T-Way Strategies and Its Applications for Combinatorial Testing

[1] Rozmie R. Othman and [2] Kamal Z. Zamli

[1] School of Computer and Communication
Universiti Malaysia Perlis (UniMAP)
PO Box 77, d/a Pejabat Pos Besar
01007 Kangar, Perlis, Malaysia

[2] School of Electrical Engineering, Universiti Sains Malaysia,
Engineering Campus, Nibong Tebal
14300 Penang, Malaysia
[1] rozmie.razif.othman@gmail.com, [2] eekamal@eng.usm.my

ABSTRACT

The adoption of t-way strategies (also termed interaction testing) for combinatorial testing is the main focus of this paper. Unlike earlier work, this paper unifies the different possible use of t-way strategies including uniform interaction, variable strength interaction, and input output based relations. In order to help engineers make informed decision on the different use of t-way strategies, this paper highlights the current state-of-the-art on different t-way strategy implementations. In doing so, this paper also discusses a step-by-step example as practical application.

KEYWORDS

software testing, interaction testing, t-way strategies, combinatorial testing

1 INTRODUCTION

The demand for multi-functional software has grown drastically over the years. To cater for this demand, software engineers are forced to develop complex software with increasing number of input parameters. As a result, more and more dependencies between input parameters are to be expected, opening more possibilities of faults due to interactions. Although traditional static and dynamic testing strategies (e.g. boundary value analysis, cause and effect analysis and equivalent partitioning) are useful in fault detection and prevention [1], however they are not sufficiently effective to detect faults due to interaction. As a result, many researchers nowadays are focusing on sampling strategy that is based on interaction testing (termed t-way testing strategies where t indicates the interaction strength) [2].

In general, t-way testing strategies offer three types of interaction possibilities for generating the test data (i.e. uniform strength, variable strength, and input output based relations). Rather than giving the test engineers (as domain experts) the flexibility to choose amongst all interaction possibilities, some strategies dictate only uniform t-way interactions (e.g. GTWay [3, 4], IPOG [5], MC-MIPOG [6], TConfig [7], and Jenny [8]) while others impose on variable strength interaction (e.g SA [9], ACS [10] and VS-PSTG [11]). In fact, there are also strategies that prescribe interactions due to input-output based relationship (e.g. ReqOrder [12], Union and Greedy [13-15]).

Addressing the aforementioned issues (and help test engineers make informed decision on the different use of t-way strategies), this paper highlights the current state-of-the-art on different t-way strategy implementations. In doing so, this paper also demonstrates a step-by-step example as practical application.

The rest of this paper is organized as follows. Section 2 illustrates the running example for demonstrating uniform strength, variable strength, and input output based relations. Section 3 highlights the existing t-way strategies in the literature. Section 4 demonstrates a practical application for t-way strategies. Finally, section 5 gives our conclusions.

2 RUNNING EXAMPLE

To facilitate discussion, consider a running example of a Pizza Ordering Application as shown in Figure 1. In this application, there are 4 options (or parameters) for the user is to choose from namely the crust, the flavour, the toppings, and the deliveries. For each of the option, there are 2 selections (or values) available. For simplification, this pizza ordering options can be represented using symbolic values (see Table 1).

Table 1: Parameters and Values Conversion

Actual Parameters and Their Values	Symbolic Representations
Crust = {Classic Hand Tossed, Crunchy Thin}	A = {a1,a2}
Flavour = {Pepperoni Delight, Vegetarian}	B = {b1,b2}
Toppings = {Pineapple, Beef}	C = {c1,c2}
Deliveries = {Eat In, Take Away}	D = {d1,d2}

Conveniently, as seen in Table 2, the pizza option representation can also be translated into a table of 4 columns (or parameters) and 2 rows (or values).

Table 2: Base Data Values

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

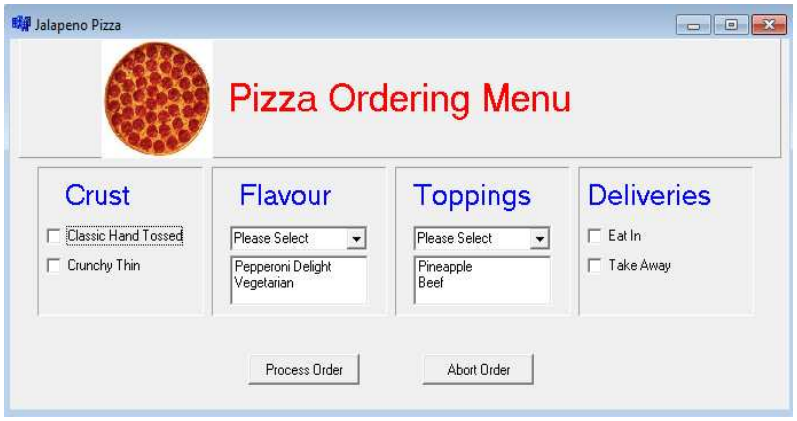


Figure 1: Pizza Ordering Application

Table 3: Exhaustive Combination

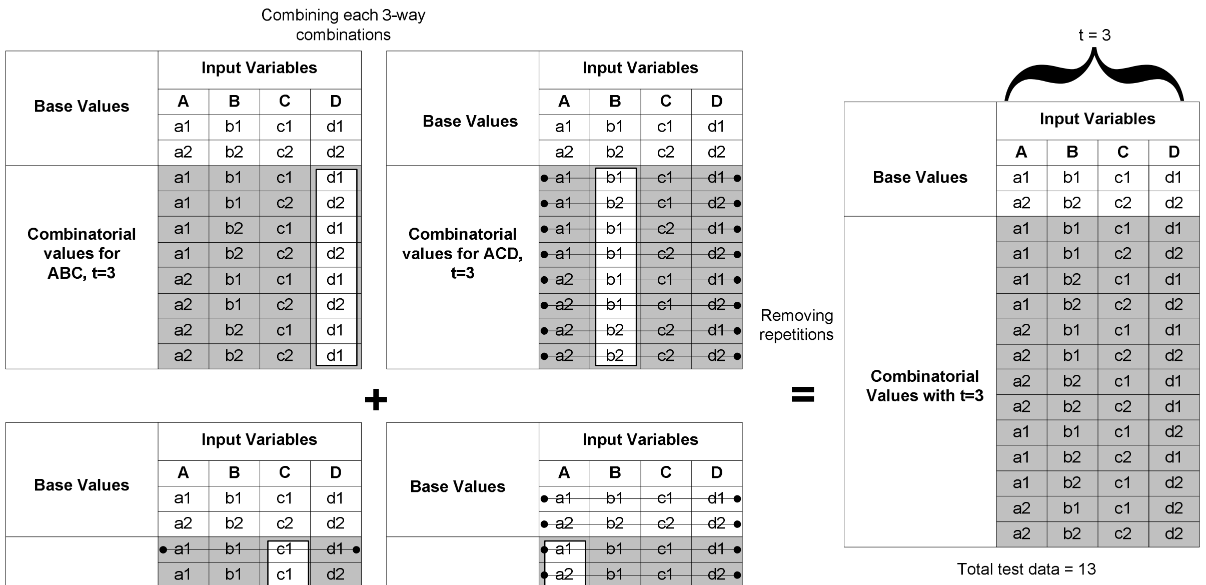
	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
All Combinatorial Values	a1	b1	c1	d1
	a1	b1	c1	d2
	a1	b1	c2	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c1	d2
	a1	b2	c2	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c1	d2
	a2	b1	c2	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c1	d2
	a2	b2	c2	d1
	a2	b2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

Here, at full strength of interaction (i.e. t=4), we can get all exhaustive combination. In this case, the exhaustive combinations would be $2^4 = 16$ (shown in Table 3).

The next sub-sections demonstrate the fact that by relaxing the interaction strength (t), the test data for testing consideration can be systematically reduced. In this case, the possible use of t-way strategies including uniform interaction, cumulative interaction, variable strength interaction, and input output relation based interaction will be demonstrated.

2.1 Uniform Strength T-way Interaction

Here, it is assumed that the interaction of variable is uniform throughout. Revisiting Table 2, and considering t=3, Figure 2 highlights how the reduction is achieved. Firstly, the interaction is



	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for ABC, t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1

	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for ACD, t=3	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c1~~	~~d2~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d1~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d2~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d1~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d2~~ ●

	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for ABD, t=3	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	a2	b1	c1	d2
	● ~~a2~~	~~b2~~	~~c2~~	~~d1~~ ●
	a2	b2	c2	d2

	Input Variables			
	A	**B**	**C**	**D**
Base Values	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d2~~ ●
Combinatorial values for BCD, t=3	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d2~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d1~~ ●
	● ~~a1~~	~~b1~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b2~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c1~~	~~d2~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d1~~ ●
	● ~~a2~~	~~b2~~	~~c2~~	~~d2~~ ●

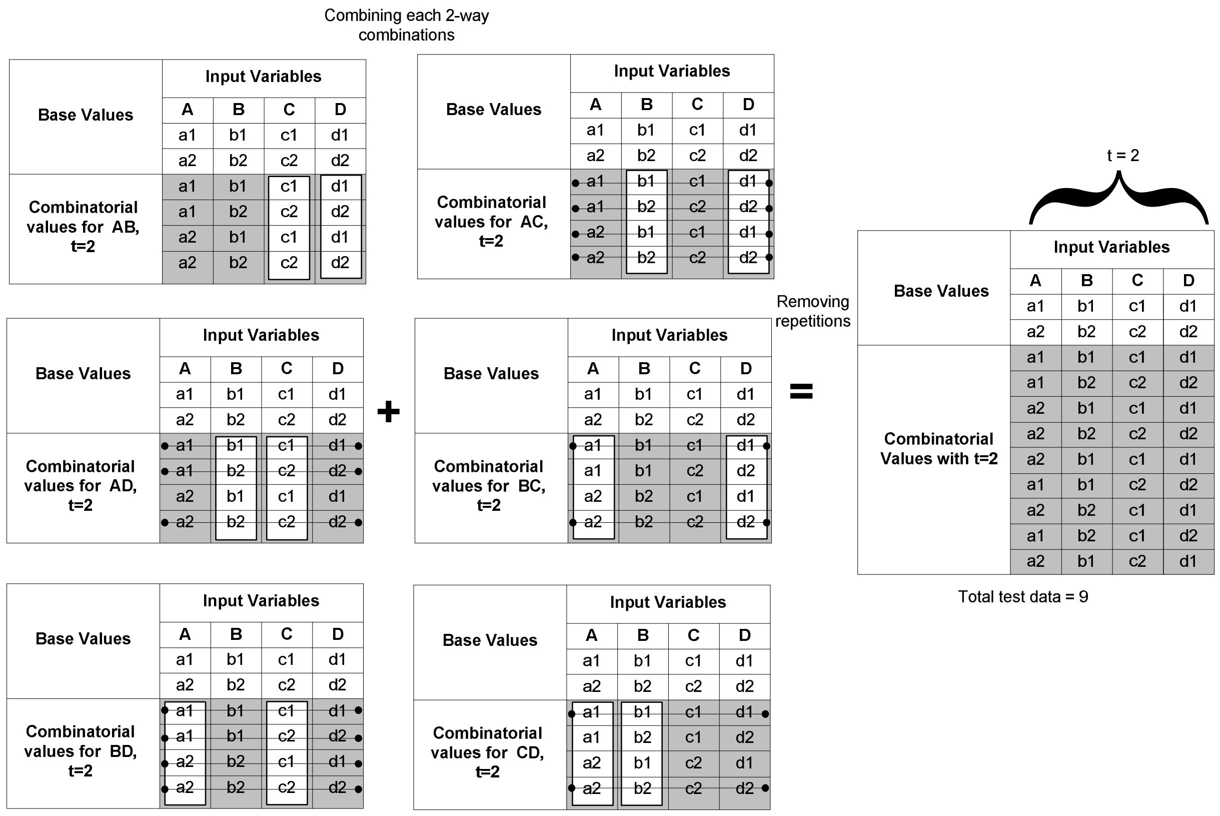
	Input Variables			
	A	**B**	**C**	**D**
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	a2	b1	c1	d2
	a2	b2	c2	d2

Figure 2: Uniform t-way Interaction Results (t=3)

broken down between parameters ABC, ABD, ACD, and BCD. Here, when parameters ABC are considered, the values for parameter D takes don't care value (i.e. any random valid values for parameter D suffices). Similarly, when parameters ABD are considered, values for parameter C takes don't care value. When parameters ACD are considered, values for parameter B takes don't care value. Finally, when parameters BCD are considered, values for parameter A takes don't care value. Combining these results, we note that there are some repetitions of values between some entries for ABC, ABD, ACD and BCD. If these repetition is removed, we can get all the combinations at t=3. Here, we note that the test suite has been reduced from 16 (for exhaustive combination) to 13 (for t=3), a saving of 18.75 percent.

2.2 Variable Strength T-way Interaction

Unlike uniform strength interaction counterparts, variable strength interaction considers more than one interaction strength in the test data generation process. Practically, a particular subset of input parameters can have a higher interaction dependency than other parameters (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 with t=2. In this case, we can assign variable coverage strength to each subset of components as well as to the whole system.



	Input Variables			
Base Values	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

	Input Variables			
Base Values	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

	Input Variables			
Base Values	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

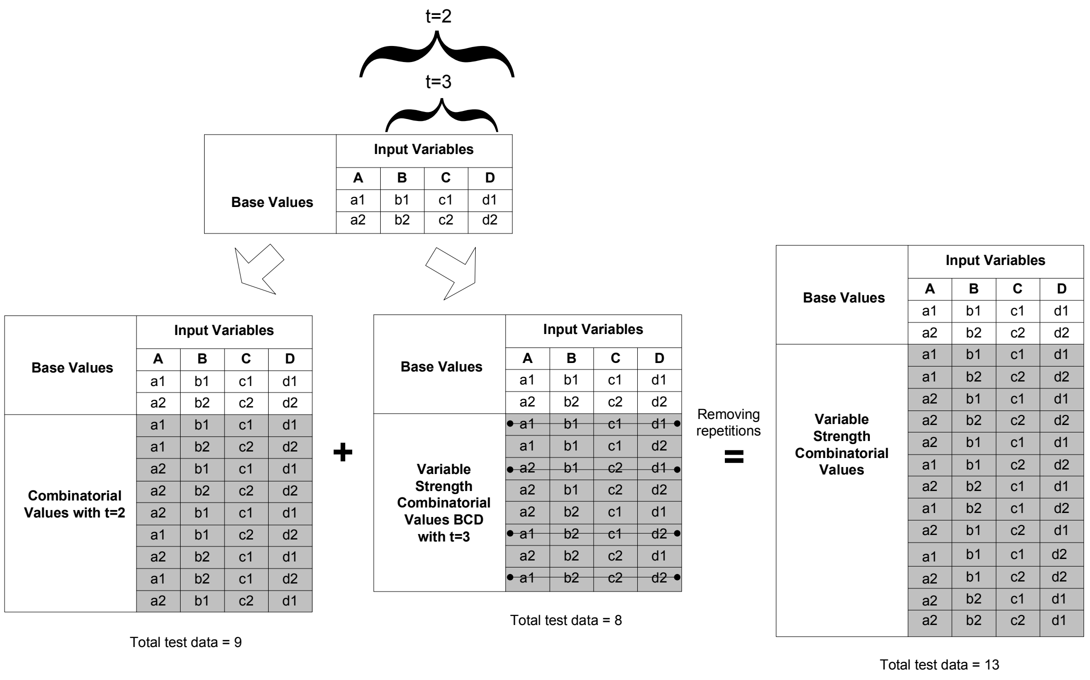
	Input Variables			
Base Values	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for BC, t=2	a1	b1	c1	d1
	a1	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d2

	Input Variables			
Base Values	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

	Input Variables			
Base Values	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

	Input Variables			
Base Values	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

Figure 3: Uniform t-way Interaction Results (t=2)



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

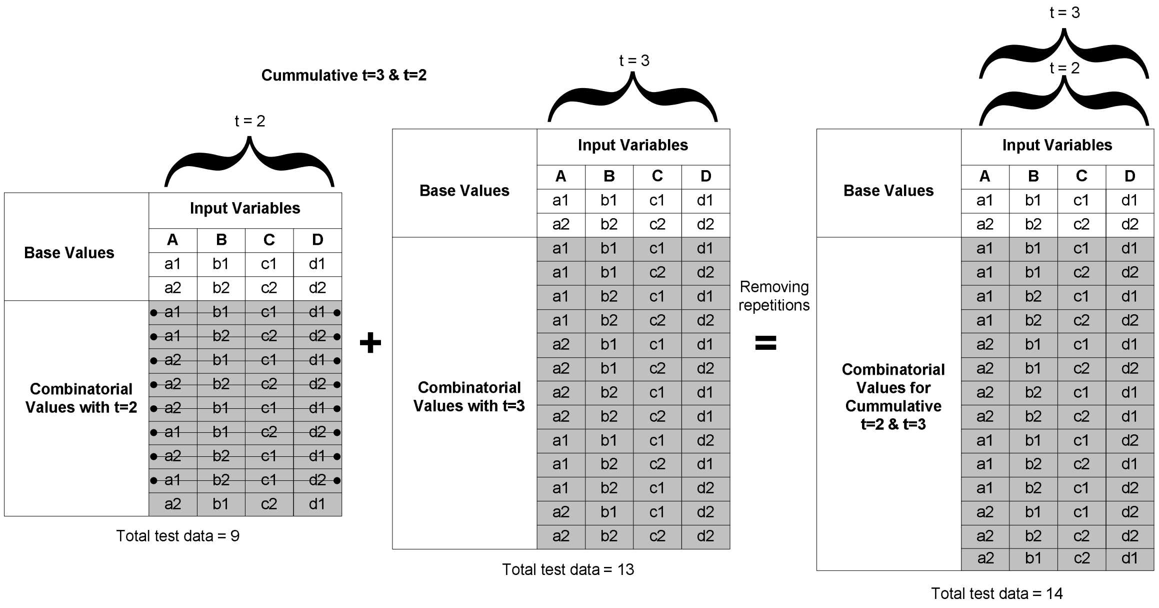
Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=2	a1	b1	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b2	c2	d2
	a2	b1	c1	d1
	a1	b1	c2	d2
	a2	b2	c1	d1
	a1	b2	c1	d2
	a2	b1	c2	d1

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~~

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

Figure 4: Variable Strength Interaction

To illustrate variable strength t-way interaction, we adopt the same example as Table 2. Now, we assume that all interaction is uniform at t=2 for all parameters (i.e. based on our result in Figure 3). Then, we consider t=3, only for parameters B,C,D. Combining both interactions yield result shown in Figure 4. Here, the test suite has been reduced



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

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

Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values for Cummulative t=2 & t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	a2	b1	c1	d2
	a2	b2	c2	d2
	a2	b1	c2	d1

Figure 5: Cummulative t=2 & t=3 Interaction Results

from 16 (for exhaustive case) to 13, a saving of 18.75 percent.

As a special case for variable strength interaction, we can also consider cumulative strength, t=3 and t=2. Revisiting Table 2, we can derive the test suite for t=2 using the same technique as t=3 (see Figure 5).

Combining the test suite with t=3, yields the following result (see Figure 6). Here, we note that T $_{suite}$ for t=2 is not necessarily a subset of T $_{suite}$ for t=3. In this case, the test suite has been reduced from 16 (for exhaustive case) to 14, a saving of 12.5 percent.

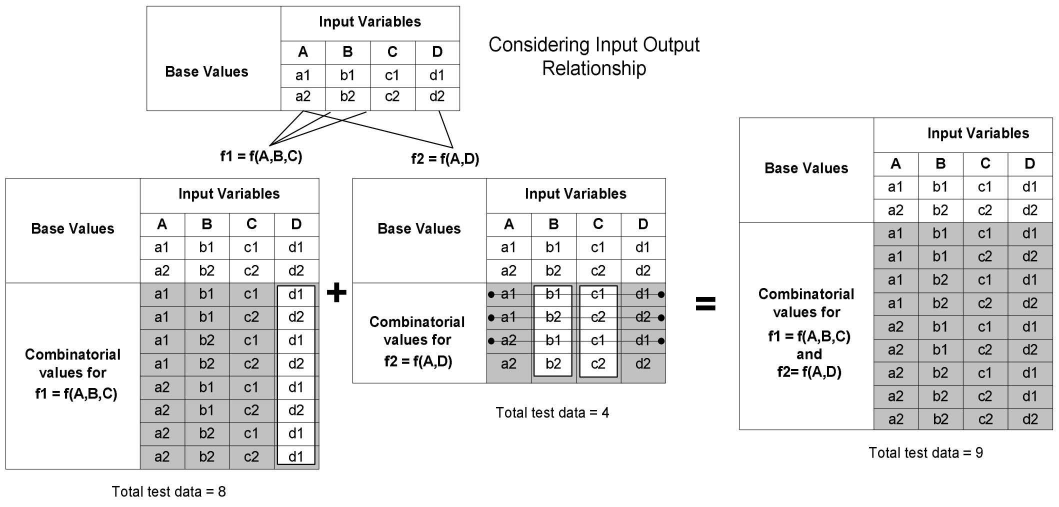
2.3 Input Output Relation Based Interaction

Similar to variable strength t-way interaction, input output relation based interaction does not deal with uniform interaction. Also, unlike other interaction possibilities discussed earlier, the reduction is performed by considering the knowledge on the input and output relationship amongst the parameter values involved (i.e. derived based on some statistical analysis such as Design of Experiments (DOE). In this manner, any input output relations based strategy implementation can address both uniform and variable strength interactions.

To illustrate the input output based interaction, we revisit Table 1 with the following input output relationship.

i. Only two outputs are considered, f1 an f2.
ii. f1 is a function of A,B,C, that is, f1=f(A,B,C).
iii. f2 is a function of A,D, that is, f2=f(A,D).

Ideally, these input output relationship are not to be assumed as they come from experimental results. Upon establishing these assumptions, we can derive the test suite accordingly. Figure 6 illustrates the complete results. Here, the test suite has been reduced from 16 (for exhaustive case) to 9, a saving of 43.75 percent.



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

	Input Variables			
	A	B	C	D
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for f1 = f(A,B,C)	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1

Total test data = 8

	Input Variables			
	A	B	C	D
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for f2 = f(A,D)	● ~~a1~~	~~b1~~	~~c1~~	~~d1~~ ●
	● ~~a1~~	~~b2~~	~~c2~~	~~d2~~ ●
	● ~~a2~~	~~b1~~	~~c1~~	~~d1~~ ●
	a2	b2	c2	d2

Total test data = 4

	Input Variables			
	A	B	C	D
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial values for f1 = f(A,B,C) and f2= f(A,D)	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a2	b2	c2	d2

Total test data = 9

Figure 6: Input Output Based Relations Interaction

3 OVERVIEW OF EXISTING T-WAY STRATEGIES

The main aim of any t-way strategies is to cover the interaction tuples of interest in an optimal manner (i.e. at most once whenever possible), hence, systematically minimizing the test cases for testing consideration. However, there is no unique solution to this NP-hard problem [16-18]. It is unlikely that a unique strategy exists that can always generate the most optimal number of test cases in every configuration.

A number of useful strategies have been developed from the last decade. The next subsections give some overview of existing t-way strategies based on the different type of interaction support given earlier (i.e. uniform strength interaction, variable strength interaction and input-output based relations).

3.1 Uniform Strength Based Strategy

As highlighted earlier, uniform strength interaction forms the basis of interaction testing, where all input parameters are assumed to be uniformly interacting (i.e. with constant interaction strength (t) throughout).

A significant number of work have focused on pairwise (t=2) strategies (e.g. Orthogonal Array Test System (OATS) [19], IRPS [20], AllPairs [21], In-Parameter-Order (IPO) [22], Test Case Generator (TCG) [23], OATSGen [24], ReduceArray2 [25], Deterministic Density Algorithm (DDA) [26], CTE-XL [27], rdExpert [28], and SmartTest [29]). As interaction strength is limited to t=2, pairwise strategies often yield the most minimum test set as compared other interaction. Although useful in some class of system, pairwise testing is known be ineffective for system with highly interacting variables [30-32]. For this reason, rather than dwelling on pairwise strategies, we are more interested on a general strategy for t-way test generation.

GTWay [3, 4] is uniform strength t-way strategy that relies heavily on its pair generation and backtracking algorithm. The pair generation algorithm first generates all the required interaction tuples for the specified interaction. Then, the backtracking algorithm iteratively traverses all tuples in order to combine mergeable tuples to form a complete test case in a greedy manner.

Jenny is also an open source uniform strength t-way strategy [8]. Jenny generates t-way test suite in stages where in the first stage; Jenny generates test case to cover all the 1-way interaction. Moving to the second stage, Jenny will extend the first stage test case to greedily cover the 2-way interaction pairs. This process will be repeated until n^{th} stage is reached where n is the interaction strength defined by user.

AETG [33, 34]. AETG is the first uniform t-way strategy that implements the pooling concept for test generation. To generate one test data, AETG will first randomly generate a number of test data candidates. From these test case candidates, the strategy will select one test data that covered the most uncovered tuples as the final test case. In case of “tie” situation, the strategy will randomly select one test data. “Tie” situation happens when more than one test data candidates covered the most uncovered tuples. To enhance its capability (e.g. for better test size), a number of variant AETG implementations have been implemented such as that of mAETG [35], and TCG [23]. The main difference between each AETG variants is on the method for the generating test data candidates.

Density [36, 37] is a uniform strength t-way strategy that is dependent on density calculation in order to determine the final test suite. Density strategy always starts with an empty test case. The "parameter density" for each unassigned parameter will be calculated and parameter with the highest parameter density will be selected. For the selected parameter, "value density" for each corresponded value will be calculated. The value with the highest value density will be fitted into selected parameter. The same process will be repeated until all parameters have valid assigned value. The detailed density determination can be found in [36].

IPOG [5] is based a novel one-parameter-at-a-time approach as in its pairwise predecessor IPO [22]. In IPOG, the interaction parameters will be generated first as the partial test suite based on the number of parameters and interaction value. The test suite is then extended with the values of the next parameters by using horizontal and vertical extension mechanism. Here, horizontal extension extends the partial test suite with values of the next parameter to cover the most interaction tuples. Upon completion of horizontal extension, vertical extension may be summoned to generate additional test cases that cover all uncovered interaction tuples. More recently, a number of variants have been developed to improve the IPOG's performance (i.e. IPOG-D [38], IPOG-F and IPOG-F2 [39]). In addition, other researchers also have come up with their own version of IPOG (i.e. Nie's version of IPOG called IPO_N [40], William's version of IPOG called TConfig [41]) and Younis's a number of version of parallel IPOG variants called MIPOG [42, 43], G_MIPOG, and MC-MIPOG.

GA [16] and GA-N [40] are two uniform t-way strategies that adopt genetic algorithm while ACA [16] is a uniform t-way strategy that implements ant colony algorithm. For genetic algorithm, the test data generation process always starts with random test cases (later refers as chromosomes). These chromosomes will undergo series of mutation processes until certain stopping criteria are met. The best chromosomes will be selected as final test suite. As for ant colony algorithm, the test data generation process is mimicking the colonies of ants travel from place to place (which representing the parameter) to find food (which represent the end of test case) via various route (which correspond to values for each parameter). The best route (measured based on the amount of pheromone left by colonies of ants) will represent the best value for a test case.

3.2 Variable Strength Based Strategy

SA [9] is perhaps the first variable strength t-way strategy in literature. Using probability-based transformation equations, SA adopts binary search algorithm to iteratively find the best test case from a large random search space. Although generating optimal test suites, this approach is very time consuming because all interaction elements needs to be analyzed exhaustively using binary search strategy.

ACS [10] is Chen version of test data generator that based on ant colony algorithm. Unlike ACA, ACS has the ability to generate variable strength test data suite. Implementation wise, ACS still based on ant colony algorithm to find the most optimized test suite.

VS-PSTG [11] is the most recent AI-based t-way strategy for generating t-way test suite. As the name suggests, VS-PSTG is based on the Particle Swarm Optimization (PSO) algorithm, which mimics the swarm behavior of birds. Internally, VS-PSTG iteratively performs local and global searches to find the candidate solution to be added to the final suite until all the interaction tuples are covered.

3.3 Input-Output Based Relations Strategy

Union and Greedy [13-15] are the first t-way strategies that adopts input output based relations. In the case of Union, the strategy generates the test suite for each output variable that cover all associated input interaction and then assign random value for all the 'don't care'. Then, the strategy finds the union of all test suites in order to reduce the number of generated test data.

Similar to Union, the Greedy strategy also generates the initial test suite that covered all associated input interaction by randomly selecting values for all don't care parameters. Nonetheless, unlike the Union strategy, the Greedy strategy picks only the unselected test case from the initial test suite which covers the most uncovered interactions as the final test suite. In this manner, the Greedy strategy often generates a more optimal test size than that of the Union strategy.

Test Vector Generator (TVG) [44] is a freeware tool that supports input output based relations. Little is known about TVG's implementation due to limited publications. Based on our experience with the tool, TVG appears to support three different reduction algorithms which are t-reduced, plus-one and random set. Comparatively, t-reduced algorithm often produces the most optimize test suite.

Integrated T-Way Test Data Generator (ITTDG) [45] generates a test case by iteratively adding the best parameter-value combination (i.e. a parameter value combination that covers the most uncovered tuples) until one complete test case is formed. In case of more than one best parameter-value combination found, ITTDG implements pooling concept introduces in AETG. The iteration continues until all tuples has been covered (and the complete test suite has been been formed).

Aura [46] implements a pooling concept for the generating final test suite. Here, Aura generates each test data candidates in a random manner. Unlike any other pooling based strategy, Aura gives the flexibility to the user to select the pool size. If a larger pool size is selected, the more optimized test suite will be produced but in the expense of execution time.

ParaOrder and ReqOrder [12] are two strategies based on IPOG [5] that address input output based relations. ParaOrder strategy implements horizontal and vertical extension for generating the final test suite, much like

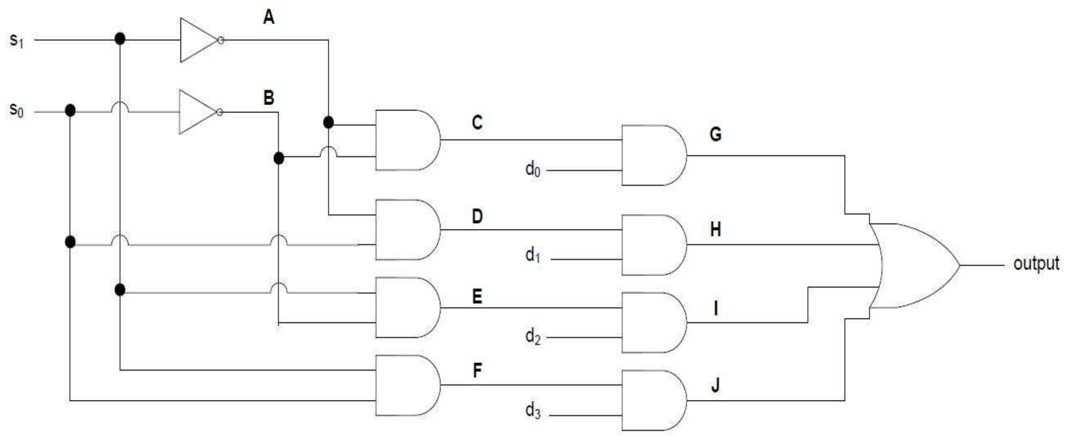


Figure 7: A Single Bit 4-to-1 Line Multiplexer

its predecessor IPOG [5]. The main difference between ParaOrder with IPOG is the fact that the initial test data (i.e. the initial exhaustive test data) for the former is generated based on the first defined input output relationships while the initial test case for the latter is generated in-defined-order-of-parameter found. In the case of ReqOrder, the selection of initial test case does not necessarily follow the first defined input output relationships rather the selection is done based on the highest input output relationship coverage.

4 PRACTICAL APPLICATIONS

In order to demonstrate the use of t-way strategies in practice, we adopt a reverse digital circuits consisting of a single bit 4-to-1 line multiplexer [31, 32, 47]. Here, the aim is to verify the MUX Java based software implementation using mutation testing based fault injection [48]. For this case study, we use ITTDG [45] as our test data generator and MuJava version 3 [49, 50] as our fault injection tool. The schematic diagram for the single bit 4-to-1 line MUX is shown in Figure 7 while the equivalent Java implementation can be found in Figure 8. Here, the Java implementation requires 6 parameters with each of which takes 2 vales (i.e. 6 2 valued parameters).

```
/* Multiplexer class will select input between d0, d1,
d2 and d3 as its output based on the value of selector
(s0, s1).
s1 = false, s0 = false => output = d0
s1 = false, s0 = true => output = d1
s1 = true, s0 = false => output = d2
s1 = true, s0 = true => output = d3
The symbol "!", "&&" and "||" represent logical
operator for NOT, AND and OR respectively
*/

public class Multiplexer {

public static String multiplex (boolean s0,
   boolean s1, boolean d0,
   boolean d1, boolean d2, boolean d3)
  {
   boolean A,B,C,D,E,F,G,H,I,J,output;
   A = !s1;
   B = !s0;
   C = A && B;
   D = A && s0;
   E = s1 && B;
   F = s1 && s0;
   G = C && d0;
   H = D && d1;
   I = E && d2;
   J = F && d3;
   output = G || H || I || J;
   return Boolean.toString(output);
   }
}
```

Figure 8: Equivalent Java Class For A Single Bit 4-to-1 Line MUX

It is important to make sure that software implementation follows exactly the hardware implementation so that the fault injection strategy can be used to represent fault in hardware implementation. In this experiment, we are to compare the effectiveness of uniform strength, variable strength, input output based relations for fault detection.

Now, we inject fault into the software implementation of MUX using MuJava. A total of 53 mutants have been generated which represent potential faults that might happen in hardware implementation. Next, we separately generate uniform strength test suite, variable strength test suite, and input output based relations test suite in order to kill the mutants using ITTDG implementation.

4.1 Uniform Strength Test Suite

We generate 5 uniform strength t-way test suite from t = 2 until 6 (i.e. exhaustive testing). The number of killed mutants by each test data is depicted in Table 4. Here, killed mutants represent faults that can be detected by the test data.

Table 4: Killed Mutants for Uniform Strength t-way Test Suite

Strength (t)	Test Data Size	Killed Mutants	% Killed Mutants
2	7	50	94%
3	12	53	100%
4	26	53	100%
5	32	53	100%
6	64	53	100%

From Table 4, all mutants can be killed completely using a 3-way test data. Thus, instead of running an exhaustive testing (which require 64 test cases), a 3-way testing (which consists of 12 test cases) is sufficient. Here, 81% of reduction has been achieved by using 3-way testing.

4.2 Variable Strength Test Suite

Here, we analyze the potential of applying variable strength interaction. As depicted in Table 4, it can be noticed from that 94% of mutants is killed at t=2 and 100% of mutants is killed at t=3. By judiciously input parameters with t=3 variable strength t-way test suite can be generated accordingly. Table 5 shows several variable strength configurations to test the same MUX and their percentage of mutants killed.

Result from Table 5 shows that using a pairwise testing (i.e. 2-way testing) for all parameters and 3-way testing for the first 4 parameters killed all mutants. Here we can see that, by using variable strength interaction, only 11 test cases are required to kill all the mutants (instead of 12 test cases as in the case of uniform interaction). Using variable strength interaction, 83% of reduction can be achieved (as compared to exhaustive testing).

Table 5: Killed Mutants for Variable Strength Test Suite

6 2-valued parameters, t=2			
Sub Strength	**Test Data Size, N**	**Killed Mutants**	**% Killed Mutants**
{s1,s0, d0} @ t=3	10	51	96%
{s1,s0, d0, d1} @ t=3	11	53	100%
{s1,s0, d0, d1, d2} @ t=3	12	53	100%
{s1,s0, d0, d1, d2, d3} @ t=3	12	53	100%

4.2 Input Output Relations Based Relation Test Suite

Analyzing Figure 7, we deduce that 5 groups of inputs are affecting the output. They are:-

- Group 1 includes input from s1, s0, d0.
- Group 2 includes input from s1, s0, d1.
- Group 3 includes input from s1, s0, d2.
- Group 4 includes input from s1, s0, d3.
- Group 5 includes input from d0, d1, d2.

Based on the above information, an input output based relations test suite is generated. The test size, number of killed mutants and percentage of killed mutants for input-output interaction are shown in Table 6.

Table 5: Killed Mutants for Input-Output Based Relations Test Suite

Test Data Size	Killed Mutants	% Killed Mutants
8	53	100%

Based on result shown in Table 5, we note that only 8 tests are required to kill all the mutants with 88% reduction (as compared to exhaustive testing).

5 CONCLUSIONS AND DISCUSSION

While the overall results suggest that input output based relations produce the smallest size test suite, this conclusion cannot be generalized to all other applications. The type of interaction is highly dependent on the problem at hand. It is the engineer's experience and knowledge on the system under test (SUT) that determines the best interaction to choose from.

As a rule of thumb, uniform strength interaction is summoned when no knowledge is known about the SUT. Variable strength interaction is useful when the effects of some sets of parameters are known to be significant to the overall operation of the SUT. As the name suggests, input output based relations interaction is helpful when the overall IO behavior of SUT can be established.

Summing up, this paper has presented three different types of interactions (i.e. uniform interaction, variable strength interaction and input-output based relations) that can be possibly been used for interaction testing. In addition, this paper also analyzes a number of existing t-way strategies based on types of interaction supported. Last but not least, this paper also elaborates on practical application where the use of different types of interactions is demonstrated within a single SUT. From the result, it can be concluded that in term of effectiveness, all three types of interactions can detect all errors (mutants) injected to the system.
Finally, while much useful research work has been done in the last decade (i.e. as evident by the large number of developed strategy implementations), the adoption of interaction testing for studying and testing real life systems has not been widespread [51]. In order to address this issue, more research into the algorithms and techniques are required to facilitate its adoption in the main stream of software engineering.

ACKNOWLEDGEMENTS

This research is partially funded by the generous MOHE fundamental grants – "Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique" and USM RU grants – "Development of Variable-strength Interaction Testing Strategy for T-Way Test Data Generation".

6 REFERENCES

1. Zamli, K. Z., Younis, M. I., Abdullah, S. A. C., Soh, Z. H. C.: *Software Testing*: First ed. KL Malaysia: Open University Malaysia, 2008.
2. Kuhn, D. R., Lei, Y., Kacker, R.: Practical Combinatorial Testing: Beyond Pairwise. IEEE IT Professional. 10 (3), 19-23 (2008)
3. 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. 181(9), 1741-1758 (2011)
4. Klaib, M. F. J.: Development Of An Automated Test Data Generation And Execution Strategy Using Combinatorial Approach. PhD. Thesis, School of Electrical And Electronics, Universiti Sains Malaysia, (2009)
5. Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., Lawrence, J.: IPOG: A General Strategy For T-Way Software Testing. In: Proceedings of the 14th Annual IEEE International Conference and Workshops on The Engineering of Computer-Based Systems, pp. 549-556, Tucson, AZ (2007)
6. Younis, M. I., Zamli, K. Z.: MC-MIPOG: A Parallel T-Way Test Generation Strategy For Multicore Systems. ETRI Journal. 32(1), 73-83 (2010)
7. Williams, A. W.: Software Component Interaction Testing: Coverage Measurment and Generation of the Configurations. Ph.D Thesis, School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada, (2002)
8. Jenny Test Tool, http://www.burtleburtle.net/bob/math/jenny.html
9. Cohen, M. B., Gibbons, P. B., Mugridge, W. B., Colbourn, C. J., Collofello, J. S.: Variable Strength Interaction Testing Of Components. In: Proceedings of 27th Annual International Computer Software and Applications Conference, pp. 413-418, Dallas, USA (2003)
10. Chen, X., Gu, Q., Li, A., Chen, D.: Variable Strength Interaction Testing With An Ant Colony System Approach. In: Proceedings of 16th Asia-Pacific Software Engineering Conference, pp. 160-167, Penang, Malaysia (2009)
11. Ahmed, B. S., Zamli, K. Z.: A Variable Strength Interaction Test Suites Generation Strategy Using Particle Swarm Optimization. Journal of Systems and Software*,* Article in Press (2011)
12. Wang, Z., Nie, C., Xu, B.: Generating Combinatorial Test Suite For Interaction Relationship. In: Proceedings of 4th International Workshop on Software Quality Assurance (SOQUA2007), pp. 55-61, Dubrovnik, Croatia (2007)
13. Schroeder, P. J.: Black-Box Test Reduction Using Input-Output Analysis. PhD Thesis, Department of Computer Science, Illinois Institute of Technology, Chicago, IL,USA, (2001)
14. Schroeder, P. J., Faherty, P., Korel, B.: Generating Expected Results For Automated Black-Box Testing. In: Proceedings of 17th IEEE International Conference on Automated Software Engineering (ASE'02), pp. 139-148, Edinburgh, Scotland, UK (2002)
15. Schroeder, P. J., Korel, B.: Black-Box Test Reduction Using Input-Output Analysis. SIGSOFT Software Engineering Notes. 25(5), 173-177 (2000)
16. Shiba, T., Tsuchiya, T., Kikuno, T.: Using Artificial Life Techniques To Generate Test Cases For Combinatorial Testing. In: Proceedings of the 28th Annual Intl.Computer Software and Applications Conf. (COMPSAC'04), pp. 72-77, Hong Kong (2004)
17. Younis, M. I., Zamli, K. Z., Klaib, M. F. J., Soh, Z. H. C., Abdullah, S. A. C., Isa, N. A. M.: Assessing IRPS As An Efficient Pairwise Test Data Generation Strategy. International Journal of Advanced Intelligence Paradigms. 2(3), 90-104 (2010)

18. Nie, C., Leung, H.: A Survey of Combinatorial Testing. ACM Computing Surveys. 43(2), (2011)
19. Krishnan, R., Krishna, S. M., Nandhan, P. S.: Combinatorial Testing: Learnings From Our Experience. ACM SIGSOFT Software Engineering Notes. 32(3), 1-8 (2007)
20. Younis, M. I., Zamli, K. Z., Isa, N. A. M.: IRPS: An Efficient Test Data Generation Strategy For Pairwise Testing. In: Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part I, pp. 493-500, Zagreb, Croatia: Springer-Verlag, (2008)
21. Allpairs Test Case Generation Tool, http://www.satisfice.com/tools.shtml
22. Lei, Y., Tai, K. C.: In-Parameter-Order: A Test Generation Strategy For Pairwise Testing. In: Proceedings of 3rd IEEE International Conference on High Assurance Systems Engineering Symposium, pp. 254-261, Washington DC, USA (1998)
23. Tung, Y. W., Aldiwan, W. S.: Automatic Test Case Generation For The New Generation Mission Software System. In: Proceedings of IEEE Aerospace Conference, pp. 431-437, Big Sky, MT, USA (2000)
24. Harrell, J. M.: *Orthogonal Array Testing Strategy (OATS) Technique*: Seilevel, Inc., 2001.
25. Daich, G. T.: Testing Combinations Of Parameters Made Easy [Software Testing]. In: Proceedings of IEEE Systems Readiness Technology Conference (AUTOTESTCON 2003),, pp. 379-384 (2003)
26. Colbourn, C. J., Cohen, M. B., Turban, R. C.: A Deterministic Density Algorithm For Pairwise Interaction Coverage. In: Proceedings. of the Intl. Conference on Software Engineering (IASTED 2004), pp. 345–352 (2004)
27. Lehmann, E., Wegener, J.: Test Case Design By Means Of The CTE XL. In: Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Copenhagen, Denmark (2000)
28. Copeland, L.: *A Practitioner's Guide To Software Test Design*. Massachusetts, USA: STQE Publishing, 2004.
29. SmartTest - Pairwise Testing, http://www.smartwaretechnologies.com/smarttestprod.htm
30. Kuhn, D. R., Wallace, D. R., Gallo, A. M.: Software Fault Interaction And Implication For Software Testing. IEEE Transaction on Software Engineering. 30(6), 418-421 (2004)
31. Younis, M. I., Zamli, K. Z.: Assessing Combinatorial Interaction Strategy For Reverse Engineering Of Combinational Circuits. In: Proceedings of the IEEE Symposium on Industrial Electronics and Applications (ISIEA 2009), Kuala Lumpur, Malaysia (2009)
32. Younis, M. I., Zamli, K. Z.: A Strategy For Automatic Quality Signing And Verification Processes For Hardware And Software Testing. Advances in Software Engineering. 1-7 (2010)
33. 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(7), 437-444 (1997)
34. AETG Web, http://aetgweb.argreenhouse.com/pricing.shtml
35. Cohen, M. B.: Designing Test Suites For Software Interaction Testing. PhD Thesis, School of Computer Science, University of Auckland, (2004)
36. Bryce, R. C., Colbourn, C. J.: A Density-Based Greedy Algorithm For Higher Strength Covering Arrays. Software Testing, Verification and Reliability. 19(1), 37-53 (2009)
37. Bryce, R. C., Colbourn, C. J.: The Density Algorithm For Pairwise Interaction Testing. Software Testing, Verification and Reliability. 17(3), 159-182 (2007)
38. Lei, Y., Kacker, R., Kuhn, R., Okun, V., Lawrence, J.: IPOG/IPOG-D: Efficient Test Generation For Multi-Way Combinatorial Testing. Journal of Software Testing, Verification and Reliability. 18(3), 125-148 (2008)
39. Forbes, M., Lawrence, J., Lei, Y., Kacker, R., Kuhn, D. R.: Refining The In-Paramater-Order Strategy For Constructing Covering Arrays. Journal of Research of the National Institute of Standards and Technology. 113(5), 287-297 (2008)
40. Nie, C., Xu, B., Shi, L., Dong, G.: Automatic Test Generation For N-Way Combinatorial Testing. In: Reussner, R., Mayer, J., Stafford, J. A., Overhage, S., Becker, S., Schroeder, P. J. (eds.) Quality

of Software Architectures and Software Quality. vol. 3712, pp. 203-211, Springer, Berlin / Heidelberg (2005)

41. TConfig, http://www.site.uottawa.ca/~awilliam/
42. Younis, M. I., Zamli, K. Z., Isa, N. A. M.: MIPOG - Modification Of The IPOG Strategy For T-Way Software Testing. In: Proceeding of The Distributed Frameworks and Applications (DFmA), Penang, Malaysia (2008)
43. Younis, M. I.: MIPOG: A Parallel T-Way Minimization Strategy For Combinatorial Testing. PhD. Thesis, School of Electrical And Electronics, Universiti Sains Malaysia, (2010)
44. TVG, http://sourceforge.net/projects/tvg
45. Othman, R. R., Zamli, K. Z.: ITTDG: Integrated T-way Test Data Generation Strategy for Interaction Testing. Scientific Research and Essays, Article in Press (2011)
46. Ong, H. Y., Zamli, K. Z.: Development of Interaction Test Suite Generation Strategy With Input-Output Mapping Supports. Scientific Research and Essays, Article in Press (2011)
47. Younis, M. I., Zamli, K. Z., Othman, R. R.: Effectiveness of the Cumulative vs. Normal Mode of Operation for Combinatorial Testing. In: IEEE Symposium on Industrial Electronics and Applications (ISIEA2010), pp. 350-354, Penang, Malaysia (2010)
48. Mano, M. M., Kime, C. R.: *Logic and Computer Design Fundamental*: Pearson Education International, 2004.
49. Ma, Y., Offutt, J., Kwon, Y.: MuJava: An Automated Class Mutation System. Journal of Software Testing, Verification and Reliability. 15(2), 97-133 (2005)
50. MuJava Version 3, http://cs.gmu.edu/~offutt/mujava/
51. Czerwonka, J.: Pairwise Testing In Real World. In: Proceedings of 24th Pacific Northwest Software Quality Conference, pp. 419-430, Portland, Oregon, USA (2006)