2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation

Interaction Testing: From Pairwise to Variable Strength Interaction

Kamal Z. Zamli
School of Electrical and Electronic Engineering
Universiti Sains Malaysia, Engineering Campus
14300 Nibong Tebal, Penang, Malaysia
e-mail:eekamal@eng.usm.my

Mohammed I.Younis
School of Electrical and Electronic Engineering
Universiti Sains Malaysia, Engineering Campus
14300 Nibong Tebal, Penang, Malaysia

Abstract—**Although desirable as an important activity for quality assurances and enhancing reliability, complete and exhaustive software testing is prohibitively impossible due to resources as well as timing constraints. While earlier work has indicated that uniform pairwise testing (i.e. based on 2-way interaction of variables) can be effective to detect most faults in a typical software system, a counter argument suggests such conclusion cannot be generalized to all software system faults. In some system, faults may also be non-uniform and caused by more than two parameters. Considering these issues, this paper explores the issues pertaining to t-way testing from pairwise to variable strength interaction in order to highlight the state-of-the-art as well as the current state-of-practice.**

Keywords-t-way testing; interaction testing; variable strength interaction;

I. INTRODUCTION

In line with increasing consumer demands for new functionalities and innovations, software applications grew tremendously in size over the last 15 years. This sudden increase has a profound impact as far as testing is concerned. Here, the size of test suite for all combinations of inputs grows significantly making exhaustive testing prohibitively and practically impossible. To address the aforementioned issues, much research is now focusing on interaction testing (termed *t-way strategy*). As the name suggests, interaction testing focuses on the detecting failures caused by t-way interaction of variables (where t indicates the interaction strength). The rationale for interaction testing stemmed from the fact that from empirical observation, the number of input variables involved in software failures is relatively small (i.e. in the order of 3 to 8), in some classes of software. If t or fewer variables are known to cause fault, test cases can be generated on some t-way combinations (i.e. resulting into a smaller set of test cases for consideration).

Earlier work on adopting interaction testing gives mixed results (e.g. ACTL [1, 2], Jenny [3], ITCH [4], TConfig [5], and TVG [6]). In some work, 2-way testing (also termed pairwise) testing appears to be sufficiently adequate for achieving good test coverage. For instance, Klaib et al demonstrates that pairwise testing can be effective to detect faults in a Graphical User Interface program with over 80% method, block, and line coverage [7].

Although useful, pairwise testing result cannot be generalized to all software system. In some system, faults may also be non uniform and caused by more than two parameters. Considering these issues, this paper explores the issues pertaining to t-way testing from pairwise to variable strength interaction in order to highlight the state-of-the-art as well as the current state-of-practice.

This paper is organized as follows. Section II highlights the t-way strategy fundamentals. Section III explores pairwise and variable strength interaction testing. Section IV highlights some of the related work. Finally, section V gives our closing remark.

II. T-WAY STRATEGIES FUNDAMENTAL

Mathematically, t-way strategies can be abstracted to a covering array. Throughout out this paper, the symbols: p, v, and t are used to refer to number of parameters (or factor), values (or levels) and interaction strength for the covering array respectively. Referring to Table I, the parameters are A,B,C, and D whilst the values are (a1, a2, b1, b2, c1, c2). Earlier works suggested three definitions for describing the covering array. The first definition is based on whether or not the numbers of values for each parameter are equal. If the number of values is equal (i.e. uniformly distributed), then the test suite is called Coverage Array (CA). Now, if the number of values in non-uniform, then the test suite is called Mixed Coverage Array (MCA) [8]. Finally, Variable Strength Covering array (VCA) refers to case when a smaller subset of covering arrays (i.e. CA or MCA) constitutes a larger covering array.

Normally, the CA takes parameters of N, t, p, and v respectively. For example, CA (9, 2, 4, 3) represents a test suite consisting of 9x4 arrays (i.e. the rows represent the size of test cases (N), and the column represents the parameter (p)). Here, the test suite also covers 2-way interaction for a system with 4 3 valued parameter.

Alternatively, MCA takes parameters of N, t, and Configuration (C). In this case, C captures the parameters and values of each configuration in the following format: $v_1^{\ p1}\ v_2^{\ p2}$, ….. $v_n^{\ pn}$ indicating that there are p1 parameters with v1 values, p2 parameters with v2 values, and so on. For example, MCA (1265, 4, $10^2 4^1 3^2 2^7$) indicates the test size of 1265 which covers 4-way interaction. Here, the configuration takes 12 parameters: 2 10 valued parameter, 1 4 valued parameter, 2 3 valued parameter and 7 2 valued parameter. Such notation can also be applicable to CA (e.g. CA (9,2,4,3) can be rewritten as CA (9,2, 3^4)).

In the case of VCA, the parameter consists of N,t, C, and Set (S). Here, S consists of a multi-set of disjoint covering array with strength larger t. For example, VCA (12, 2, $3^2 2^2$, {CA (3,$3^1 2^2$)}) indicates the test size of 12 for pairwise interaction (with 2 3 valued parameter and 2 2 valued parameter) and 3-way interaction (with 1 3 valued parameter and 2 2 valued parameter).

III. RUNNING EXAMPLE

In order to aid the discussion, consider the following software system example in Fig. 1.

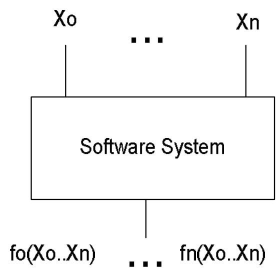


Figure 1. Model of a Typical Software System

Assume that the input set X = {x_0.... x_n} significantly affects the output, noted as fo (x_0.... x_n) to fn (x_0.... x_n). If X is known to take a set of data values: D(x_0), D(x_1)... D(x_n), then the system must be tested against the set of all possible combinations of D. Here, the result is an ordered n-tuples {d0,d1…dn} where each d_i is an element of D(x_i). The size of the test suite would be the product size of all D(x):

$$T_{suite} = \{ D(x_0) \times D(x_1) \times \ldots D(X_n)\}$$

Obviously, the test suite T_{suite} can grow exponentially with the increase size of data element in the set D(x_0), D(x_1)... D(x_n). As far as the actual test data of T_{suite} is concerned, one can consider the interaction between all n variables x0, x21, x2...xn, termed, *exhaustive test*. Optionally, one can also consider the interaction of any t-way interactions of variables. Here, the value of t can take the minimum of 2 and the maximum of n-1. As a running example, let us assume that the starting test case for X, termed *base test case*, has been identified in Table I. 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 DATA VALUES

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

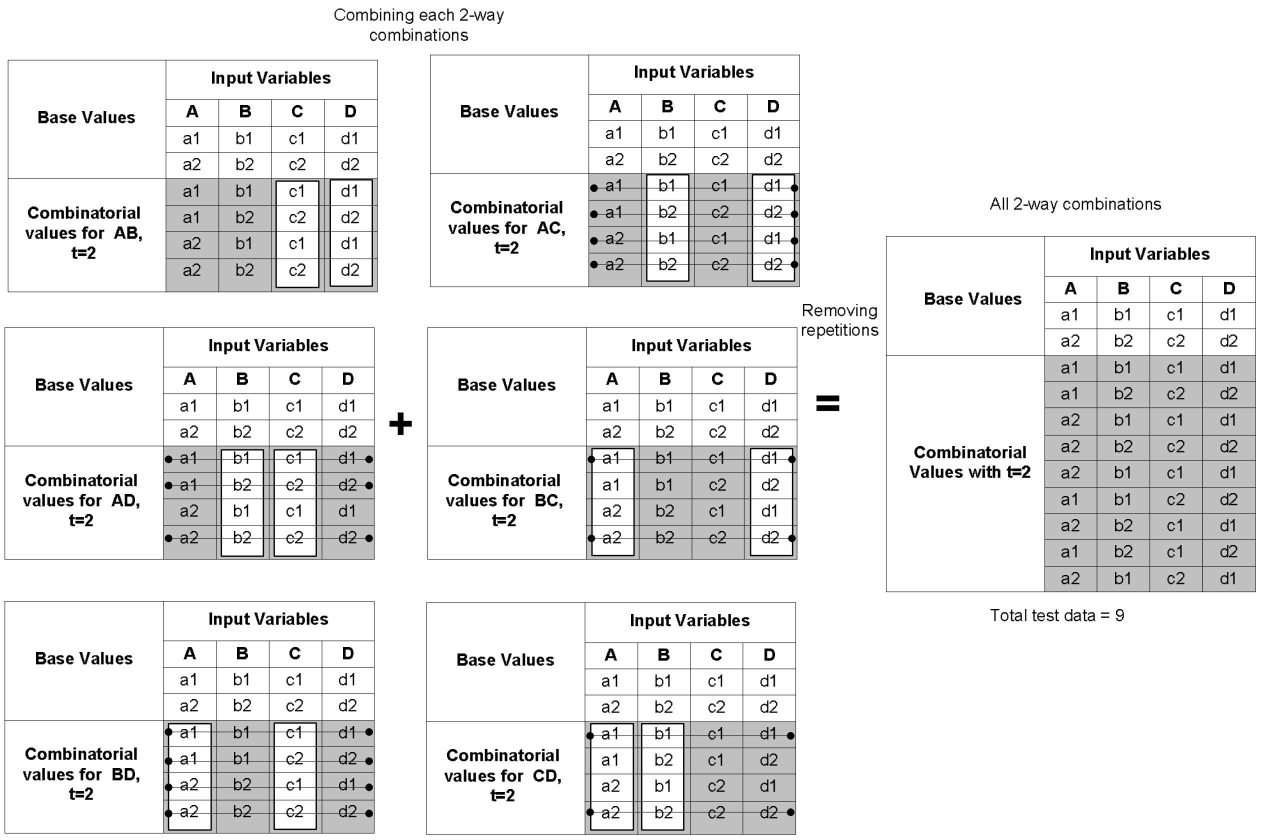
Here, at full strength of interaction (i.e. t=4), we can get all exhaustive combination. In this case, the exhaustive combinations would be 24 = 16 and can be generated using a simple technique (see Table II). Here, one can view each variable as column matrix. For column A, one must repeat the input 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. Here, at full strength of interaction (i.e. t=4), we can get all exhaustive combination. In this case, the exhaustive combinations would be 24 = 16 and can be generated using a simple technique (see Table II). Here, one can view each variable as 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.

TABLE II. EXHAUSTIVE COMBINATION

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

As highlighted earlier, considering all exhaustive interaction is infeasible for large number of parameters and values. If we consider uniform pairwise interaction (i.e. t=2), we can have interactions between AB, AC, AD, BC, BD, and CD (see Table III). Here, when parameters AB are considered, the values for parameter CD are don't cares (i.e. any random valid values for parameter CD suffice). Similarly, when parameters AC are considered, values for parameter BD are don't cares. When parameters AD are considered, values for parameter BC are don't care. When parameters BC are considered, values for parameter AD are don't cares. When parameters BD are considered, values for parameter AC are don't cares. Finally, when parameters CD are considered, values for parameter AB are don't cares. Combining these results, we note that there are some repetitions of values between some entries. If these repetition is removed, we can get all the combinations at t=2.

TABLE III. PAIRWISE INTERACTION RESULT, CA (9,2, 2^4).


Combining each 2-way combinations
+
Removing repetitions
=
All 2-way combinations
Total test data = 9

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for AB, t=2	a1	b1	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b2	c2	d2

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for AC, t=2	~~a1~~	~~b1~~	~~c1~~	~~d1~~
	~~a1~~	~~b2~~	~~c2~~	~~d2~~
	~~a2~~	~~b1~~	~~c1~~	~~d1~~
	~~a2~~	~~b2~~	~~c2~~	~~d2~~

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for AD, t=2	~~a1~~	~~b1~~	~~c1~~	~~d1~~
	~~a1~~	~~b2~~	~~c2~~	~~d2~~
	a2	b1	c1	d1
	~~a2~~	~~b2~~	~~c2~~	~~d2~~

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for BC, t=2	~~a1~~	~~b1~~	~~c1~~	~~d1~~
	a1	b1	c2	d2
	a2	b2	c1	d1
	~~a2~~	~~b2~~	~~c2~~	~~d2~~

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for BD, t=2	~~a1~~	~~b1~~	~~c1~~	~~d1~~
	~~a1~~	~~b1~~	~~c2~~	~~d2~~
	~~a2~~	~~b2~~	~~c1~~	~~d1~~
	~~a2~~	~~b2~~	~~c2~~	~~d2~~

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for CD, t=2	~~a1~~	~~b1~~	~~c1~~	~~d1~~
	a1	b2	c1	d2
	a2	b1	c2	d1
	~~a2~~	~~b2~~	~~c2~~	~~d2~~

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=2	a1	b1	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b2	c2	d2
	a2	b1	c1	d1
	a1	b1	c2	d2
	a2	b2	c1	d1
	a1	b2	c1	d2
	a2	b1	c2	d1

TABLE IV. INTERACTION ELEMENT ANALYSIS

Interactions	Elements	Occurrences		Interactions	Elements	Occurrences
AB	a1 b1	2		AC	a1 c1	2
	a1 b2	2			a1 c2	2
	a2 b1	3			a2 c1	3
	a2 b2	1			a2 c2	2
AD	a1 d1	1		BC	b1 c1	3
	a1 d2	3			b1 c2	2
	a2 d1	4			b2 c1	2
	a2 d2	1			b2 c2	1
BD	b1 d1	4		CD	c1 d1	4
	b1 d2	1			c1 d2	1
	b2 d1	1			c2 d1	1
	b2 d2	3			c2 d2	3

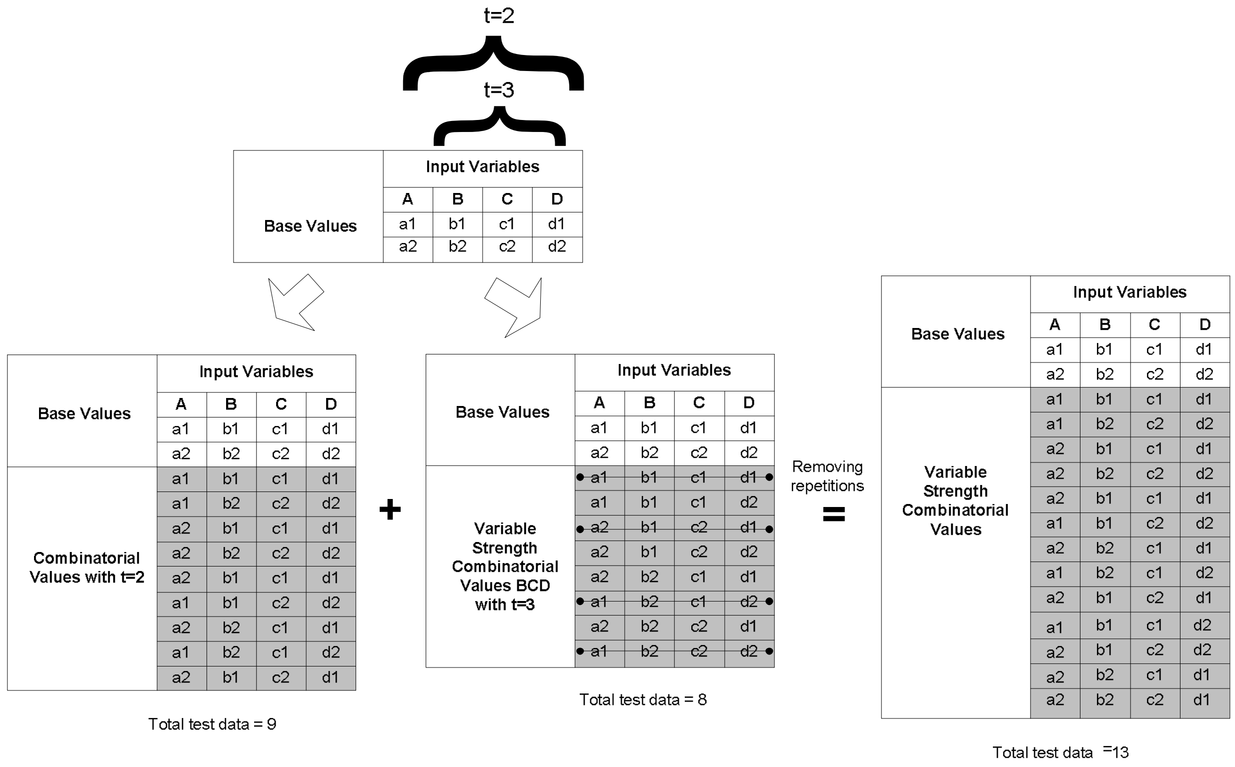
Here, we note that the test suite has been reduced from 16 (for exhaustive combination) to 9 (for t=2), a saving of nearly 43 percent. Using the notation discussed earlier, we can write this test suite as $T_{suite} = CA\ (9,3,\ 2^4)$.

In this case, we note that by relaxing the interaction, we can systematically reduce the test data for consideration significantly. Ensuring that all interaction elements appear at least once is important to ascertain the correctness of the t-way strategy being adopted. Here, the number of interaction elements can be predicted using the following expression.

$$\text{Interaction elements} = \binom{p}{t} v^{t} = \frac{p!}{t!(p-t)!} v^{t} \qquad (1)$$

It is worth noting here that unlike CA which has a static equation to determine the number of interaction elements, MCA and VCA requires explicit calculation based on the number of defined parameters and values in order to determine the number of tuples. This is performed by considering the sum of products of each individual's interaction sets. For example, when MCA (N, 3, $3^1 2^3$) is considered, the total number of interaction elements = 3*2*2+ 3*2*2 + 3*2*2+ 2*2*2= 44. When MCA (N, 3, 4,

TABLE V. VARIABLE STRENGTH INTERACTION RESULT, VCA (13,2,2^4, {CA(3,2^3)})



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

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=2	a1	b1	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b2	c2	d2
	a2	b1	c1	d1
	a1	b1	c2	d2
	a2	b2	c1	d1
	a1	b2	c1	d2
	a2	b1	c2	d1

Total test data = 9

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Variable Strength Combinatorial Values BCD with t=3	~~a1~~	~~b1~~	~~c1~~	~~d1~~
	a1	b1	c1	d2
	~~a2~~	~~b1~~	~~c2~~	~~d1~~
	a2	b1	c2	d2
	a2	b2	c1	d1
	~~a1~~	~~b2~~	~~c1~~	~~d2~~
	a2	b2	c2	d1
	~~a1~~	~~b2~~	~~c2~~	~~d2~~

Total test data = 8

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

Total test data =13

$3^1 2^3$) is considered, the total number of interaction elements = 3*2+ 3*2+3*2+2*2 + 2*2+ 2*2= 28. Now, for MCA (N, 4, $3^1 2^3$), the total number of interaction elements =3*2*2*2= 24. Finally, in the case of VCA, the total number of interaction elements is the sum of tuples from all the disjoint set of CAs and MCAs. For instance, for VCA (12, 2, $3^2 2^2$, {CA (3,$3^1 2^2$)}), the total number of interaction elements = 3*3 + 3*2 + 3*2 + 3*2 + 3*2 + 2*2 = 37.

Going back to our running example, the complete analysis of the final result from Table IV demonstrates that some interaction elements appears four, three and two times respectively, indicating that this solution is not the most optimum. This phenomenon is expected as the choice of values when "don't care" happen are randomly selected. As this combinatorial explosion problem is NP complete, significant efforts in the literature are focusing to obtain a t-way strategy that give optimum results, that is, most the interaction elements appear at most once, whenever possible. Here, the reduction tends to be maximized, hence, reducing the test costs.

As the interaction strength increases (i.e. from t=2), the combinations to be considered also increases significantly. To illustrate this issue, we consider an aircraft collision avoidance system (TCAS) module from the Federal Aviation Administration which has been used as case study in other related works [9-12] [2]. Here, the TCAS module has twelve parameters: 7 parameters have 2 values, 2 parameters have 3 values, 1 parameter has 4 values, and 2 parameters have 10 values. Pairwise testing requires 100 test cases. 3-way testing requires 400 test cases. 4-way requires 1265. 5-way requires 4196. 6-way requires 10851. 7-way requires 26061. 8-way requires 56742. 9-way requires 120361. 10-way requires 201601. 11-way requires 230400. Finally, 12-way requires 460800.

For large system with many parameters, considering higher order t-way test set can lead toward combinatorial explosion problem. On one side of the coin, we can consider pairwise testing in order to get the most minimum tests. However, we cannot have the guarantee that we will find faults caused by higher order interactions (e.g. t=3 or t=4). On the other side of the coin, if we consider high order interactions, more costs would be expected as the test size is likely to increase accordingly.

Practically, in many real applications, interaction may not be uniform throughout all parameters. Here, a particular subset of variables can have a higher interaction dependency than other variables (indicating failures due to the interaction of that subset may have more significant impact to the overall system). For example, consider a subset of components that control a safety-critical hardware interface. We want to use stronger coverage in that area (i.e. t=3). However, the rest of our components may be sufficiently tested using pairwise testing. In this case, we can assign variable coverage strength to each subset of components as well as to the whole system.

To illustrate variable strength t-way interaction, we revisit our running example from Table I. Now, we assume that all interaction is uniform at t=2 for all parameters (i.e. based on our result in Table III. Then, we consider t=3, only for parameters B,C,D. Combining both interaction yields result shown in Table V. Here, the test suite has been reduced from 16 (for exhaustive case) to 13, a saving of

18.75 percent. Using the notation described earlier, we can write this reduction as T_{suite} = VCA $(13,2,2^4, \{CA(3,2^3)\})$. Here, rather than generalizing to all, we have selected BCD as the only variables with t=3.

IV. Related Work

Earlier studies demonstrate that there are two approaches for generating t-way test suite (or covering array for CA, MCA and VCA). These approaches can be either algebraic or computational strategies [8]. Algebraic strategies generate the test suite directly by means of mathematical transformations [11]. Unlike algebraic strategies, computational strategies often rely on the generation of all tuples and search the tuples space to generate the required test suite until all tuples have been covered. In the case where the number of tuples to be considered is significantly large, adopting computational strategies can be expensive especially in terms of the space required to store the tuples and the time required to explicit enumeration. On a positive note, computational approaches are more adaptable for constraint handling [13, 14] and test prioritization [15].

Grindal et al classified combinatorial strategies into two main categories: deterministic and non-deterministic strategies [16]. Given the same parameter values, deterministic strategies produce the same test suite for every run. In contrast, the generated test suite in non-deterministic strategies is highly non-deterministic (i.e. the same input parameter values may lead to different test suite). More recently, Forbes et al reported that deterministic strategies are more preferable than non-deterministic strategies, even though running a non-deterministic strategy multiple times may minimize the test size in some system [1].

As far as usage is concerned, interaction testing has a wide range of applications. Significant efforts in the literature put focus on pairwise testing. Mandl adopts pairwise coverage using Orthogonal Latin Square (OLS) to testing an Ada compiler [17]. Berling and Runeson adopted interaction testing to identify real and false targets in target identification system [18]. Lazi´c and Velaˇsevi´c employed interaction testing on modeling and simulation for automated target-tracking radar system [19]. White has also applied the technique to test Graphical User Interfaces (GUI) [20]. Other applications of interaction testing include regression testing through the GUI [21] and fault localization [22, 23]. Tang and Chen, Boroday, and Chandra et al. investigated circuit testing in hardware environment, proposing test coverage that includes each 2t of the input settings for each subset of t inputs [24-26]. Seroussi and Bshouti explored a comprehensive treatment for circuit testing [27]. In addition, Dumer examined the related questions of isolating memory faults, and adopted binary covering arrays [28].

Concerning the general interaction testing, much work has also been undertaken in the literature. Dunietz et al. demonstrated the need for higher order strength. In this case, Dunietz et al. demonstrated that significant block coverage is obtained when testing with two-way interactions, but higher strength is needed for good path coverage [29]. In other work, it is found that 100% of faults detectable by a relatively low degree of interaction, typically 4-way combinations [30-32].

The National Institute of Standards and Technology (NIST) investigated the application of interaction testing for 4 application domains: medical devices, a Web browser, a HTTP server, and a NASA distributed database. Here, 95% of the actual faults involved 4-way interaction whilst all of the faults were detected with 6-way interaction [10, 33]. Younis and Zamli introduced a novel approach to use interaction testing for test data generator for reverse engineering of combinational circuit [34]. Unlike the NIST study, Younis and Zamli highlighted the requirement for high degree interaction test suite (i.e. t>6). All the aforementioned related work highlighted the potential of adopting interaction testing for both hardware and software evaluation. Another upcoming application of interaction testing is on gene interactions. Instead of having to run 20,000 experiments to see if two genes randomly chosen from the genome of a 20,000-gene organism interact, biologists might get by with only 10 to 50 experiments [35].

Considering the support for variable strength interaction, much useful effort is also emerging. Cohen et al. proposed the first model t-way strategy with variable strength based capability based on simulated annealing [36]. Although generating optimal test suites, this approach is very time consuming because all interaction elements needs to be analyzed exhaustively using binary search strategy. Wang et al. has extended the model proposed by Cohen et al [36] and proposed a more general strategy relying on two greedy algorithms. The first algorithm is based on one-test-at-a-time strategy while the other algorithm is based on in-parameter-order strategy [37]. Although useful as far as addressing the limitation of the Cohen's model in terms of the need for the interaction strength (t) involved to be disjoint, Wang et al approach produces non-optimized set for mixed parameter values. Recently, Chen et. al. have proposed a variant algorithm based on ant colony approach in order to support variable strength capability [38]. Similar to Cohen et al [36], this approach is also time consuming and supports low interaction strength 2<t<3. Apart from these approaches, new version of TVG [6] and PICT [39] also address variable strength capabilities, nonetheless, these tools often generate test set which are typically larger than other approaches.

V. Closing REmarks

Summing up, the research into t-way strategies opens up many challenging issues especially involving algorithms for variable strength interaction testing. Considering the possibility of saving that variable strength interaction testing brings to software development costs (whilst enhancing the fault finding capability of t-way testing) , it is worthwhile to strategize efforts to develop new algorithms and strategies for future use.

Acknowledgment

This research is funded by the generous fundamental grants – "Investigating Heuristic Algorithm to Address Combinatorial Explosion Problem" from Ministry of Higher Education (MOHE), Malaysia.

References

[1] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the In-Parameter-Order Strategy for Constructing Covering Arrays," *Jrnl. of Research of the NIST,* vol. 113, pp. 287-297., Oct. 2008.

[2] 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,* vol. 18, pp. 125-148, 2008.

[3] B. Jenkins, "Jenny Test tool", http://www.burtleburtle.net./bob/math/jenny.html.

[4] A. Hartman, T. Klinger, and L. Raskin, "WHITCH: IBM Intelligent Test Configuration Handler," IBM Haifa and Watson Research Laboratories, Apr. 2005.

[5] A. Williams, "TConfig Test Tool. ," School of Information Technology and Eng., University of Ottawa. http://www.site.uottawa.ca/~awilliam/.

[6] J. Arshem, "Test Vector Generator Tool (TVG)", http://sourceforge.net/projects/tvg.

[7] M.J. Klaib, K.Z. Zamli, N.M. Isa, M.I. Younis, and R. Abdullah, “G2Way – A Backtracking Strategy for Pairwise Test Data Generation”, in *Proc. of the 15th Asia-Pacific Software Eng. Conf.*, 2009, pp. 463-470.

[8] M. B. Cohen, "Designing Test Suites for Software Interaction Testing (PhD Thesis)," in *Comp. Science* Auckland: University of Auckland, 2004.

[9] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software Fault Interactions and Implications for Software Testing," *IEEE Trans. on Software Eng.,* vol. 30, pp. 418-421, 2004.

[10] D. R. Kuhn and V. Okun, "Pseudo Exhaustive Testing For Software," in *Proc. of the 30th NASA/IEEE Software Eng. Workshop*, 2006, pp. 25-27.

[11] 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 Eng. of Computer-Based Systems*, Tucson, AZ, 2007, pp. 549-556.

[12] M. I. Younis and K. Z. Zamli, "MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems," *ETRI Journal,* Vol. 32, February 2010, pp. 73-82.

[13] M. Grindal, J. Offutt, and J. Mellin, "Conflict Management when Using Combination Strategies for Software Testing," in *Proc. of 18th Australian Software Eng. Conf.*, 2007.

[14] M. B. Cohen, M. B. Dwyer, and J. Shi, "Interaction testing of highly-configurable systems in the presence of constraints.," in *Proc. of the Intl. Symp. on Software Testing and Analysis (ISSTA 2007)*, New York, NY, USA, 2007, pp. 129–139.

[15] R. C. Bryce and C. J. Colbourn, "Prioritized Interaction Testing for Pairwise Coverage with Seeding and Avoids," *Information and Software Technology Jrnl.,* vol. 48, pp. 960-970, 2006.

[16] M. Grindal, J. Offutt, and S. Andler, "Combination Testing Strategies: a Survey," *Software Testing, Verification and Reliability,* vol. 15, pp. 167-199, 2005.

[17] R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing," *Comm. of the ACM,* vol. 28, pp. 1054-1058, 1985.

[18] T. Berling and P. Runeson, "Efficient Evaluation of Multifactor Dependent System Performance Using Fractional Design," *IEEE Trans. on Software Eng.,* vol. 29, pp. 769–781, 2003.

[19] L. J. Lazi´c and D. Velaˇsevi´c, "Applying Simulation and Design of Experiments to the Embedded Software Testing Process," *Software Testing, Verification and Reliability,* vol. 14, pp. 257–282, 2004.

[20] L. White and H. Almezen, "Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences," in *Proc. Of the Intl. Sym. on Software Reliability Eng.*, Piscataway, NJ, 2000, pp. 110–121.

[21] A. M. Memon and M. L. Soffa, "Regression Testing of GUIs," in *Proc. of the 9th European Software Eng. Conf. (ESEC) and 11th ACM SIGSOFT Intl. Sym. on the Foundations of Software Eng. (FSE-11)*, Helsinki, Finland, 2003, pp. 118–127.

[22] C. Yilmaz, M. B. Cohen, and A. Porter, "Covering Arrays for Efficient Fault Characterization in Complex Configuration Spaces," *IEEE Trans. on Software Eng*., vol. 31, pp. 20–34, 2006.

[23] M. S. Reorda, Z. Peng, and M. Violanate, "System-Level Test and Validation of Hardware/Software Systems," in *Advanced Microelectronics Serie,* London: Springer-Verlag, 2005.

[24] D. T. Tang and C. L. Chen, "Iterative Exhaustive Pattern Generation for Logic Testing," *IBM Jrnl. Research and Development,* vol. 28, pp. 212-219, 1984.

[25] S. Y. Boroday, "Determining Essential Arguments of Boolean Functions " in *Proc. of the Conf. on Industrial Mathematics*, Taganrog, 1998, pp. 59-61.

[26] A. K. Chandra, L. T. Kou, G. Markowsky, and S. Zaks, "On Sets of Boolean n-Vectors with All k-Projections Surjective," *Acta Informatica 20,* vol. 20, pp. 103-111, October 1983.

[27] G. Seroussi and N. H. Bshouty, "Vector Sets for Exhaustive Testing of Logic Circuits," *IEEE Trans. on Information Theory,* vol. 34, pp. 513-522, 1988.

[28] Dumer, "Asymptotically Optimal Codes Correcting Memory Defects of Fixed Multiplicity," *Problemy Peredachi Informatskii,* vol. 25, pp. 3–20, 1989.

[29] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino, "Applying Design of Experiments to Software Testing," in *Proc. of the Intl. Conf. on Software Eng. (ICSE ’97)*, Boston, MA, 1997, pp. 205–215.

[30] D. R. Wallace and D. R. Kuhn, "Failure Modes in Medical Device Software: an Analysis of 15 Years of Recall Data," *Intl. Jrnl. of Reliability, Quality, and Safety Eng.,* vol. 8, 2001.

[31] D. R. Kuhn and M. J. Reilly, "An Investigation of the Applicability of Design of Experiments to Software Testing," in *Proc. of the 27th Annual NASA Goddard Software Eng. Workshop (SEW-27'02)*, 2002, pp. 91-95.

[32] M. I. Younis and K. Z. Zamli, "A Strategy for Automatic Quality Signing and Verification Processes for Hardware and Software Testing," *Advances in Software Engineering,* pp. 1-7, 2010.

[33] R. Kuhn, Y. Lei, and R. Kacker, "Practical Combinatorial Testing: Beyond Pairwise," *IEEE IT Professional,* vol. 10, pp. 19-23, June 2008.

[34] M. I. Younis and K. Z. Zamli, "Assessing Combinatorial Interaction Strategy for Reverse Eng. of Combinational Circuits," in *Proc. of the IEEE Symp. on Industrial Electronics and Applications (ISIEA 2009*), Kuala Lumpur, Malaysia, 2009.

[35] CALTECH, "Researchers Create New Matchmaking Service Computer System To Study Gene Interactions.," http://media.caltech.edu/press_releases/12805.

[36] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, and J. S. Collofello, "Variable Strength Interaction Testing of Components," in *Proc. of the 27th Annual International Computer Software and Applications Conf. (COMPSAC’03)* Dallas,TX, USA: IEEE Computer Society, 2003, pp. 413-418.

[37] W. Ziyuan, X. Baowen, and N. Changhai, "Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite," in *Proc. of the 8th International Conference on Quality Software*, 2008, pp. 155-160.

[38] X. Chen, Q. Gu, A. Li, and D. Chen, "Variable Strength Interaction Testing with an Ant Colony System Approach," in *Proc. of the Asia Pacific Software Eng. Conf. 2009 (APSEC 2009)*, 2009, pp. 160-167.

[39] J. Czerwonka, "Pairwise testing in real world: Practical extensions to test case generator," in *Proc. of 24th Pacific Northwest Software Quality Conf.*, Portland,Oregon, USA, 2006, pp. 419-430.