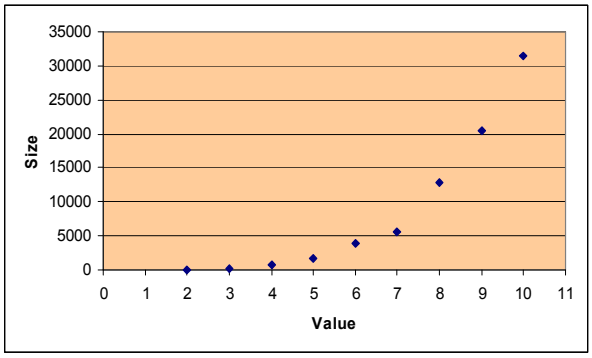

Figure 4. Test Size versus Values

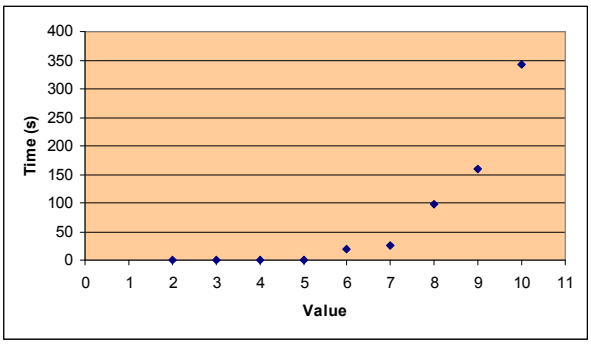

Figure 5. Execution Time versus Value

Referring to Fig. 4 and Fig. 5, we conclude that both the test size and execution time are proportional quadratically with the number of values.

Next, we plot the test size versus number of parameters as given in Fig. 6. Then, we also plot the execution time versus number of parameters as shown in Fig. 7.

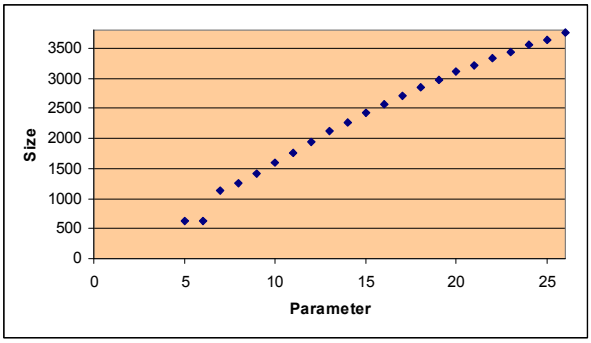

Figure 6. Test Size versus Parameter

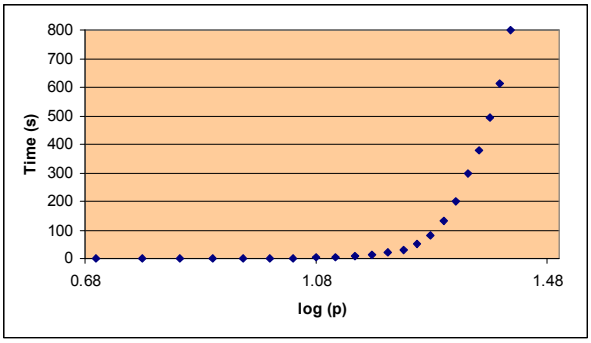

Figure 7. Execution Time versus log (p)

Referring to Fig. 6, we conclude that the test size grows logarithmically with the increasing the number of parameters. From Fig. 7, we note that the execution time grows quadratically with respect to logarithmic scale of parameters.

Finally, we are also interested to investigate the characteristics of the test size and execution time against varying the strength of coverage (t). In this case, we plot the test size versus (t) from Table 6 as given in Fig. 8. Similarly, we also plot the execution time versus (t) as shown in Fig. 9.

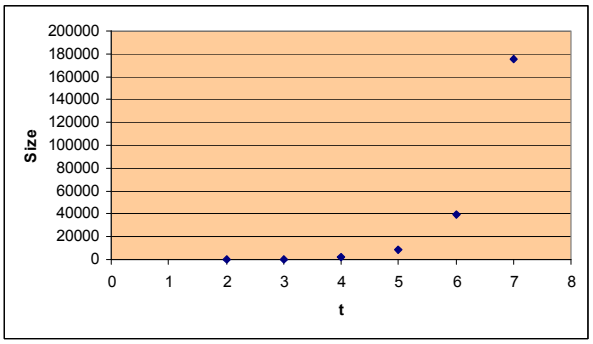

Figure 8. Test Size versus (t)

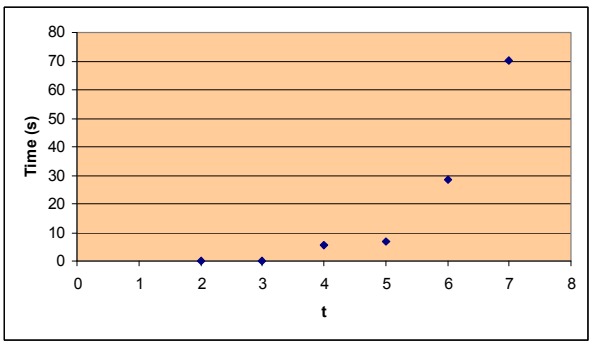

Figure 9. Execution Time versus (t)

From Fig. 8 and Fig. 9, it is evident that the test size as well as execution time grows exponentially as the strength of coverage (t) increases. Putting all together, the test size as well as the execution time can be summoned in $O(v^{t} \log p)$.

VI. COMPARISON

In order to compare ITTW with other existing strategies, we select a common case study. The common case study involving the Traffic Collision Avoidance System Module (TCAS) from the Federal Aviation Administration [20] [21]. TCAS consists of 12 parameters (2 10-valued parameters, 1 4-valued parameters, 2 3-valued parameters, and 7-2 valued parameters).

In our experiments, we execute ITTW and all the other strategies (i.e. IPOG, IPOD, IPOF, IPOF2, ITCH, Jenny, TConfig, and TVG) within the same system configuration and running environment in order to ensure fair comparison. In this case, the system consists of a desktop running Windows XP with 1.6GHZ CPU and 1GB memory, with JDK 1.6 installed. The results of subjecting the above mentioned strategies to TCAS module are tabulated in Table 7 with t=4. Here, it should be noted that all strategies are implemented using Java programming language except Jenny, which is implemented using C language.

Referring to Table 7, it is evident that ITTW outperforms other existing strategies in both test size and execution time. Regarding the test size IPOF and IPOG are more comparable to ITTW. Regarding the execution time IPOD is more comparable to ITTW.

TABLE VII. RESULT FOR TCAS MODULE WITH T=4

Strategy	Size	Time (second)
ITTW	**1283**	**0.296**
IPOG	1367	3.081
IPOD	2522	0.305
IPOF	1363	1.812
IPOF2	1690	1.521
ITCH	1484	5201
Jenny	1527	3.45
TConfig	1476	>20 hours
TVGII	1599	102.045

VII. CONCLUSION

Summing up, this paper has described the development of a deterministic t-way strategy (called ITTW) for systematic software test data minimization. Results demonstrate that the strategy works well in terms of the coverage of the tuples. As part of our future work, we are investigating the implementation of ITTW on the Grid Environment.

ACKNOWLEDGMENT

The work described in this paper is funded by the eScience Fund – Development of a Mobile Agent Based Parallel and Automated Java Testing Tool" from the MOSTI.

REFERENCES

[1] P.E. Ammann, and A.J. Offutt, "Using Formal Methods to Derive Test Frames in Category-Partition Testing". In *Proc. of the 9th Annual Conference on Computer Assurance (COMPASS'94)*, IEEE CS Press, June 1994, pp. 69-80.

[2] B. Beizer, "*Software Testing Technique"*. Thomson Computer Press, 2nd Edition, 1990.

[3] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design. " *IEEE Transactions on Software Engineering 23(7)*, July 1997, pp. 437-444.

[4] M. B. Cohen, "Designing Test Suites for Software Interaction Testing." PhD thesis. University of Auckland, 2004.

[5] K. A. Bush, "Orthogonal Arrays of Index Unity", Annals of Mathematical Statistics, Vol. 23, pp. 426-434, 1952.

[6] R. Mandl, "Orthogonal Latin Squares: an Application of Experiment Design to Compiler Testing," Communications of the ACM, Vol. 28 (10), pp. 1054-1058, 1985.

[7] M.F.J. Klaib, K. Z. Zamli, N. A. Mat .Isa, M. I..Younis, and 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.

[8] M. I. Younis, K. Z. Zamli, and N. A. M. ISA, "IRPS - An Efficient Test Data Generation Strategy for Pairwise Testing", in Proc. of the 12th Intl. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems (KES2008), Zagreb, LNAI Vol. 5177, pp. 493-500.

[9] http://www.satisfice.com/testmethod.shtml.

[10] Y. Lei and K. C. Tai, "In-Parameter-Order: A Test Generating Strategy for Pairwise Testing", IEEE Transaction on Software Engineering, Vol. 28 (1), pp. 1-3 2002.

[11] http://www.alphaworks.ibm.com/tech/cts. Last accessed on August 6, 2009.

[12] G. T. Daich, "Testing Combinations of Parameters Made Easy," in Proc. of the IEEE Systems Readiness Technology Conference (AUTOTESTCON 2003), Sept. 2003, pp. 379- 384.

[13] R. Bryce and C. J. Colbourn, "The Density Algorithm for Pairwise Interaction Testing", Software Testing, Verification and Reliability, Vol. 17, pp. 159 - 182 2007.

[14] R. Krishnan, S. M. Krishna, and P. S. Nandhan, "Combinatorial Testing: Learnings from our Experience", ACM SIGSOFT Software Engineering Notes, Vol. 32, pp. 1-8, May 2007.

[15] http://www.alphaworks.ibm.com/tech/whitch. Last accessed on August 6, 2009.

[16] http://www.burtleburtle.net/bob/math. Last accessed on August 6, 2009.

[17] http://www.site.uottawa.ca/~awilliam. Last accessed on August 6, 2009.

[18] http://sourceforge.net/projects/tvg. Last accessed on August 6, 2009.

[19] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "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, March 2007, pp. 549-556.

[20] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, IPOG/IPOG-D: Efficient Test Generation for Multi-way Combinatorial Testing, Software Testing, Verification and Reliability, v 18(3), 125-148, 2008.

[21] R.N. Kacker, J.Y. Lei, J. F. Lawrence, M. Forbes, D. R. Kuhn, V. Hu, R. M. Rivello, T. Xie, R. Bryce, S. Sampath, S. Chaki,and A. Gurfinkel, Automated Combinatorial Testing for Software Systems ,Mathematical and Computational Sciences Division, NIST Report, January 2008, pp. 38-40.

[22] M. Forbes, J. Lawrence, Y. Lei, R.N. Kacker, and D. R. Kuhn, Refining the In-Parameter-Order Strategy for Constructing Covering Arrays, Journal of Research of the National Institute of Standards and Technology, Volume 113, Number 5, October 2008. pp. 287-297.

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

[24] D. R. Kuhn and V. Okun, "Pseudo-exhaustive Testing For Software," in Proc. of the 30th NASA/IEEE Software Engineering Workshop, April 25-27, 2006.